UNIVERSIDAD TÉCNICA DEL NORTE

INSTITUTO DE POSTGRADO

MAESTRÍA EN INGENIERÍA DE SOFTWARE

"ANÁLISIS DE FRAMEWORKS DE DESARROLLO DE API REST Y SU IMPACTO EN EL RENDIMIENTO DE APLICACIONES WEB CON ARQUITECTURA SPA."

Trabajo de Investigación previo a la obtención del Título de Magíster en Ingeniería de Software

DIRECTOR:

Mgs. Mauricio Xavier Rea Peñafiel

AUTOR:

Ing. Daisy Gabriela Valencia A.

IBARRA - ECUADOR

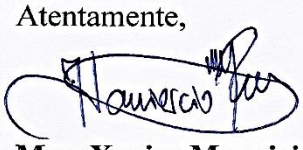
2018

APROBACIÓN DEL TUTOR

Ibarra, 11 de mayo del 2018

Yo, Mgs. XAVIER MAURICIO REA PEÑAFIEL, con cédula de identidad Nro. 1002485744, tutor del Trabajo de Grado: "ANÁLISIS DE FRAMEWORKS DE DESARROLLO DE API REST Y SU IMPACTO EN EL RENDIMIENTO DE APLICACIONES WEB CON ARQUITECTURA SPA", presentado por la maestrante DAISY GABRIELA VALENCIA ALTAMIRANO, para optar por el grado de Magister en Ingeniería de Software, ha sido guiado y revisado periódicamente por lo que doy fe que dicho trabajo reúne los requisitos y méritos suficientes para ser sometido a presentación pública y evaluación por parte del jurado examinador que se designe.

Atentamente,



Mgs. Xavier Mauricio Rea Peñafiel
TUTOR

APROBACIÓN DEL ASESOR

"ANÁLISIS DE FRAMEWORKS DE DESARROLLO DE API REST Y SU IMPACTO EN EL RENDIMIENTO DE APLICACIONES WEB CON ARQUITECTURA SPA"

Elaborado por la maestrante**:** DAISY GABRIELA VALENCIA ALTAMIRANO.

Trabajo de grado de Maestría aprobado en nombre de la Universidad Técnica del Norte, por el Asesor.

Ibarra, 11 de mayo del 2018

Mgs. Diego Trejo España
CI:1002149290
ASESOR

AUTORÍA

Yo, DAISY GABRIELA VALENCIA ALTAMIRANO, con cédula Nro. 1003319264, declaro bajo juramento que el trabajo aquí descrito es de mi autoría, que no ha sido previamente presentado para ningún grado, ni calificación profesional, que he consultado referencias bibliográficas que se incluyen en este documento y que todos los datos presentados son resultados de mi trabajo.

Daisy Gabriela Valencia Altamirano
AUTORA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1003319264		
APELLIDOS Y NOMBRES:	VALENCIA ALTAMIRANO DAISY GABRIELA		
DIRECCIÓN:	ISLA SANTA CRUZ 3-11 Y TULCÁN.		
EMAIL:	gvalencia1991@hotmail.es		
TELÉFONO FIJO:	(06) 2545 283	**TELÉFONO MÓVIL:**	0986360388

DATOS DE LA OBRA	
TÍTULO:	ANÁLISIS DE FRAMEWORKS DE DESARROLLO DE API REST Y SU IMPACTO EN EL RENDIMIENTO DE APLICACIONES WEB CON ARQUITECTURA SPA
AUTOR:	VALENCIA ALTAMIRANO DAISY GABRIELA
FECHA: DD/MM/AAAA	11 / 05 / 2018
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	☐ **PREGRADO** ■ **POSGRADO**
TITULO POR EL QUE OPTA:	MAGISTER EN INGENIERÍA DE SOFTWARE
DIRECTOR / ASESOR:	**MGS. XAVIER REA PEÑAFIEL / MGS. DIEGO TREJO ESPAÑA**

2. CONSTANCIAS

La autora manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es la titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 11 días del mes de mayo de 2018

LA AUTORA:

Daisy Gabriela Valencia Altamirano

CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Yo, DAISY GABRIELA VALENCIA ALTAMIRANO, con cédula de identidad Nro. 1003319264, manifiesto mi voluntad de ceder a la Universidad Técnica del Norte los derechos patrimoniales consagrados en la Ley de Propiedad Intelectual del Ecuador, artículos 4, 5 y 6; en calidad de autor de la obra o trabajo de grado denominado "ANÁLISIS DE FRAMEWORKS DE DESARROLLO DE API REST Y SU IMPACTO EN EL RENDIMIENTO DE APLICACIONES WEB CON ARQUITECTURA SPA", que ha sido desarrollado para optar por el título de Magister en Ingeniería de Software, en la Universidad Técnica del Norte, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En mi condición de autor me reservo los derechos morales de la obra antes citada.

En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Técnica del Norte.

Daisy Gabriela Valencia Altamirano
CI: 1003319264
AUTORA

DEDICATORIA

Dedico este trabajo de tesis a Dios que me ha bendecido con todo lo que tengo y lo que soy.

A mi querida madre Rosita, gracias por amarme tanto que haces que mi vida sea más feliz.

A mi querido padre Luis, gracias por enseñarme y compartirme todo tu conocimiento.

Gaby

AGRADECIMIENTO

Expreso mi agradecimiento:

A las grandiosas personas que trabajan y trabajaron en el Instituto de Posgrado de la Universidad Técnica del Norte, durante el periodo de la maestría.

Al Mgs. Xavier Rea Peñafiel, Director de Tesis, por su apoyo, tolerancia, y orientación en este proyecto y durante todo el tiempo que le he conocido.

Al Mgs. Diego Trejo España, Asesor de Tesis, por su guía en la finalización del mismo.

Daisy Gabriela Valencia Altamirano

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

UNIVERSIDAD TÉCNICA DEL NORTE

INSTITUTO DE POSGRADO

PROGRAMA DE MAESTRÍA

"ANÁLISIS DE FRAMEWORKS DE DESARROLLO DE API REST Y SU IMPACTO EN EL RENDIMIENTO DE APLICACIONES WEB CON ARQUITECTURA SPA."

Autor: Daisy Gabriela Valencia Altamirano

Tutor: Mgs. Mauricio Xavier Rea Peñafiel

Año: 2018

RESUMEN

Para decidir el WBAF (Web Backend Application Framework) para el desarrollo de API REST, que mejor se ajuste a las necesidades y arquitectura de una aplicación web, es importante referirse al rendimiento de las aplicaciones. La alta demanda de una mejor funcionalidad y usabilidad en aplicaciones web puso en funcionamiento nuevas técnicas, con la arquitectura SPA utiliza la técnica de buscar datos asincrónicamente y actualizar el contenido de la página o el componente entero sin actualizarlo. El frontend debe tener un bajo acoplamiento a las API REST desarrolladas en el backend; en esta investigación planteamos si el framework de desarrollo de las API REST impacta en el rendimiento de la aplicación web con arquitectura SPA. Los WBAF analizados son: Express.js que es un web application framework para Node.js y el framework ASP.NET Core. Mediante el Mapping Study se determinó que las métricas Response Time y Throughput determinan el rendimiento de las aplicaciones web y la metodología de benchmark nos sirve para obtener información sobre el comportamiento del rendimiento de las API REST, con los frameworks Express.js y ASP.NET Core para posteriormente compararlos con respecto al rendimiento de la aplicación web. Como conclusión podemos decir que si impacta el framework de desarrollo de API REST sobre el rendimiento de ejecución de aplicaciones web con SPA solamente cuando es una cantidad elevada de usuarios concurrentes, caso contrario ante la percepción del usuario no impactaría el framework de desarrollo.

Palabras clave: Frameworks, Express.js, ASP.NET Core, Rendimiento, Single Page Application.

UNIVERSIDAD TÉCNICA DEL NORTE

INSTITUTO DE POSGRADO

PROGRAMA DE MAESTRÍA

"ANÁLISIS DE FRAMEWORKS DE DESARROLLO DE API REST Y SU IMPACTO EN EL RENDIMIENTO DE APLICACIONES WEB CON ARQUITECTURA SPA."

Autor: Daisy Gabriela Valencia Altamirano

Tutor: Mgs. Mauricio Xavier Rea Peñafiel

Año: 2018

ABSTRACT

The WBAF (Web Backend Application Framework) to use for the development of API REST, it's the best for architecture of a web application, it's important to refer to the performance of the applications. The high demand for better functionality and usability in web applications put into operation new techniques, with the SPA architecture the technique used is to search data asynchronously and update the content of the page or the entire component without updating it. The frontend must have a low coupling to the REST APIs developed in the backend; In this research we propose whether the development framework impacts the performance of the web application with SPA architecture. The WBAF analyzed are: Express.js with Node.js and the ASP.NET Core framework. The Mapping Study it was determined that the Response Time and Throughput metrics determine the performance of web applications and the benchmark methodology helps us to obtain information on the performance behavior of the REST APIs with the Express.js and ASP.NET Core frameworks to compare them with respect to the performance of the web application. In conclusion we can say that "YES" impacts the REST API development framework on the performance of web application execution with SPA, only when there is a high number of concurrent users, otherwise the user's perception would not impact the development framework.

Keywords: Frameworks, Express.js, ASP.NET Core, Performance, Single Page Application.

CAPÍTULO 1. INTRODUCCIÓN

1.1. Problema de investigación

1.1.1. Antecedentes

Con el advenimiento de las redes sociales, los usuarios utilizan estos sitios web simultáneamente; esto plantea el requisito de que los sitios web deben ser muy rápidos, escalables, un acceso 24/7 y con soporte de cualquier número de usuarios simultáneos sin degradación de rendimiento.

La WWW (*World Wide Web*) evolucionó desde la presentación de contenido estático sobre contenido creado dinámicamente hasta contenido legible por máquina. En las páginas web tradicionales, el cliente recupera la información del servidor y se requiere que espere a que se cargue todo el contenido, lo que ocurre de forma síncrona. En las aplicaciones web de hoy en día, las páginas suelen cargarse como una plantilla y enlazarse con diferentes componentes y sus modelos, y cuando el valor en el modelo cambia, la vista se actualiza automáticamente desde el servidor a través de las llamadas API (*Application Programming Interface*), que son desarrolladas en el *backend* de la aplicación. De acuerdo con la arquitectura de SPA (*Single Page Application*), el *backend* actúa como una interfaz y en su lugar pone mucha más responsabilidad e incrementa la complejidad según el número de componentes en el cliente (De Luca, Epicoco, Lezzi, & Aloisio, 2012).

A nivel mundial y en Latinoamérica, aunque no existan datos de empresas específicas, a partir del 2000 según detalla (API economy, 2015) se ha incrementado el uso de API, y una de ellas el API REST; a pesar de que en el Ecuador aún no es tendencia el uso de API REST, por ser una tecnología nueva. En las empresas financieras a nivel mundial consumen servicios de nuevos operadores oferentes de API como PayPal, Billtrust, Tilt, y Amazon.

Netflix es una de las empresas pioneras en los servicios REST, ya que recibe más de 5 mil millones de peticiones diarias a sus API públicas, también cabe mencionar que en la industria de viajes se incluye a British Airways, Expedia, TripIt, y Yahoo Travel, que han abrazado a las API y se abren a los desarrolladores externos, publicando sus servicios.

Según el Research Center (ProgrammableWeb, 2013) el aumento del uso de API se produce desde el 2005, entre sus categorías más destacadas se encuentran las áreas: sociales, financieros, negocios, mapeo, *e-commerce*, *government*, ciencias, mensajería, pagos, telefonía; las API son servicios que pueden ser consumidos por otras empresas para integrar información, también, entre las empresas internamente crean sus propios servicios, para integrar los diferentes sistemas. En la revista Technology Forecast (Baya, Gruman, & Parker, 2012) señala que el crecimiento del uso de las API REST es mayor ante otras API de integración, esto es por su simplicidad en el manejo de datos con JSON o XML, siendo mayormente usado JSON.

En el Gobierno Provincial de Imbabura, en la Dirección de Tecnología actualmente desarrollan el ERP institucional enfocado en una arquitectura SOA. Para el desarrollo de los servicios que comprende el ERP se distribuye en el *frontend* con SPA y el *backend* se desarrolla API REST orientados a datos.

1.1.2. Planteamiento del problema

El presente trabajo de investigación surge de la necesidad de decidir el WBAF (*Web Backend Application Framework*) para el desarrollo de las API REST, que mejor se ajuste a las necesidades y arquitectura del ERP Institucional del GAD Provincial de Imbabura.

Uno de los requerimientos no funcionales definidos del ERP institucional se refiere al rendimiento de la aplicación web. En la Prefectura de Imbabura la arquitectura del *frontend* es SPA, así las API REST desarrolladas en el *backend* de la aplicación del ERP institucional no solo tiene la demanda de rendimiento de las aplicaciones, sino también la importancia de las aplicaciones para proporcionar un buen tiempo de respuesta (Mesbah & van Deursen, 2007). En el tiempo de respuesta se calcula la latencia de solicitud, que es el tiempo que toma el servidor para responder a una petición específica (Bixby, 2012).

Para cubrir ese requerimiento es necesario decidir entre varios WBAF que se encuentran en el mercado del desarrollo web, y que pueda desarrollar API REST que respondan fácilmente a la integración con SPA en el *frontend.*

1.1.3. Formulación del problema

¿Qué impacto tiene el *framework* de desarrollo de API REST sobre el rendimiento de ejecución de aplicaciones web con SPA?

1.1.3.1. *Pregunta de investigación*

¿Cuáles son las métricas de evaluación del rendimiento de aplicaciones web desarrolladas con API REST y SPA?

1.1.3.2. *Variable e Indicadores*

Para el proyecto se identificado las siguientes variables:

- **Variable independiente. -** *Web backend application framework* (WBAF).
- **Variable dependiente. -** Rendimiento de aplicaciones web dinámicas con SPA.

Tabla 1. Operacionalización de la variable independiente

Variable	Conceptualización	Dimensiones	Indicadores	Sud Indicadores	Ítems Básicos	Técnicas e Instrumentos
***Web Backend Application Framework* WBAF**	Marco de trabajo que soporta el desarrollo del *backend* de una aplicación.	Eficiencia de desempeño	Comportamiento temporal	Latencia del API REST	¿Cuánto es el tiempo de respuesta de la API REST?	Observación del caso de prueba
			Capacidad	Número de usuarios concurrentes	¿Cuál es el límite máximo de usuarios concurrentes?	Observación del caso de prueba

Nota: Elaboración propia

Tabla 2. Operacionalización de la variable dependiente

Variable	Conceptualización	Dimensiones	Indicadores	Sub Indicadores	Ítems Básicos	Técnicas e Instrumentos
Rendimiento de las Aplicaciones web dinámicas con SPA.	Capacidad del tiempo de respuesta de las aplicaciones web que no requiere consultas extras al servidor para cargar y navegar entre las páginas.	Eficiencia de desempeño	Comportamiento temporal	Tiempo de carga de archivos html, js, css	¿Cuál es el tiempo de carga de archivos?	Observación del caso de prueba

Nota: Elaboración propia

1.2. Objetivos de la investigación

1.2.1. Objetivo general

Analizar el impacto del WBAF (*Web Backend Application Framework*) de API REST en el rendimiento de aplicaciones web dinámicas con SPA (*Single Page Application*) para definir el WBAF que mejor se adapte al desarrollo de aplicaciones web del Gobierno Provincial de Imbabura.

1.2.2. Objetivos específicos

- Construir las API REST que cumplan con un conjunto de requerimientos del ERP institucional en los *frameworks*: Express.js y ASP.NET Core.
- Diseñar un plan de pruebas de rendimiento, mediante criterios de evaluación para API REST y SPA.
- Comparar el rendimiento de aplicaciones web dinámicas con SPA con el *framework* Express.js y con el *framework* ASP.NET Core, mediante un *benchmark* de *frameworks* según los resultados del plan de pruebas.

1.3. Justificación

La importancia de esta investigación se fundamenta en que es un tema actual, porque desarrollar aplicaciones web modernas es un gran desafío para las empresas de software, ya que se someten a ciclos de desarrollo ultrarrápidos, debido a las necesidades de los clientes, los rápidos cambios de los requisitos, el gran desafío del mercado. En este enfoque se debe empezar a aplicar arquitecturas web que puedan proporcionar, mayor rendimiento y escalabilidad a las aplicaciones web (Shanker Maurya & Shankar, 2012).

La alta demanda de una mejor funcionalidad y usabilidad en aplicaciones web puso en funcionamiento nuevas técnicas. La técnica más importante en SPA es buscar datos asincrónicamente y actualizar el contenido de la página o el componente necesario sin tener

que actualizar toda la página. Esta actualización puede ser una actualización de componente parcial o un reemplazo completo de un componente con otro componente (Mikowski & Powell, 2013).

Como requerimiento no funcional del ERP Insitucional, es el óptimo rendimiento de la aplicación web. El rendimiento se puede medir por el número de solicitudes por segundo que una aplicación puede administrar para una configuración de hardware determinada. Las soluciones más eficientes requerirán menos recursos, especialmente en un banco de información de una institución pública, donde cada aumento porcentual en los costos de alojamiento puede tener un enorme impacto en los ingresos y la gestión de la institución (Koffel, Choosing A Web Framework, 2013).

La capacidad de respuesta es un ingrediente clave en la importancia del SPA en la baja latencia de la solicitud. Amazon hizo pruebas A / B sobre este asunto y los resultados mostraron que un aumento de latencia de 100 ms disminuyó las ventas con 1 %. Google notó durante los experimentos que un aumento de 0,5s en la solicitud de una búsqueda dio lugar a una disminución del 20 % en el tráfico y los ingresos (Linden, 2006).

En el Gobierno Provincial de Imbabura en el *frontend* del ERP institucional, tiene una arquitectura SPA con el *framework* Angular2, ya que el desarrollo del *frontend* con esta arquitectura y *framework* se encuentra en una tendencia de tecnologías web para el 2017 según blogs en linkedin, y el repositorio github. El *frontend* debe tener un bajo acoplamiento a las API REST desarrolladas en el *backend* con Express.js que es un *web application framework* para Node.js y con el *framework* ASP.NET Core, ya que en la institución se cuenta con licencias de Visual Studio. En este proyecto de investigación para la parte de desarrollo de las API REST y el *frontend* con SPA se propone utilizar el marco de trabajo SCRUM, pero con algunas adaptaciones realizadas por la Jefatura de Software y Web, que ha considerado

necesarias por ser implementada en una institución pública donde el *core* de negocio no es el desarrollo de software.

Dado que el desarrollo y mantenimiento del software es una gran inversión, y el *framework* utilizado a menudo no se puede cambiar, y como la institución desea crear software de alta calidad, es importante saber en qué *framework* se desea desarrollar las API REST.

¿Express.js permite crear API REST que tengan un tiempo de respuesta menor que ASP.NET Core, o sería al contrario o en la misma medida; con un *frontend* desarrollado con SPA? Los objetivos de este proyecto son llevar a cabo diferencias generales y similitudes de ambos enfoques construyendo el mismo grupo de requerimientos de API REST en Express.js como con ASP.NET Core para posteriormente compararlos con respecto al rendimiento de la aplicación web.

CAPÍTULO 2. MARCO REFERENCIAL

2.1. Antecedentes

En la actualidad, para la búsqueda de información se manejan repositorios de investigación en todo el mundo; para iniciar en la búsqueda de antecedentes de trabajos de tesis relacionados a las API REST y SPA se recurrió a repositorios de los centros de educación superior a nivel nacional, en lo cual no se encontró información que aporte a la investigación y ante lo expuesto se acudió a repositorios internacionales.

En el repositorio de Chalmers University Of Technology de Gotemburgo- Suecia, se encontró una tesis de maestría "*A schematic for comparing web backend application frameworks With regards to responsiveness and load scalability* "(Un esquema para comparar los *web backend application frameworks* respecto a la capacidad de respuesta y la escalabilidad de la carga) que diseño un esquema para evaluar y comparar los *frameworks* de aplicaciones web *backend* en lo que respecta a la capacidad de respuesta y escalabilidad de carga, los resultados parciales correlacionan con *benchmarks* llamados tipos de prueba. Estos tipos de prueba en combinación con métodos para visualizar los resultados constituyen el esquema. El esquema se evaluó a través de un ejemplo en ejecución en el que se compararon Express.js y .NET MVC. El esquema demostró ser exitoso ya que fue posible sacar una conclusión sobre estos marcos a partir de los resultados en el contexto de uso (Dosé & Lilja, 2015).

Y en el repositorio de la Auburn University de Auburn-Alabama, se encontró la tesis de maestría "Developing Single page application with best practices" (Desarrollo de *Single Page Application* con mejores prácticas) que introduce al desarrollo de SPA y cómo difieren de las aplicaciones web tradicionales, también examina diferentes formas de desarrollar un SPA y propone una manera de desarrollar un SPA utilizando las mejores prácticas. (Wilkhu, 2017)

El aporte de este proyecto de investigación es el de realizar una comparativa sobre el rendimiento de una aplicación web con SPA y la integración de API REST desarrolladas en WBAF específicos, y poder evaluar el *framework* que mejor se adapte a las necesidades del desarrollo del ERP del Gobierno Provincial de Imbabura.

2.2. Referentes teóricos

Esta sección explora y analiza el marco teórico del WBAF *(web backend application framework),* el desarrollo de API REST y el rendimiento del *frontend* con SPA (*Single Pages Application).* El estudio aborda los temas de discusión sobre los WBAF para el desarrollo de API REST y como estos inciden en el rendimiento de aplicaciones web con la arquitectura SPA. La teoría relevante sobre los conceptos claves de la investigación será usada para el desarrollo del *benchmark* de WBAF.

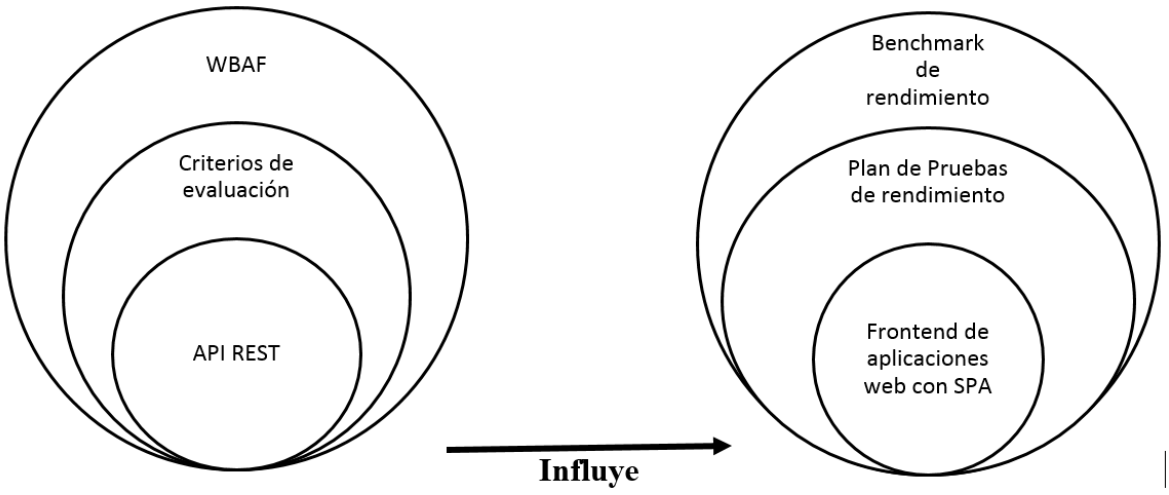


Figura 1. Categorías Fundamentales.
Nota: Elaboración propia

2.2.1. *Web backend application framework*

Una aplicación web comprende todos los elementos que son necesarios para que la solución basada en la web pueda satisfacer sus requisitos. Para la construcción de aplicaciones web existen varios *frameworks*. Un *framework* es una aplicación reutilizable, semi-completa que puede ser especializada para producir aplicaciones concretas y específicas. El *framework*

describe los objetos que componen el sistema, como éstos interactúan, y cuáles son sus responsabilidades (Johnson & Foote, 1988). De acuerdo a Islam, Islam, Islam, & Halim (2011) un *web application framework* (WAF) es un tipo de *framework* o base diseñado específicamente para ayudar a los desarrolladores a crear aplicaciones web. Estos *frameworks* típicamente proporcionan una funcionalidad básica común a la mayoría de las aplicaciones web, como la gestión de sesiones de usuario, la persistencia de datos y los *templates* de sistemas. Mediante el uso de un *framework* adecuado, un desarrollador a menudo puede ahorrar una cantidad significativa de tiempo en la construcción de una aplicación web.

Con base a la definición de WAF, en combinación con la demanda de los elementos de una aplicación web como: sistema operativo (OS), el servidor web, la plataforma, el WBAF y la implementación; se ha acuñado el término Web backend application framework (WBAF) para el framework de desarrollo de los servicios en el área de backend.

2.2.2. Criterios de Evaluación

Smith y Williams definen el rendimiento como el grado en que un sistema de software o componente satisface sus objetivos en el tiempo (2002). Hay muchas dimensiones de rendimiento, y dos importantes son la capacidad de respuesta y la escalabilidad. La capacidad de respuesta es la capacidad de un sistema para cumplir con sus objetivos de tiempo de respuesta. El tiempo de respuesta es el tiempo requerido para responder a un evento, y el rendimiento es el número de eventos procesados en una duración específica (Bass, Clements, & Kazman, 2003). La escalabilidad es definida por Bondi como la capacidad de un sistema para seguir cumpliendo su requerimiento de respuesta a medida que aumenta la demanda de la función de software (escalabilidad de carga) (1994).

En el contexto de API REST, el rendimiento se refiere al número de solicitudes que se procesan durante una duración específica, y el tiempo de respuesta se refiere al tiempo que tarda las API REST en responder a las solicitudes entrantes.

La escalabilidad de carga es la capacidad de seguir atendiendo solicitudes con el tiempo de respuesta requerido cuando se incrementa la cantidad de solicitudes concurrentes. Las API REST son más escalables de carga, por lo tanto, lo que resulta en un menor costo de alojamiento para la empresa (Koffel, 2013).

El rendimiento es una calidad omnipresente de los sistemas de software; todo lo afecta, desde el propio software a todas las capas subyacentes, como el sistema operativo, middleware, hardware, redes de comunicación, etc (Woodside, Franks, & Petriu, 2007). Estos elementos que afectan el desempeño de una entidad se denominan influencias en esta tesis. Estas influencias se caracterizan por una relación transitiva, por lo que el WBAF afectará el rendimiento de la aplicación web.

2.2.3. Web API

Web API o Servicio Web es una aplicación que puede ser accedido por una maquina en lugar de seres humanos, que utiliza el protocolo de aplicación HTTP, el estándar de nomenclatura URI y el lenguaje de marcado XML (Richardson & Ruby, 2007).

Fielding describe (2000) que, al diseñar una aplicación web, la arquitectura se puede elegir entre dos arquitecturas de API:

- RPC *(Remote Procedure Call*) por ejemplo SOAP basado en XML o JSON RPC basado en JSON.
- REST (*Representational State Transfer*)

Los servicios web que siguen la arquitectura REST se conocen a menudo como API REST. Es importante notar que esta arquitectura es más bien un conjunto de recomendaciones que un

estándar. Se utilizará el término API REST en su significado original como lo describe Fielding y los investigadores Pautsasso and Wilde (2009), la diferencia entre las arquitecturas basadas en SOAP y REST de la siguiente manera:

- SOAP: Enfoque prescriptivo "asumiendo mundos cerrados y relaciones contractuales". El enfoque se centra en la integración.
- REST: Enfoque descriptivo que "abastece a un mundo abierto con interacciones ad-hoc". El enfoque se centra en la cooperación.

2.2.3.1. *API REST*

REST *(Representation State Transfer)* es una arquitectura basada en red que Fielding y Row (2000) introdujo y definió por primera vez para proporcionar interoperabilidad entre sistemas informáticos en Internet. Una arquitectura REST se define por un conjunto de restricciones y sus recursos se identifican mediante URI (identificadores uniformes de recursos); estos recursos pueden representar objetos de datos como uno de varios formatos de datos como JSON o XML.

REST utiliza el protocolo HTTP como medio de transporte para crear servicios web ligeros, flexibles y escalables. No es un protocolo sino una idea de desarrollar servicios similares a la web y no está vinculado a una plataforma o tecnología específica. La propiedad clave de REST es la simplicidad que proviene del hecho de que REST utiliza el protocolo http de la tecnología web. Las cuatro operaciones básicas de REST se listan en la siguiente tabla:

Tabla 3. Operaciones básicas de REST.

Método HTTP	**Operación**
GET	Retornar datos.
POST	Crear nuevos datos
PUT	Actualizar datos
DELETE	Eliminar datos

Nota: Elaboración propia

En aplicaciones con API REST, el consumidor del servicio utiliza el método HTTP para definir la operación que se debe realizar. El proveedor del servicio maneja y procesa la solicitud y envía la respuesta al consumidor para informarle del resultado de la solicitud, mediante el código de estado de respuesta. Hay muchos códigos diferentes, pero los siguientes son los más comúnmente utilizados:

- 200 Ok: la respuesta HTTP estándar representa éxito en el retorno.
- 201 Created: Este código de estado debe ser devuelto cada vez que se crea la nueva instancia. Debe ser el resultado de la solicitud POST.
- 400 Bad Request: indica que la solicitud del cliente no se procesó, ya que el servidor no pudo entender lo que el cliente está pidiendo.
- 401 Unauthorized: indica que el cliente no puede acceder a los recursos y debe volver a solicitar con las credenciales requeridas.
- 403 Forbidden: indica que la solicitud es válida y el cliente está autenticado, pero el cliente no tiene permiso para acceder al recurso.
- 404 Not Found: indica que el recurso solicitado no está disponible ahora.
- 500 Internal Server Error: indica que la solicitud es válida, pero se produjo un error inesperado al procesar la solicitud. (Haldar, 2017).

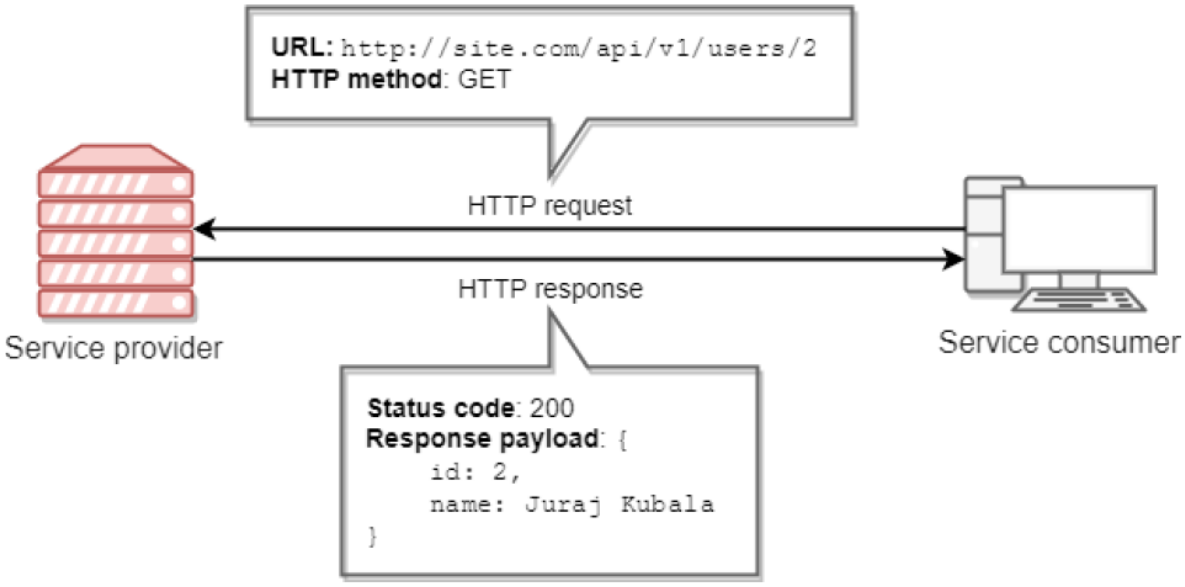


Figura 2.- Comunicación de la API REST entre el servidor y el cliente.
Nota: Tomado de Haldar (2017).

2.2.4. Frameworks backend

2.2.4.1. *Express*

En la documentación de Mozilla, (Del Pino, 2018) explica que Express es el *framework* web más popular de Node.js. Node.js es una plataforma de software que ayuda a crear aplicaciones de red asíncronas y orientadas a eventos. Contiene librerías del servidor HTTP integradas que permiten a los desarrolladores crear su propio servidor web y crear aplicaciones web altamente escalables. El motor de tiempo de ejecución de JavaScript V8 utilizado por Node.js es el mismo motor utilizado en el navegador Chrome de Google (Nodejs.org, 2018). El motor V8 compila el código directamente en el código nativo de máquina, dejando de lado el proceso de ejecución del interpretador, lo que da a Node.js un gran impulso en el rendimiento (Eloff & Torstensson, 2012). Además del motor V8, Node.js se compone de varios componentes, como se muestra en la figura 3.

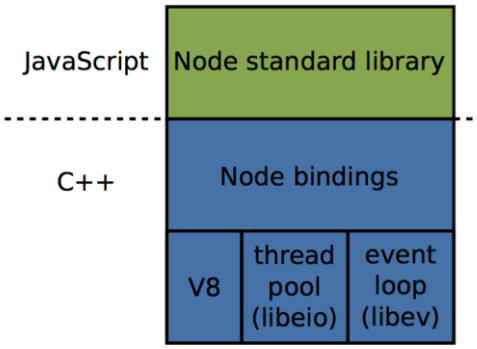


Figura 3.- Arquitectura de Node.js.
Nota: Tomado de Eloff y Torstensson *(2012)*

El motor V8 proporciona el entorno de tiempo de ejecución para la aplicación, y libeio maneja el grupo de subprocesos para realizar llamadas de E / S asíncronas (no bloqueantes) a libev, el bucle de eventos. (Eloff & Torstensson, 2012)

La función de E/S asíncrona sin bloqueo de Node.js juega un papel importante en la administración de recursos y la mejora del rendimiento de las aplicaciones Node.js. A diferencia de otros servidores mainstream como Apache e IIS, Node.js usa una operación de E/S sin bloqueo de un solo enlace. Lo que esto significa es que en lugar de ejecutar cada sesión (*request*) en un hilo separado y proporcionar una cantidad asociada de RAM para cada sesión, Node.js usa un único hilo para ejecutar todas las solicitudes e implementa un bucle de evento para evitar el bloqueo de E/S. En un servidor multiproceso, a medida que aumenta el número de subprocesos, la sobrecarga del servidor (programación y huellas de memoria) aumenta, lo que da como resultado un rendimiento lento del sistema en general. Node.js usa un enfoque de E/S basado en eventos y sin bloqueo para manejar todas las solicitudes al servidor (Erb, 2012).

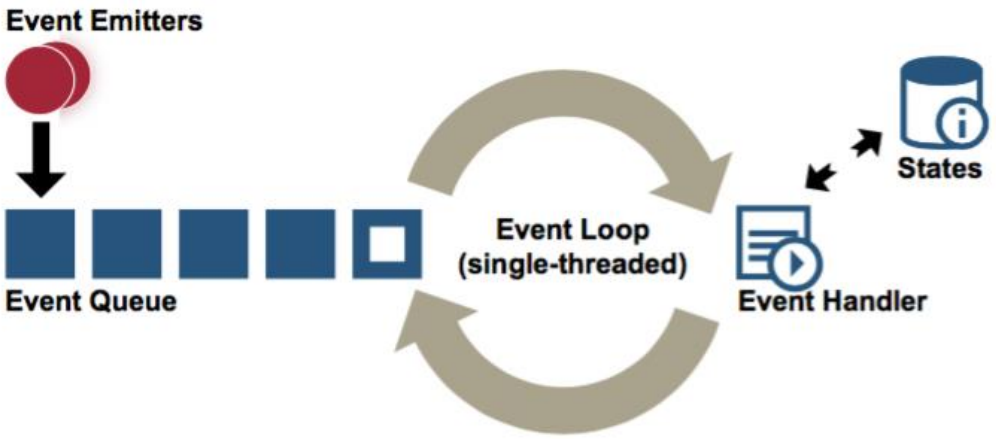


Figura 4.- Modelo del proceso de Node.js.
Nota: Tomado de Erb *(2012)*

En la figura 4, Node.js crea un bucle de eventos con manejadores de eventos para todas las solicitudes. Cuando se produce una operación de E/S, el controlador asociado se pone en cola para su ejecución y una función de devolución de llamada emite un evento después de que se completa la operación de E/S. Mientras tanto, otras operaciones de E/S se siguen ejecutando fuera del bucle de eventos del servidor. Por lo tanto, Node.js realiza las operaciones de E/S de forma asíncrona y no bloquea ninguna ejecución de scripts, lo que permite que el bucle de eventos responda a otras solicitudes (Erb, 2012).

Node.js tiene un rico ecosistema de desarrollo con varias librerías y gestores de paquetes compatibles. Uno de los principales gestores de paquetes que viene incluido previamente con Node.js durante la instalación es npm. Npm se ejecuta desde el intérprete de línea de comandos (CLI) y administra todas las dependencias para las aplicaciones Node.js. Los nombres de los módulos y las dependencias se pueden incluir con sus números de versión en el archivo 'package.json' dentro de la estructura de la carpeta, y los módulos se descargan emitiendo un simple comando 'npm install' desde la CLI (Herron, 2013, págs. 37 - 42).

Npm maneja automáticamente todo el proceso de descarga y guarda todos los módulos nombrados en la carpeta 'node-modules'. Los módulos se pueden actualizar cambiando su

número de versión en 'package.json', y se pueden distribuir fácilmente cargando un único archivo 'package.json' en el servidor en lugar de cargar la carpeta grande 'node-modules'. Después de la actualización o carga, la emisión del comando 'npm install' instala los módulos y dependencias específicos. Por lo tanto, las aplicaciones de Node.js son livianas, flexibles y fáciles de compartir (Herron, 2013, págs. 43 - 47).

Express es un módulo de nodo que proporciona un marco mínimo y flexible para las aplicaciones web Node.js. Funciona sobre los módulos de nodo central sin ocultar ninguna de las funciones de Node.js. Además, proporciona funciones robustas y limpias para agregar a los módulos de nodos, por lo que el desarrollo de la aplicación Node.js usando Express es mucho más fácil que usar los módulos de nodo nativos (Express official website). Debido a la simplicidad de Express, ha sido adoptado por grandes compañías como MySpace, Paypal, Apiary.io, Persona y Ghost. El uso de Express framework en la parte superior de Node.js ayuda a mantener la claridad del código. También hace que la integración del módulo sea fácil de manejar y proporciona una estructura de solución para las aplicaciones (Ihrig & Bretz, 2015, pág. 142). Express se instala utilizando el administrador de paquetes npm que emite el comando "npm install express" (Express official website).

Un servidor Express se compone de tres bloques de construcción: enrutador, rutas y middleware. La funcionalidad principal de un servidor web depende de sus excelentes métodos de enrutamiento. En una comunicación de cliente servidor, un cliente solicita algunos recursos del servidor, el servidor localiza los recursos y responde enviando los recursos al cliente. Esta es la funcionalidad principal de un servidor web y requiere excelentes métodos de enrutamiento para atender la solicitud. Express hace que este tedioso trabajo sea realmente fácil al permitir a los desarrolladores crear rutas en una estructura simple. Una ruta en Express es una combinación de un verbo HTTP y una ruta. El verbo HTTP generalmente es uno de los cuatro

métodos HTTP: GET, POST, PUT y DELETE, y la ruta es la ubicación del recurso (URI) (Ihrig & Bretz, 2015, págs. 144-146).

Middleware en Express son las funciones que tienen la función de patrón (req, res, next). REQ es la solicitud entrante del cliente, RES es la respuesta del servidor y NEXT es la función de devolución de llamada. Por lo tanto, las funciones de middleware ejecutan cualquier código dentro de él, manejan los objetos de solicitud y respuesta, finalizan el ciclo *request-response* y llaman a la siguiente función de middleware. Una función de middleware actual siempre debe llamar a la siguiente función de middleware, incluso en el caso de un ciclo de *request-response* incompleto para evitar que se cuelgue la solicitud (Express official website) (Ihrig & Bretz, 2015). En la figura 5 se muestra un servidor web Express simple que contiene los tres bloques de construcción.

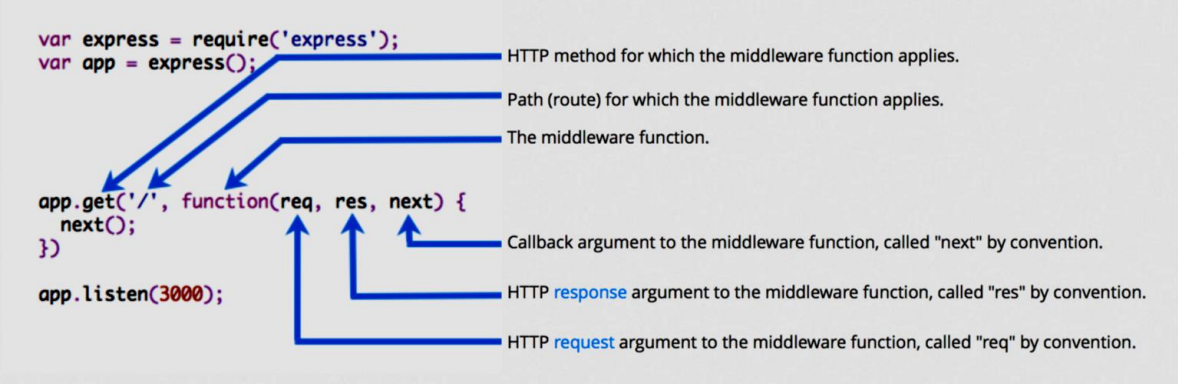


Figura 5.- Servidor web Express.
Nota: Tomado de Express official website *(Express official website)*

La Figura 5 ilustra los tres bloques de construcción de una aplicación Express: enrutador, ruta y middleware. Las primeras dos líneas usan el método Node.js require () para cargar el módulo express en la aplicación creando el objeto de la aplicación. La tercera línea es un enrutador simple con una ruta a la ubicación '/' y una función de middleware. La aplicación escucha el puerto 3000 para cualquier solicitud.

2.2.4.2. ASP.NET Core

ASP.NET Core es un *framework* web, open-source, multiplataforma, de alto rendimiento pensado para crear aplicaciones web modernas, basadas en la nube, conectadas a Internet.

ASP.NET Core proporciona los siguientes beneficios según (Roth, Anderson, & Luttin, 2017):

- Una historia unificada para crear UI web y API web.
- Integración de marcos de trabajo del cliente modernos y flujos de trabajo de desarrollo.
- Un sistema de configuración listo para la nube y basado en el entorno.
- Inyección de dependencia incorporada.
- Una canalización de solicitud HTTP ligera, de alto rendimiento y modular.
- Posibilidad de alojar en IIS, Nginx, Apache, Docker o auto-host en su propio proceso.
- Versiones de aplicaciones lado a lado cuando se orienta a .NET Core.
- Herramientas que simplifican el desarrollo web moderno.
- Posibilidad de compilar y ejecutar en Windows, macOS y Linux.
- De código abierto y centrado en la comunidad.
- Maneja paquetes NuGet para optimizar las aplicaciones.

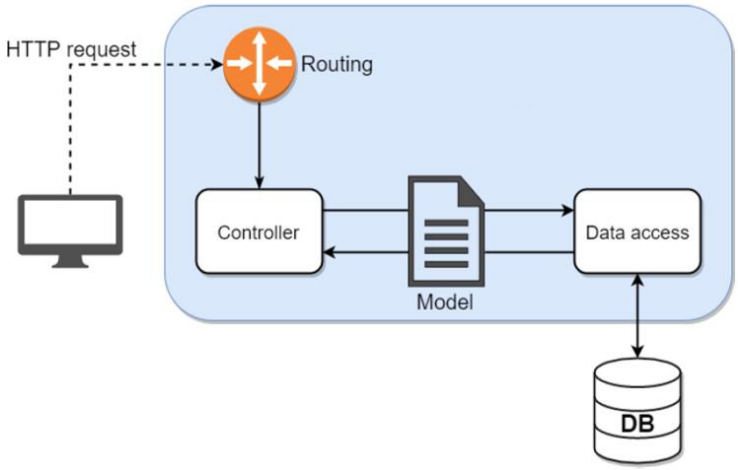


Figura 6.- Arquitectura de ASP.NET Core.
Nota: Tomado de Roth, Anderson, & Luttin (2017)

2.2.5. Desarrollo de Frontend con SPA

Ali Mesbah y Arie van Deursen (2007) definen "la interfaz de una *single-page* web está compuesta de componentes individuales que se pueden actualizar / reemplazar de forma independiente, de modo que no es necesario volver a cargar toda la página en cada acción del usuario".

La definición incluye una serie de atributos clave que ayudan a definir un SPA:

- Interfaz web: utilizada en la web, se centra en las interfaces de usuario.
- Componentes individuales: se divide en componentes más pequeños que se relacionan entre sí.
- Actualiza y reemplaza: un componente puede cambiarse o reemplazarse en cualquier momento por otro componente.
- Recarga: toda la página nunca se vuelve a cargar, aunque se puede cargar contenido nuevo en algunas secciones.
- Acciones del usuario: el SPA es responsable de manejar las acciones del usuario, como la entrada desde el mouse y el teclado.

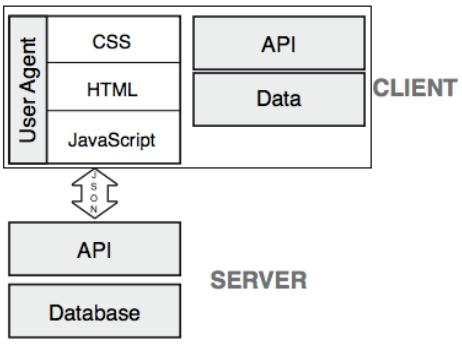


Figura 7. Arquitectura Single Page Application

Nota: Tomado de Peacock *(2000)*.

Al ejecutar Javascript en SPA, el navegador primero realiza una solicitud al servidor web. El servidor web responderá con el cliente de Javascript, incluidos los recursos necesarios para ejecutarlo. Una vez que se transfiere el cliente, se inicializará y estará listo para ejecutarse. Cuando el usuario interactúa con el cliente, como hacer clic en elementos gráficos, esto puede requerir la obtención de nuevos datos del servidor. En lugar de volver a cargar toda la página, el cliente hará una pequeña solicitud a la API en el servidor web. La API es simplemente una interfaz que permite a los clientes comunicarse con el servidor en un formato de datos que es fácil de entender tanto para el cliente como para el servidor (De Luca, Epicoco, Lezzi, & Aloisio, 2012).

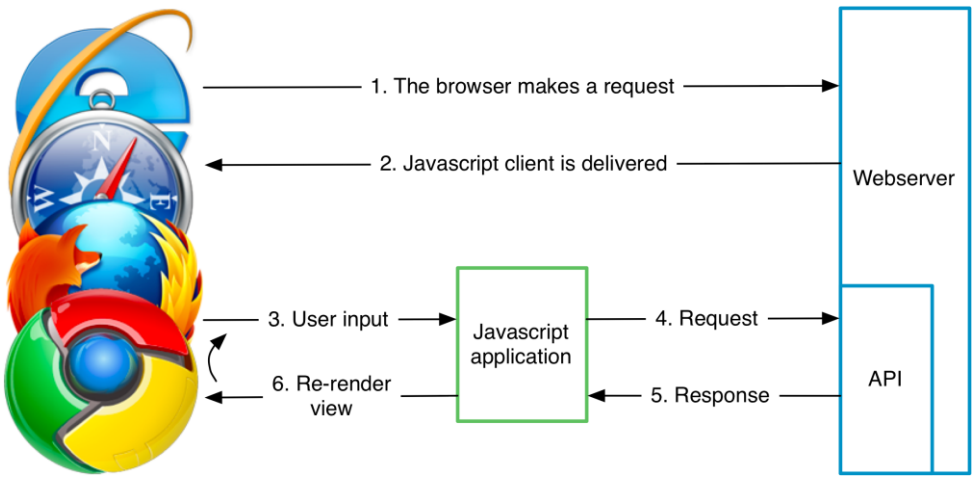


Figura 8. Flujo de comunicación entre el navegador y el servidor web.

Nota: Tomado de De Luca, Epicoco, Lezzi, & Aloisio (2012).

2.2.5.1. Angular Framework

El desarrollo del cliente con arquitectura SPA, es con Angular Framework para crear una aplicación más rápida y fácil de usar. El SPA traslada gran parte de la lógica de presentación y negocios al navegador, pero el servidor sigue siendo muy importante.

El objetivo principal de Angular es simplificar la creación de aplicaciones cliente en HTML y JavaScript o en un lenguaje como TypeScript que se compila en JavaScript. El framework consta de varias bibliotecas, algunas de ellas son centrales y otras son opcionales. El desarrollador escribe una aplicación angular con templates de componentes en HTML con angular, se desarrolla clases de componentes para administrar estas plantillas, agrega lógica de aplicaciones a los servicios y componentes en módulos (Angular, 2018).

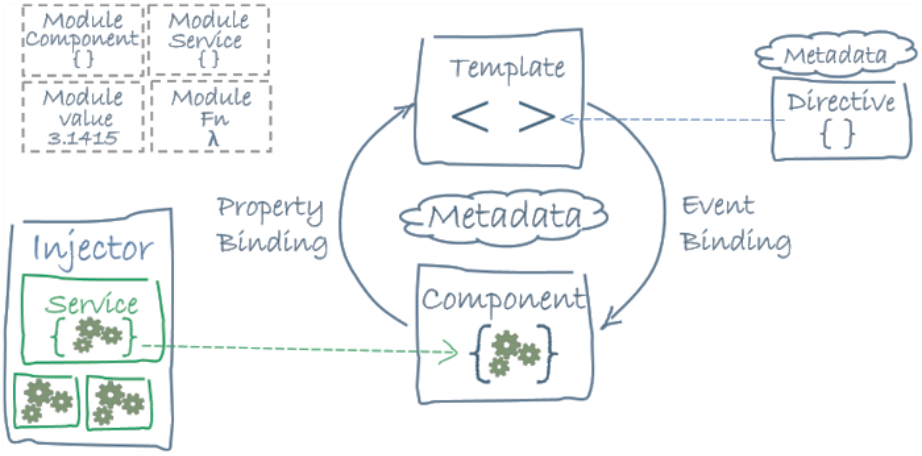


Figura 9.- Arquitectura de Angular Framework.

Tomado: Del sitio de Angular *(2018)*

2.2.6. Pruebas de Rendimiento

Alonso Amo, Martínez Normand, y Segovia Pérez (2005), afirman que las pruebas tienen como propósito evaluar el tiempo de repuesta del sistema o memoria que ocupa, de acuerdo a esta afirmación Tuya Gonzalez, Ramos Román y Dolado Cosín (2007, pág. 43) afirman que las pruebas determinan si los tiempos de respuesta del sistema tanto en condiciones normales como en condiciones especiales, se encuentran dentro de los limites predefinidos; y Pressman (2010),

menciona la importancia de estas pruebas que prueban el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado, porque según es inaceptable que un software proporcione las funciones requeridas, pero no se ajuste a los requisitos de rendimiento.

Según Patil y Joshi (2012), el objetivo de la prueba de rendimiento es verificar el rendimiento del sistema especificado (por ejemplo, tiempo de respuesta, disponibilidad del servicio). Donde se ejecuta simulando cientos o más accesos de usuarios simultáneos durante un intervalo de tiempo definido. La información sobre los accesos se registra y luego se analiza para estimar los niveles de carga que agotan los recursos del sistema. Para las aplicaciones web, el rendimiento del sistema es un problema crítico porque a los usuarios de la Web no les gusta esperar demasiado para responder a sus solicitudes, también esperan que los servicios estén siempre disponibles.

2.2.6.1. *Prueba de Benchmark*

En este proyecto de investigación se aplica la prueba de Benchmark, que es un tipo de pruebas de rendimiento. Cuando el objetivo es obtener información sobre el comportamiento de rendimiento de una aplicación determinada, una posible solución es realizar una prueba de benchmark (Aversano, Rak, & Villano, 2013), (Rak, Turtur, & Villano, 2015) (Rak, y otros, 2013) (Cuomo, Rak, & Villano, 2015) (Li, Zong, Kandula, Yang, & Zhang, 2011) (Sharma, Shenoy, Sahu, & Shaikh, 2011) (Liew & Su, 2012) concuerdan que los resultados de las pruebas de benchmark de aplicaciones a menudo alimentan herramientas de predicción del rendimiento, basadas en simulaciones o modelos analíticos, que permiten predecir el efecto final de múltiples opciones de configuración e implementación y para realizar un ajuste efectivo del rendimiento de las aplicaciones.

2.2.7. Fundamentación Filosófica

El proyecto propuesto se enmarca en el paradigma critico propositivo; de acuerdo con (Herrera, Medina, & Naranjo, 2008), el paradigma crítico porque analiza y cuestiona una realidad socio-tecnológica de la problemática que se está investigando; y propositivo por cuanto busca plantear una alternativa de control al problema planteado en esta investigación.

2.2.8. Fundamentación Legal

La normativa legal entorno al desarrollo del proyecto de investigación se centra en la generación de tecnología que aporte al crecimiento tecnológico del país. Para sintetizar lo expuesto se presenta el siguiente análisis:

- El Plan Nacional de Gobierno electrónico 2014 – 2017 en base a la Carta Iberoamericana de Gobierno electrónico (2013), formula 12 principios que precautelan el derecho de los ciudadanos a relacionarse con el Estado electrónicamente. Como el:
    - "Principio de adecuación tecnológica: Garantiza que las administraciones elegirán las tecnologías más adecuadas para satisfacer sus necesidades, por lo que se recomienda el uso de estándares abiertos y de software libre en razón de la seguridad, sostenibilidad a largo plazo y la socialización del conocimiento."
- La Constitución de la República del Ecuador (2008) que garantiza la soberanía nacional, y define los sectores estratégicos entre los cuales están las tecnologías como hardware y software:
    - "Art. 322. Se reconoce la propiedad intelectual de acuerdo con las condiciones que señale la ley. Se prohíbe toda forma de apropiación de conocimientos colectivos, en el ámbito de las ciencias, tecnologías y saberes ancestrales…"
    - "Art. 385. El sistema nacional de ciencia, tecnología, innovación y saberes ancestrales, en el marco del respeto al ambiente, la naturaleza, la vida, las culturas y la soberanía, tendrá como finalidad…

- Desarrollar tecnologías e innovaciones que impulsen la producción nacional, eleven la eficiencia y productividad, mejoren la calidad de vida y contribuyan a la realización del buen vivir.

CAPÍTULO 3. MARCO METODOLÓGICO

3.1. Descripción del área de estudio

La investigación tiene una orientación al modelo de prototipado, y estará basado en las políticas de desarrollo de la Jefatura de Software y Web del Gobierno Provincial de Imbabura. Sin embargo, esta investigación podrá ser adoptada por cualquier organización que implemente API REST y SPA.

3.2. Diseño y tipo de investigación

La investigación es cuantitativa tal y como la definen Hungler, Tatano y Polit (2004) y (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2010), considerando que se centra en un pequeño número de conceptos, comienza con ideas preconcebidas sobre la forma en que los conceptos están relacionados, utiliza procedimientos estructurados y herramientas formales para la recolección de datos. Se basa en la recolección y análisis de datos y se supone que los métodos estadísticos hacen posible el estudio de la generalización (Mascarenhas, 2012) y utiliza un método estadístico para analizar datos recogidos e intenta explicar el objeto o fenómeno observado en función del comportamiento de las variables analizadas (Casarin & Casarin, 2012). Mediante la observación de los casos de prueba de los frameworks a utilizar se podrá generar una perspectiva teórica y la recolección de datos será a partir desde las perspectivas de los frameworks estudiados. El resultado de este trabajo debe ser el análisis del impacto de WBAF de API REST en el rendimiento de aplicaciones web con SPA.

3.2.1. Modalidad de Investigación

Para llevar a efecto la presente investigación se plantea la modalidad de investigación documental. Esta investigación analiza artículos publicados sobre WBAF y API REST, SPA y temas de rendimiento de aplicaciones web.

Según (Yuni & Urbano, 2014) la utilización de esta estrategia metodológica de obtención de información aportará a conocer la realidad de los objetos estudiados en la realización de este proyecto, con el fin de interpretar el análisis de los elementos de esta investigación.

Para la obtención de información se utilizará Mapping Study porque nos permite responder a las preguntas de investigación, encontrando todos los resultados relevantes de la investigación de estudios empíricos "primarios" (Kitchenham & Charters, 2007). Este estudio tiene por objeto identificar la brecha en el conjunto de estudios primarios, en los que se requieren estudios primarios nuevos o mejores.

Petticrew y Roberts (2008) sugieren que tal estudio "implica una búsqueda de la literatura para determinar qué tipo de estudios abordan la cuestión de la revisión sistemática se han llevado a cabo, dónde se publican, en qué bases de datos se han indexado, qué tipo de resultados se han evaluado y en qué poblaciones ".

Las primeras etapas de un Mapping Study son en general muy similares a las de una revisión sistemática de la literatura, aunque es probable que la propia investigación sea mucho más amplia, a fin de abordar adecuadamente el alcance más amplio de dicho estudio. Estas tres etapas son:

1. Identificación de estudios primarios que puedan contener resultados de investigación pertinentes (búsqueda);
2. Seleccionar los estudios primarios apropiados de éstos después de un examen adicional (inclusión / exclusión);
3. Cuando proceda, realizar una evaluación de la calidad de los estudios seleccionados (sesgo / validez).

3.2.2. Tipos o Niveles de Investigación

El proyecto de investigación maneja variables y la relación que una incide en la otra, en el cual se podría aplicar el nivel correlacional y (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2010) explica que el nivel correlacional evalúa el grado de asociación entre dos o más variables, midiendo cada una de ellas y, después, cuantificar y analizar la vinculación; tales correlacionales se sustenta en hipótesis sometidas a prueba.

3.2.3. Métodos

Por la aplicación de una investigación cuantitativa en este proyecto de investigación, se apoya en el método inductivo, que (Yuni & Urbano, 2014) conceptualiza que la lógica inductiva es un tipo de razonamiento que comienza con la observación repetida de los fenómenos, y se puede llegar a conclusiones como resultado de la inferencia de similitudes observadas en los casos estudiados.

3.2.4. Estrategias técnicas

Al no existir una población o muestra, el procedimiento para obtener los resultados será mediante las pruebas ejecutadas en cada WBAF aplicado en los API REST y el rendimiento de la aplicación web con SPA, y también complementado con información obtenida de investigaciones anteriores.

3.3. Procedimiento de investigación

3.3.1. Técnica Mapping Study

En este trabajo hemos aplicado la técnica Mapping Study con la intención de responder a la siguiente pregunta de investigación:

¿Cuáles son las métricas de evaluación del rendimiento de aplicaciones web desarrolladas con API REST y SPA?

El objetivo consiste en identificar los criterios de evaluación del rendimiento de aplicaciones web. Para ello, se estableció la siguiente cadena de búsqueda:

("performance testing metrics" AND("api rest" OR "web application" OR "SPA")) .

La estrategia de búsqueda se basó en las siguientes decisiones:

- La cadena de búsqueda se adaptó a los siguientes motores de búsqueda:
    - Science@Direct (www.sciencedirect.com)
    - Springer Link (link.springer.com)
    - IEEE Xplore Digital Library (ieeexplore.ieee.org)
- Tipo de estudios considerados:
    - Artículos de conferencias, artículos de revistas.
- Periodo de publicación: Desde el año 2010.

Puesto que API REST es una tendencia donde el 90% de los artículos encontrados eran posteriores a 2010.

3.3.1.1. *Selección de estudios*

Los criterios de inclusión para la selección de los estudios primarios son los siguientes:

- Estudios que describen métricas de evaluación del rendimiento de aplicaciones web.
- Estudios que describen implementación de pruebas de rendimiento en aplicaciones web desarrolladas con API REST y/o SPA.

Se excluyeron los estudios que cumplían los siguientes criterios:

- Estudios que no responden de manera rigurosa a la pregunta de investigación.
- Estudios que no aportan ninguna propuesta o información relevante relativas a las aplicaciones web desarrolladas con API REST y SPA

Tras la selección inicial de estudios primarios, se llevó a cabo la evaluación de su calidad en dos etapas. Primero se revisó su idoneidad, teniendo en cuenta los criterios de inclusión y exclusión, y después se llevó a cabo una revisión más precisa en paralelo con el proceso de extracción de información. Durante este proceso, se verificó la relevancia y la calidad de los estudios, teniendo en cuenta la claridad de sus métodos y propuestas.

A través del procedimiento de búsqueda (detallado en la Figura 10) se encontraron 3 estudios primarios, listados en la Tabla 3.

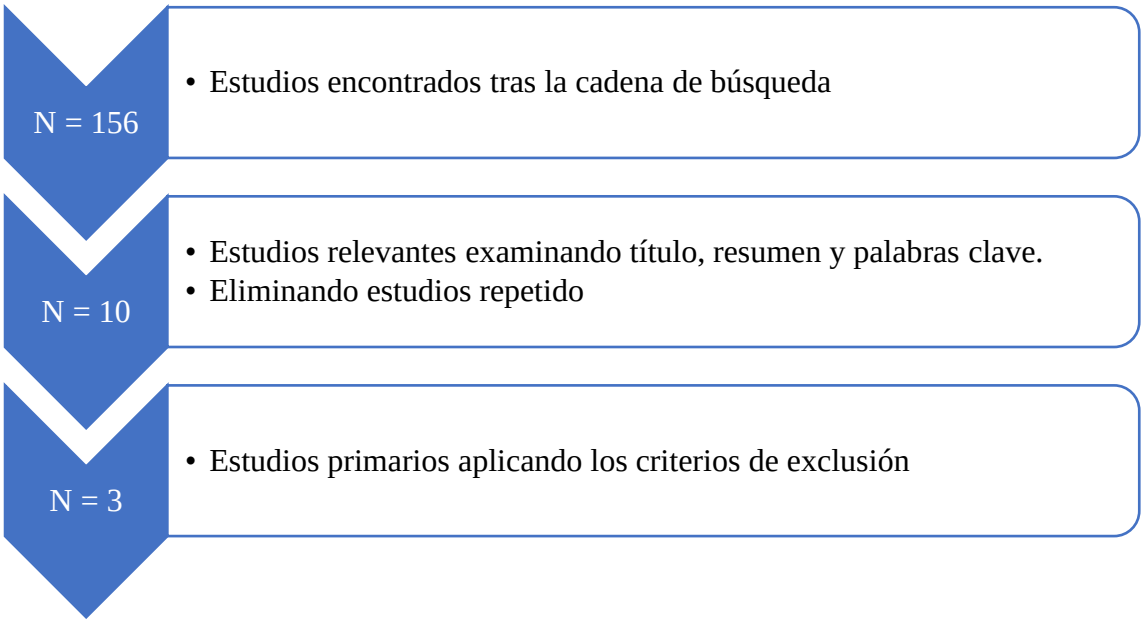


Figura 10. Procedimiento de búsqueda.

Tomado: Elaboración propia.

Tabla 4. Estudios primarios.

Autores y año publicación	**Título**	**Nro estudios primarios**	**Tipo de estudio**	**Fuente**
Zhu, Fu, & Li **(2010)**	Research the performance testing and performance improvement strategy in web application.	7	Experimental	IEEE
Freitas & Renata **(2014)**	An Ontology for Guiding Performance Testing	30	Caso de estudio	IEEE
Kalita, Khanikar, & Bezboruah **(2011)**	Investigation on performance testing and evaluation of PReWebN: a Java technique for implementing web application	25	Caso de estudio	IEEE

Nota: Elaboración propia

3.3.1.1.1. Zhu, Fu y Li - Research the performance testing and performance improvement strategy in web Application

La investigación realizada por (Zhu, Fu, & Li, 2010) analiza los tipos, indicadores y métodos de prueba de las pruebas de rendimiento de la web, esta investigación se enfocó en los siguientes indicadores.

- Tiempo de respuesta

El tiempo de respuesta también se conoce como el tiempo de espera desde el punto de vista del usuario. Es desde el momento que el cliente envía una solicitud para recibir la respuesta del servidor y se mide en unidades de tiempo. En general, es como la carga de usuarios de bajo nivel aumenta lentamente, pero una vez que uno o algunos recursos del sistema se han agotado, el tiempo de espera aumentará rápidamente.

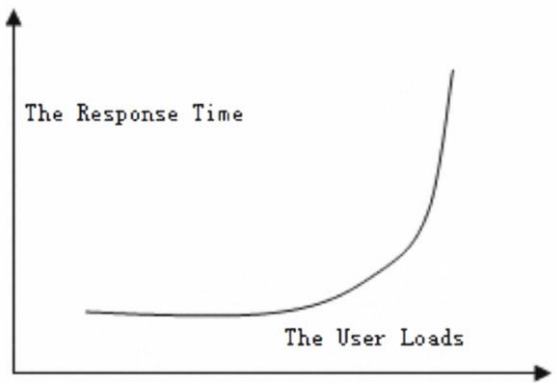


Figura 11. El tiempo de respuesta en relación con la carga de usuarios.

Tomado: De Zhu, Fu y Li (2010)

En la figura 11, se observa un aumento repentino en el tiempo de respuesta, a menudo se debe a que uno o varios recursos del sistema logran la máxima utilización. Por ejemplo, para configurar un servidor web que utiliza un numero de hilos fijo, maneja las solicitudes concurrentes de los usuarios y cuando el número de solicitudes concurrentes exceda el número

de hilos que el servidor puede pagar, cualquier solicitud entrante se colocará en la cola de solicitudes y esperará el procesamiento.

El período de tiempo de espera se divide en muchos fragmentos pequeños y estos fragmentos se dividen en dos tipos principales: latencia de red y latencia de aplicación. La latencia de red se refiere al tiempo de espera que los datos se transfieran de un servidor a otro. El tiempo de espera de la aplicación que consumen los datos procesados dentro de un servidor.

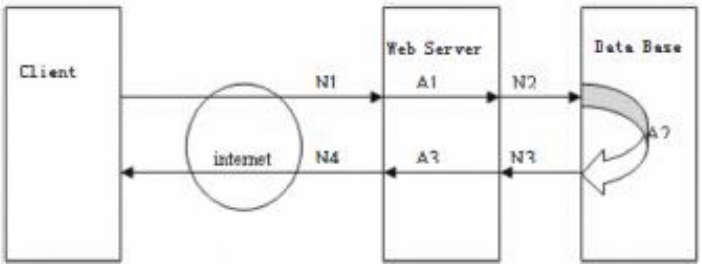


Figura 12. Curso de una típica solicitud web que procesa los diferentes tiempos de espera.

Tomado: De Zhu, Fu y Li (2010)

En la Figura 6 se muestra: Todo el tiempo de espera (tiempo de respuesta) = (NI + N2 + N3 + N4) + (AI + A2 + A3) donde Nx variable de la latencia de la red, Ax variable del tiempo de espera de la aplicación. El tiempo de respuesta suele ser la principal decisión de NI y N4, el tiempo de espera representa la forma en que los clientes acceden a Internet. La latencia de red N2 y N3 a menudo dependen del rendimiento del equipo de conmutación de servidor. La latencia de la aplicación (AI, A2 y A3) depende de la lógica de negocio desarrollada.

- Throughput

El Throughput se refiere al número de solicitudes de usuario que el sistema maneja dentro de una unidad de tiempo en particular. La unidad usada comúnmente es el número de solicitudes/segundo o el número de páginas/seg.

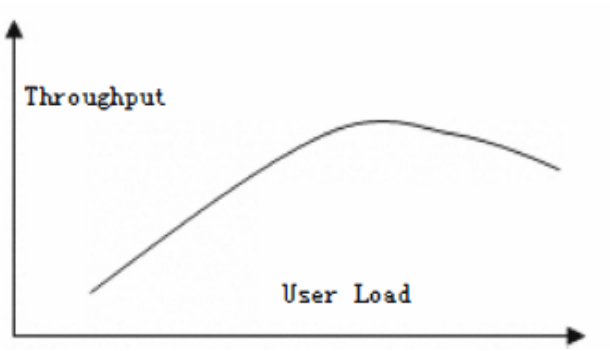


Figura 13. Throughput con la carga del usuario entre las curvas características típicas.

Tomado: De Zhu, Fu y Li (2010)

En la etapa inicial, el Throughput del sistema y la carga del usuario aumentan proporcionalmente. Sin embargo, debido a las limitaciones de recursos del sistema, el Throughput no puede aumentarse infinitamente. Gradualmente alcanzará un pico, y luego el Throughput total del sistema disminuirá a medida que aumente la carga.

- Utilización de recursos

La utilización se refiere al uso sistemático de diferentes recursos, como la CPU del servidor, la memoria, el ancho de banda de la red, etc. A menudo se usa para representar la mayor cantidad de recursos, medida como un porcentaje.

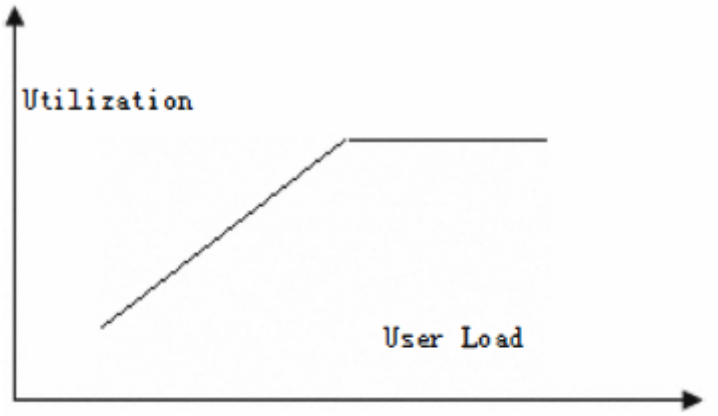


Figura 14. Utilización de los recursos en relación con las características de la carga del usuario.

Tomado: De Zhu, Fu y Li (2010)

3.3.1.1.2. Freitas & Renata - An Ontology for Guiding Performance Testing

Este artículo (Freitas & Renata, 2014) presenta la ontología y la compara con otras relacionadas, explorando las tecnologías semánticas para demostrar la factibilidad práctica de desarrollar aplicaciones basadas en ontología para ayudar a los evaluadores con la planificación y la administración de pruebas de rendimiento.

Esta investigación define que la métrica del rendimiento también es un concepto importante de la ontología, que se divide en dos categorías principales: las métricas relacionadas con las máquinas y las métricas relacionadas con las transacciones. Las métricas de máquina, representadas por el concepto MachinePerformanceMetric, se subdividen en categorías más especializadas, como métricas de memoria, procesador, proceso y disco. Por otro lado, las métricas relacionadas con una transacción, que son las necesarias para esta investigación están representadas por la clase PTMetric, que tiene instancias para representar el tiempo de respuesta y el Throughput.

3.3.1.1.3. Kalita, Khanikar, & Bezboruah - Investigation on performance testing and evaluation of PReWebN: a Java technique for implementing web application

En la investigación (Kalita, Khanikar, & Bezboruah, 2011), afirma que las pruebas de software son un proceso para evaluar la eficiencia, el rendimiento, la escalabilidad, la portabilidad, el cumplimiento, la interoperabilidad y la eficacia de un sistema. En el desarrollo de software, las pruebas se utilizan en puntos clave de control en un proceso general para determinar si se cumplen o no los objetivos. A nivel del sistema, el desarrollador o revisor independiente puede someter un servicio a una o más pruebas de rendimiento. La QoS de una aplicación web se mide en términos de tiempo de respuesta, Throughput y disponibilidad. Una de las mejores formas de medir un la QoS de la aplicación es realizar pruebas de carga.

3.3.1.2. Discusión

El análisis se limita principalmente a identificar en las fuentes principales, las métricas que se debe tomar en cuenta para que sea medible el rendimiento de los sitios web y también las formas de análisis adoptadas por los autores. Además, también consideramos la practicidad de utilizar casos de estudio y evaluar las métricas necesarias para comparar el rendimiento de aplicaciones web desarrolladas en diferentes frameworks.

Con base en la revisión literaria, cada autor definió métricas para medir el rendimiento de una aplicación web, pero dos métricas concordaron los autores, como se muestra en la tabla 4.

Tabla 5. Cuadro comparativo de métricas del mapping study

	Métricas			
Autores	**Tiempo de Respuesta**	**Throughput**	**Disponibilidad**	**Utilización de Recursos**
Zhu, Fu, & Li	X	X		X
Freitas & Renata	X	X		
Kalita, Khanikar, & Bezboruah	X	X	X	X

Nota: Elaboración propia

Para la pregunta de investigación planteada, las métricas de evaluación del rendimiento de aplicaciones web desarrolladas con API REST y SPA que se utilizaron para esta investigación son:

- Tiempo de Respuesta *(Response Time):* medido en milisegundos (ms), es el tiempo transcurrido entre la solicitud de un servicio hasta el final del servicio (producción del resultado). Este índice está directamente relacionado con el rendimiento percibido por los usuarios finales, e incluye los gastos generales introducidos por la API adoptada.

- *Throughput:* medido en solicitudes / s, es el número de solicitudes completadas por segundo. De hecho, el throughput es una medida de la capacidad de un sistema para completar las tareas asignadas dada la cantidad de recursos disponibles.

3.3.2. Implementación

La aplicación web donde se realizó las pruebas de rendimiento se desarrolló para cumplir los requerimientos del módulo de Mapeo de Actores, con la siguiente arquitectura:

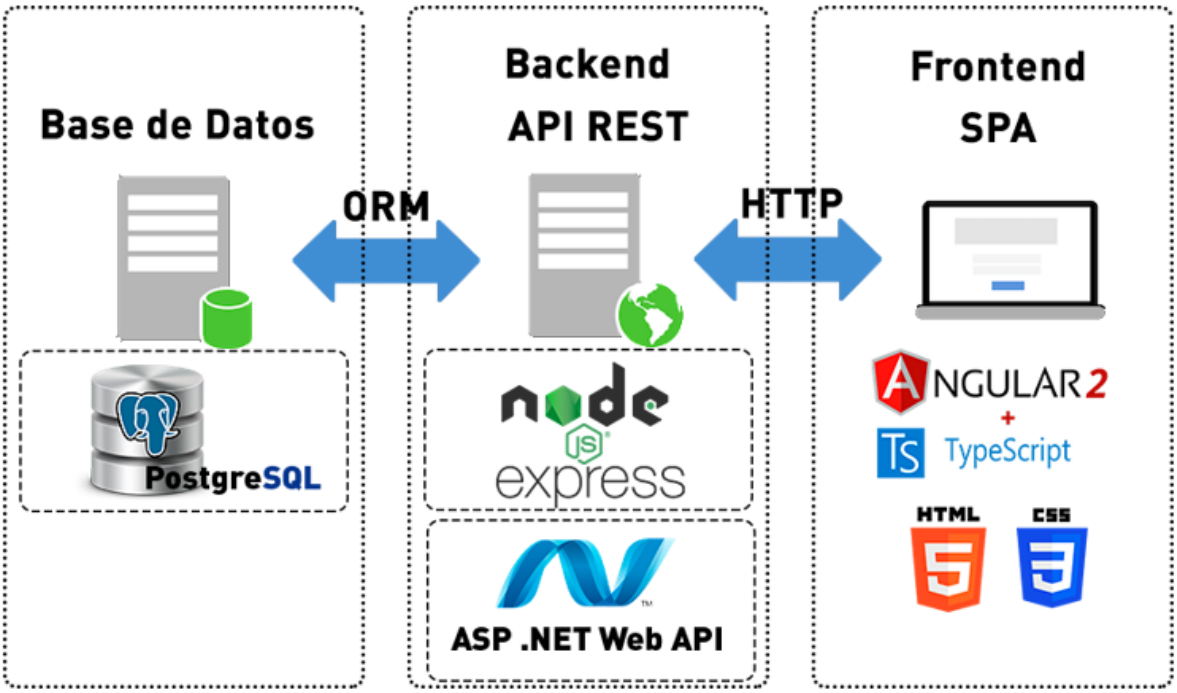


Figura 15.- Arquitectura de Aplicación Web mediante servicios REST y SPA.

Tomado: Elaboración propia.

El proyecto implica el desarrollo de componentes y servicios de software que apoyen el mantenimiento de la información de actores sociales para la Prefectura de Imbabura. El software constituye una "solución vertical" que se desarrolló sobre la base de la arquitectura de datos de la Suite de Aplicaciones de Gestión Institucional. Los requerimientos son historias de usuario, clasificadas en épicas y se encuentran en el ANEXO 1.

3.3.2.1. *Desarrollo de API REST*

Los servicios desarrollados en el backend se encuentran en el ANEXO 2. Las API REST que se desarrollaron para cumplir con los requerimientos del sistema, se basa en el envío y recepción de datos contenidos en la base de datos, es decir existe una gran cantidad de datos de entradas y salidas, y el backend debe soportar este entorno.

3.3.2.1.1. *Express js*

Node.js

Node.js es el primer programa que se descarga ya que proporciona la plataforma en la que se crea una aplicación MEAN. Se puede descargar desde el sitio web: https://nodejs.org/en/download. Después de la descarga e instalación, puede verificarse emitiendo el comando node -v desde el terminal o CLI como se ilustra en la figura 16.

Figura 16.- Versión de Node.js.

Tomado: Elaboración propia.

La Figura 16 ilustra el comando para ver la versión de Node.js instalada en la computadora. La versión utilizada en la aplicación es 8.2.1. Confirma que Node.js está instalado correctamente en la máquina de desarrollo.

Express es un WBAF en Javascript para Node.js que ayuda a organizar y estructurar cómo el usuario interactúa con la aplicación web. Node.js y Express.js utilizan una tecnología que sigue el modelo event-driven y non-blocking I/O, que son sistemas basados en eventos que no bloquea entrada/ salida (Valdez, 2016).

Después de la instalación de Node.js, se crea un directorio raíz y dentro del directorio, express se instala emitiendo el comando desde la terminal npm install -g express. También se puede

instalar un generador expreso para proporcionar una estructura simple a la aplicación. Todos los módulos y dependencias de nodo se instalaron usando el comando npm install.

Después de la instalación y configuración de todas las herramientas y programas, la estructura de los archivos de la aplicación es como en la figura 17.

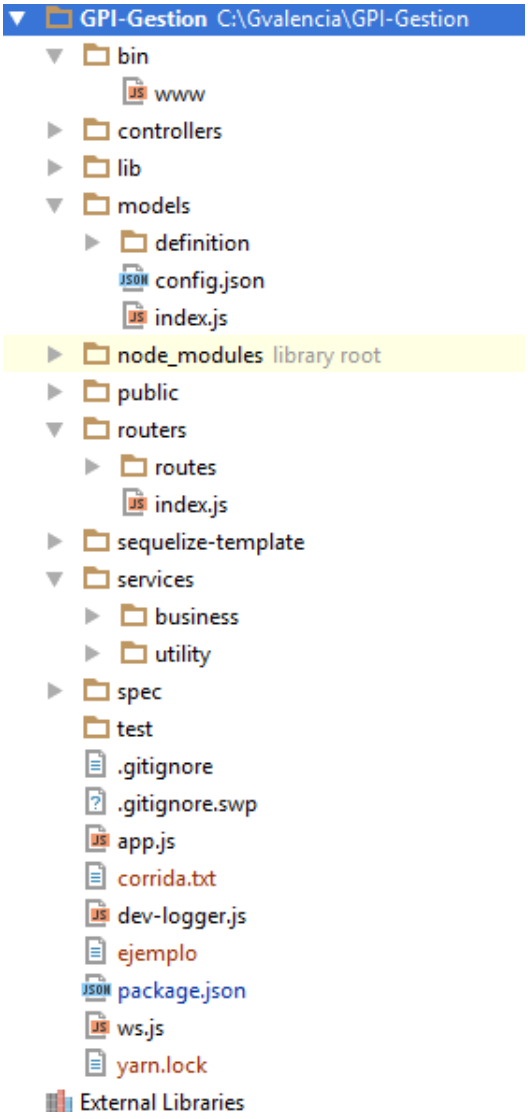


Figura 17.- Estructura de archivos de la aplicación con Express.js

Tomado: Elaboración propia.

El archivo 'package.json' contiene todas las dependencias requeridas para ejecutar la aplicación. El archivo 'app.js' contiene todo el middleware Express y las rutas API de la aplicación. El archivo 'www' contiene el comando para iniciar el servidor en el puerto 3000.

Los modelos, vistas y controladores están separados según la arquitectura MVC. Todas las dependencias del lado del servidor están instaladas en la carpeta 'node_modules'.

paquete.json

El archivo 'package.json' contiene toda la información importante sobre la aplicación y las dependencias necesarias para compilar y ejecutar la aplicación.

```
{
  "name": "server",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "test-mac": "NODE_ENV=test nodemon ./bin/www",
    "test": "set NODE_ENV=test&& nodemon ./bin/www",
    "start": "set NODE_ENV=development&& nodemon ./bin/www"
  },
  "dependencies": {
    "body-parser": "^1.16.1",
    "connect-busboy": "0.0.2",
    "cookie-parser": "^1.4.3",
    "cors": "^2.8.1",
    "express": "^4.14.1",
    "fs-extra": "^4.0.1",
    "fstream": "^1.0.10",
    "lodash": "^4.17.4",
    "morgan": "^1.8.1",
    "multer": "^1.3.0",
    "path": "^0.12.7",
    "pg": "6.4.0",
    "q": "^1.4.1",
    "request": "^2.81.0",
    "sequelize": "^3.30.2",
    "sequelize-cli": "^2.5.1",
    "sequelize-hierarchy": "^1.2.0",
    "sequelize-virtual-fields": "^1.1.0",
    "serve-favicon": "^2.4.3",
    "socket.io": "^2.0.3"
  },
  "devDependencies": {
    "@angular/cli": "^1.2.7",
    "debug": "^2.6.1",
    "nodemon": "^1.11.0"
  }
}
```

app.js

El archivo app.js es el archivo de configuración principal de la aplicación Express. Configura el servidor e importa todos los módulos para ejecutar la aplicación.

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var busboy = require('connect-busboy');
var fs = require('fs-extra');
var cors = require('cors');
var app = express();
app.use(cors());
// uncomment after placing your favicon in /public
app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(busboy());

if (app.get('env') === 'test') {
  console.log("Trabajando en PRUEBA........");
}

if (app.get('env') === 'development') {
  console.log("Trabajando en DESARROLLO........");
}
// production error handler
// no stacktraces leaked to user
if (app.get('env') === 'production') {
  console.log("AMBIENTE DE PRODUCCION........");
  app.use(express.static(path.join(__dirname, '/app')));
  app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.render('error', {
      message: err.message,
      error: {}
    });
  });
}

var router = require('./routers')(app);
app.use(function(err, req, res, next) {
  res.status(err.status || 500);
});

module.exports = app;
```

www.js

Configuración del puerto y creación del Server HTTP, y el enrutamiento de los archivos.

```
var app = require('../app');
var debug = require('debug')('server:server');
var http = require('http');
var models = require("../models");

/**
```

```
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);
var ws = require('../ws.js')(server);
```

Models

Para operar con una base de datos, se utilizará un paquete Sequelize. Sequelize es una biblioteca ORM (mapeo relacional de objetos), que proporciona una integración de Node.js con PostgreSQL permitiendo la conversión de datos entre una base de datos relacional y una base de datos ejecutada con lenguaje de programación orientado a objetos.

Por almacenar los datos en una base de datos es esencial crear esquemas y modelos. Los esquemas definen las estructuras de datos en documentos individuales y los modelos representan estos documentos, que luego podrían usarse para el almacenamiento de datos y la recuperación de datos.

```
'use strict';

var fs        = require('fs');
var path      = require('path');
var Sequelize = require('sequelize');
require('sequelize-virtual-fields')(Sequelize);
var basename  = path.basename(module.filename);
var env       = process.env.NODE_ENV || 'development';
var config    = require(__dirname + '/config.json')[env];
var db        = {};
var definitions = path.join(__dirname,'/definition');

if (config.use_env_variable) {
  var sequelize = new Sequelize(process.env[config.use_env_variable]);
} else {
  var sequelize = new Sequelize(config.database, config.username,
config.password, config/*{
    dialect: "postgres",
    define:{
      timeStamp: false,
      freezeTableName: true,
      timestamps: false
    }
  }*/);
}
```

```
db.sequelize = sequelize;
db.Sequelize = Sequelize;
db.sequelize.initVirtualFields();
module.exports = db;
```

Para crear la API REST se utiliza una biblioteca Express.js llamada “sequelize node-restful --- librería en el archivo de creación del modelo”. Si se pasa una tabla en PostgrSQL a esta librería, genera automáticamente rutas RESTful.

```
'use strict';
module.exports = function(sequelize, DataTypes) {
   var Geografica =  sequelize.define('Geografica', {
        geoId: {
            type: DataTypes.INTEGER,
            field: 'geoId',
            allowNull: false,
            primaryKey: true
        },
        geoCodigo:{
            type: DataTypes.STRING(255),
            field: 'geoCodigo',
            allowNull: false
        },
        geoNombre:{
            type: DataTypes.STRING(255),
            field: 'geoNombre',
            allowNull: false
        },
        geoAbreviatura:{
            type: DataTypes.STRING(255),
            field: 'geoAbreviatura',
            allowNull: false
        }
    }, {
        schema: 'public',
        tableName: 'geografica',
        timestamps: false,
        classMethods: {

            associate: function(models) {
                this.belongsTo(models.GeograficaTipo,{
                    as: 'GeoGeoTip',
                    foreignKey: 'geotipoId',
                    onDelete: 'CASCADE',
                    onUpdate: 'NO ACTION'
                });
            }
        }
     });
    return Geografica;
};
```

Services

Donde se desarrolla la lógica del negocio de los servicios REST.

```
var Q = require('q');
var models = require('../../models');
var service = {};
service.listGeografica = listGeografica;

function listGeografica(pBusqueda) {
    var deferred = Q.defer();
        models.Geografica.findAll({
            where: pBusqueda.Geografica,
            include: [{
                model: models.GeograficaTipo,
                as: 'GeoGeoTip',
                where: pBusqueda.GeograficaTipo
            }]
        }).then(function (listadoRet) {
            deferred.resolve(listadoRet);
        }).error(function (error) {
            deferred.reject(error);
        });
    });
    return deferred.promise;
}
```

Controllers

Middleware de las API REST.

```
var geograficaService = require('../services/business/geografica.service');
var controlador = {};
controlador.listGeografica = listGeografica;
module.exports = controlador;
function listGeografica(req, res) {
    geograficaService.listGeografica(req.body).then(function
(retGeograficas) {
        res.status(200).jsonp(retGeograficas);
    }).error(function (errores) {
        res.status(404).jsonp(errores);
    });
}
```

Routers

El diseño de los servicios web REST implementado se encuentra en el directorio / api /. En la carpeta /routes/ se encuentran las rutas que el servicio web está utilizando para responder y controlar el flujo de información. Aquí se establece los URI que realizan acciones a través de

los servicios web. Se incluye los controladores que se transmiten junto con los verbos HTTP en el servicio web. Estos controladores contienen la información para cada solicitud (request).

```
var api = '/api';
module.exports = function (app) {
        app.use(api + '/geograficas', require('./routes/geograficas'));
}
```

3.3.2.1.2. ASP .NET CORE

La mayoría de las aplicaciones para ASP.NET usan IIS como un servidor de aplicaciones. La versión actual de IIS es IIS8 que se lanzó junto con Windows 8 y Windows Server 2012 (McMurray, s.f.). Con esto vino el soporte para el protocolo WebSockets (Microsoft, s.f.). IIS es gratuito, ya que forma parte del sistema operativo Windows. Es una característica de Windows que debe activarse. Para el desarrollo, no siempre es necesario implementar una aplicación en IIS. Junto con el marco de ASP.NET y Visual Studio, los desarrolladores obtienen acceso a IIS Express. Como con todos los productos de la serie Express de Microsoft, es una versión ligera del original. Proporciona lo que normalmente necesita durante el desarrollo, pero su rendimiento no es suficiente para la producción.

El siguiente diagrama muestra el diseño básico de la aplicación (Anderson & Wasson, 2017).

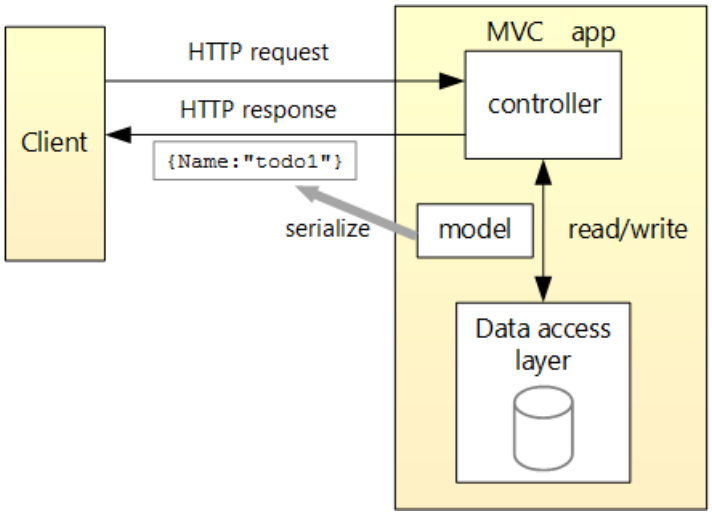


Figura 18.- Diseño de la aplicación.

Tomado: De Anderson y Wasson *(2017)*

La estructura de los archivos de la aplicación.

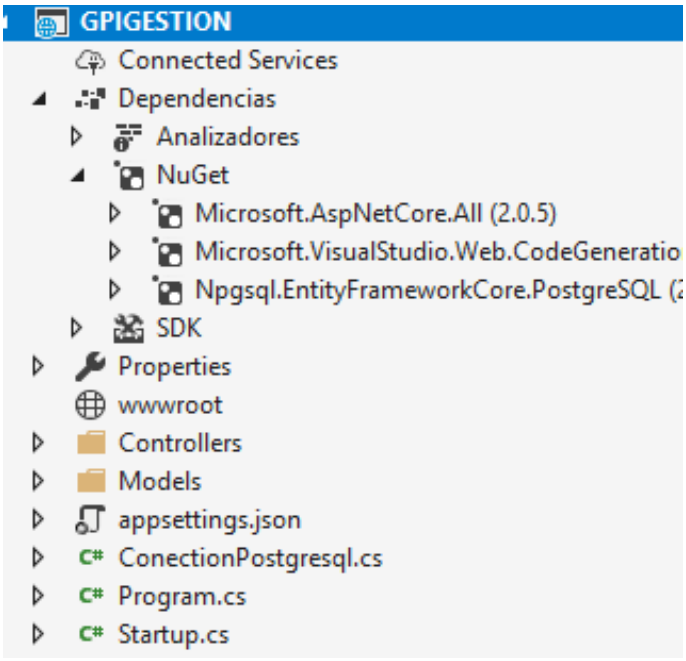


Figura 19.- Estructura del API REST en ASP.Net Core

Tomado: Elaboración propia

Models

Se crea una clase con la estructura del esquema de la base de datos, Asp.net core con Entity Framework Core es un mapeador relacional de objetos (ORM) que permite a los desarrolladores de .NET trabajar con una base de datos usando objetos .NET) (Miller R. , 2016). El contexto de la base de datos es la clase principal que coordina la funcionalidad de Entity Framework para un modelo de datos dado. Esta clase se crea derivando de la clase Microsoft.EntityFrameworkCore.DbContext.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace GPIGESTION.Models
{
    public class geografica
    {
        public int geoId { get; set; }
        public string geoCodigo{ get; set; }
        public string geoNombre { get; set; }
```

```
        public string geoAbreviatura { get; set; }
}
}
```

Controllers

El controlador contiene la lógica de control de flujo. Un controlador determina qué respuesta enviar al usuario cuando el usuario realiza una solicitud del navegador. (Walther, 2008)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using GPIGESTION.Models;
using Npgsql;
namespace GPIGESTION.Controllers
{
    [Route("api/[controller]")]
    public class GeograficaController : Controller
    {
        // GET api/<controller>/5
        [HttpGet("{id}")]
        public IActionResult Get(int id)
        {
            var resultado = new List< geografica>();
            var conexion = new ConectionPostgresql();
            var cadenaSQL = string.Empty;
            using (var db = conexion.AbreConexion())
            {
                cadenaSQL = "SELECT * FROM geografica ";
                try
                {
                    var comando = new NpgsqlCommand(cadenaSQL, db);
                    var lector = comando.ExecuteReader();
                    while (lector.Read())
                    {
                        var objetoGeografica = new geografica();
                        objetoGeografica.geoId = (int)lector["geoId"];
                        objetoGeografica.geoCodigo =
(string)lector["geoCodigo"];
                        resultado.Add(objetoGeografica);
                    }
                }
                catch (Exception)
                {
                    throw;
                }
            }
           return new ObjectResult(resultado);
        }
    }
}
```

3.3.2.2. Desarrollo del Cliente con SPA

El framework Angular 2 necesita para compilar Node.js y el manejador de paquetes de Node NPM (Node Package Manager).

Estructura de archivos del proyecto Cliente:

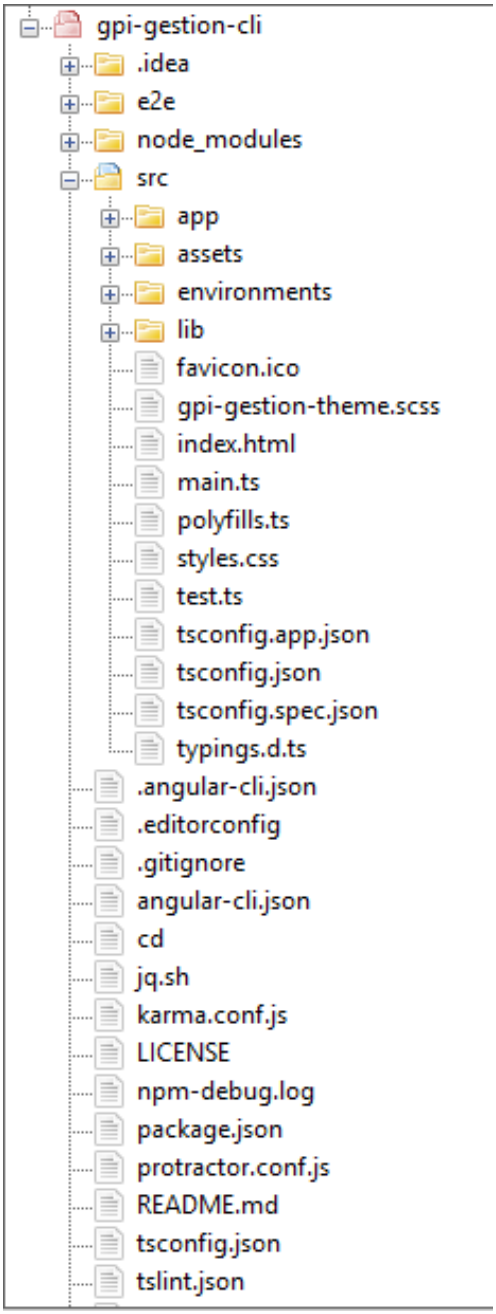


Figura 20.- Estructura del proyecto Cliente con SPA.

Tomado: Elaboración propia.

Los archivos más importantes son:

- main.ts. - El punto de entrada principal de la aplicación que inicia el módulo raíz de la aplicación para que se ejecute en el navegador.
- tsconfig.ts. - Archivo de configuración para el compilador de TypeScript.
- index.html. – Archivo que se carga al principio de la aplicación y el CLI agrega los demás archivos que forman los componentes de la aplicación.
- package.json.- Lista los paquetes del proyecto.
- .angular-cli.json. - Archivo de configuración para CLI angular. En este archivo, se pueden establecer varios valores predeterminados y también configurar qué archivos se incluyen cuando se crea el proyecto.

Todos los componentes, servicios, módulos y todo lo que el desarrollador crea se encuentra en la carpeta / src / app. Es importante elegir la estrategia de estructura adecuada para esta carpeta para que esté organizada y sea fácil de refactorizar.

AppModule contiene el componente superior en una jerarquía de componentes, también proporciona el enrutamiento de alto nivel e importa los módulos que son necesarios para toda la aplicación.

```
@NgModule(
  {
    declarations: [
      AppComponent
    ],
    imports: [
      BrowserModule,
      FormsModule,
      BrowserAnimationsModule,
      HttpModule,
      CoreModule.forRoot({userName: 'Miss Marple'}),
      AppMaterialModule,
      PrimeNGModule.forRoot(),
      AppRouting,
      GPICIModule,
      GPIMAModule,
      GPITURModule,
      EsriMapModule,
      GpiCirmModule
```

```
    ],
    providers: [PgqlService],
    bootstrap: [AppComponent]
  })
export class AppModule {
}
```

CoreModule contiene todos los servicios que deben ser un singleton. CoreModule solo se importa en AppModule y nunca se importa en ningún otro módulo. CoreModule proporciona todos los servicios responsables de realizar las solicitudes de la API REST.

```
@NgModule({
            imports: [
              CommonModule,
              RouterModule,
              AppSharedModule.forRoot()
            ],
            declarations: [
              SuiteCmp,
              NoContent,
              About
            ],
            exports: [
              SimpleNotificationsModule,
              AppSharedModule,
              SuiteCmp,
              NoContent,
              About
            ],
            providers: [ConfigService, UserService, ParametrosService,
AppSearchService]
          }
)
export class CoreModule {

  constructor (@Optional() @SkipSelf() parentModule: CoreModule) {
    if (parentModule) {
      throw new Error(
        'CoreModule is already loaded. Import it in the AppModule only'
      );
    }
  }

}
```

Shared se encuentran los archivos que se conectan con las API REST:

```
@Injectable()
export class GeograficasService {
  _baseUrl: string = '';
```

```
  constructor (private http: Http,
               private configService: ConfigService,
               private appNotificationsService: AppNotificationsService
  ) {
    this._baseUrl = configService.getApiURI();
  }


  listGeografica (pFiltro): Observable<any> {
    return this.http
               .post(`${this._baseUrl}/geograficas/geograficaslist`,
pFiltro, {
                       headers: this.configService.getHeaders()
                     }
               )
               .map(res => res.json())
               .catch(this.handleError);
  }
```

Componentes es el mediador entre el template y el servicio:

```
@Component(
  {
    selector: 'geograficas',
    templateUrl: './geograficas.html',
    encapsulation: ViewEncapsulation.None
  }
)


export class GeograficasCmp implements OnInit {
  geograficas: TreeNode[];

  selectedValues: string[] = [];
  buscadoGeografica: any;

  appSearchCanal = 'search';
  AppSearchEmmiterSearchByGeografica        =
AppSearchService.get(this.appSearchCanal);

  filtro = {
    name: "Campo Acción",
    value: ["CampoAccion.geoId", "$eq"]
  };

  filtros = [];

  constructor (private appSearchService: AppSearchService,
               private appNotificationsService: AppNotificationsService,
               private geograficasService: GeograficasService
  ) {

  }

  ngOnInit () {
    this.listarGeograficas();
  }
```

```
  listarGeograficas () {


this.geograficasService.listGeograficaEstructura(this.appSearchService.filt
ros)
        .subscribe(response => {
                    this.geograficas = response;
                  }
        );

  }
```

Template es el archivo con código html y referencia los componentes:

```
<p-treeTable [value]="geograficas" selectionMode="multiple"
[(selection)]="selectedGeografica" class="arbol">
  <p-column field="geoNombre" header="Geografica">
    <template pTemplate type="header">
      Geografica
      <div style="align-content: flex-end">
        <button pButton type="text" icon="fa-expand" (click)="expandAll()"
></button>
        <button pButton type="text" icon="fa-compress"
(click)="collapseAll()" ></button>
          
        <input type="text" pInputText
               [(ngModel)]="buscadoGeografica"
               placeholder="Busqueda"
               size="17"
        />
        <button pButton type="text" (click)="onSearchClick()" icon="fa-
search"></button>
        <button pButton type="text" (click)="onClearClick()" icon="fa-
trash"></button>
      </div>
    </template>
    <template let-geografica="rowData" pTemplate type="body">
      <p-checkbox [value]="geografica.data.geoId"
                  [label]="geografica.data.geonombre"
                  [(ngModel)]="selectedValues"
                  (onChange)="onChangeNodeSelection()"
                  [ngStyle]="{'white-space':'normal'}"></p-checkbox>
    </template>
  </p-column>
</p-treeTable>
```

3.3.3. Metodología Benchmark

(Patil & Joshi, 2012) describe la metodología del benchmark de frameworks para aplicaciones web, y (Echeverría Perez & Abella Paumier) en su investigación realizan un procedimiento que coincide con Patil. En esta investigación, relacionando la metodología de Patil con la medición del performance de las API REST desarrolladas en diferentes frameworks y con el

consumo de las API REST en aplicaciones web con SPA definimos las siguientes actividades del benchmark:

a. Identificar los entornos de pruebas
b. Diseñar las pruebas
c. Ejecutar las pruebas
d. Analizar los resultados

3.3.3.2. Identificar los entornos de pruebas

Para este proyecto se definen dos entornos de pruebas:

a. Primera prueba. - el tiempo de response / request de la API REST desarrollada en cada framework, La prueba realizada consiste simular 50, 100 y 500 accesos de usuarios respectivamente, se definió consumir una API con HTTP GET.
b. Segunda prueba. - el tiempo de carga de archivos .html , .js y de objetos JSON desde el servidor a front end de la aplicación web. La prueba realizada consiste en definir 1 test con el acceso a un usuario.

Las pruebas se ejecutan en un equipo con las siguientes características:

Tabla 6. Características de equipo de pruebas.

Hardware	**Software**
Computador portátil Toshiba Satellite S55t – B5273NR, procesador Intel ® Core ™ i7-4710HQ CPU @ 2.50GHz, RAM 8 GB, SO 64 bits.	Sistema Operativo Windows 8.1 Pro de 64 Bits. Base de Datos PostgreSQL 9.3 con registros en las tablas de 5844.

Nota: Elaboración propia

Las especificaciones del equipo donde se realizaron las pruebas son las siguientes:

Tabla 7. Especificaciones del escenario de pruebas

Especificaciones	**Valor -> Equivalencia**
Uso de CPU	1.18 GHz -> 1.1 %
Uso de Memoria	534,1 MB -> 27%
Uso de Disco	0.1 MB / s -> 2%
Uso de Red	0 Mbps -> 0%
Número de servicios en ejecución	96
Número de procesos	96

Nota: Elaboración propia

3.3.3.3. *Diseñar las pruebas*

Para el proceso de las pruebas de las API REST desarrolladas en los frameworks Express.js y ASP.NET Core se dispuso de la ejecución desde una herramienta de test de API REST y de la aplicación web con SPA con herramienta de desarrolladores implementados en el navegador.

El consumo de la API REST se realiza desde un servicio HTTP GET con las siguientes métricas:

- Media: tiempo promedio en milisegundos para un conjunto de resultados.
- Min: tiempo mínimo que demora un thread en acceder a una página.
- Max: tiempo máximo que demora un thread en acceder a una página
- Rendimiento: rendimiento medido en los requerimientos por segundo / minuto / hora.
- Kb/sec: rendimiento medido en Kbytes por segundo.
- Media en bytes: tamaño medio de respuesta del servidor (en bytes)

En la aplicación web la carga de archivos utilizando los servicios evaluados anteriormente.

- Requests. – elementos retornados.
- Transferred. – tamaño de los elementos retornados.
- Load. - tiempo en cargar los documentos

3.3.3.4. Ejecutar las pruebas

Para la ejecución de las pruebas se utilizó las siguientes herramientas:

Herramienta de test de API REST:

- Apache JMeter. - se puede usar para probar el rendimiento tanto en recursos estáticos como dinámicos (archivos, servlets, scripts Perl, objetos Java, bases de datos y consultas, servidores FTP y más). Se puede usar para simular una carga pesada en un servidor, red u objeto para probar su potencia o para analizar el rendimiento general bajo diferentes tipos de carga.

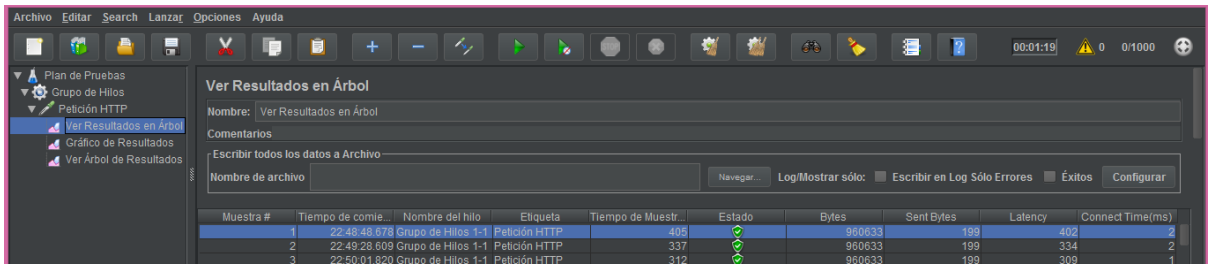


Figura 21.- Herramienta JMETER.

Tomado: Elaboración propia.

Herramienta de test de aplicaciones web:

- Herramienta para desarrolladores del navegador web Google Chrome. - es una herramienta que monitorea diferentes métricas de la carga de los archivos y consumo de servicios de las aplicaciones web.

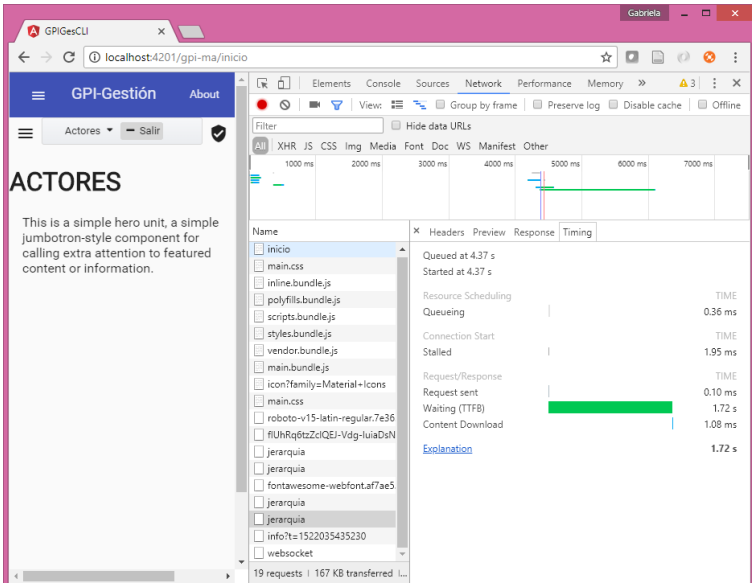


Figura 22. Herramienta para desarrolladores Navegador Google Chrome.

Tomado: Elaboración propia

CAPÍTULO 4. RESULTADOS Y DISCUSIÓN

Para comparar el rendimiento de aplicaciones web dinámicas con SPA con el framework Express.js y con el framework ASP.NET Core, se ejecutó las pruebas planteadas en el capítulo anterior.

4.1. Primera Prueba

En la primera prueba, recupera un objeto JSON de un HTTP GET en cada petición de las API REST de los frameworks Express.js y ASP.NET Core y con el mismo número de usuarios para demostrar las diferencias en el rendimiento.

Análisis:

Se analizaron los resultados a través de un intervalo de confianza[1] con un nivel de confianza al 95%. Para un primer análisis, se supone que la población tiene una distribución Normal. Para un segundo análisis, dado que la muestra es grande, no se requiere hacer la suposición de que la muestra tiene una distribución Normal ya que por el Teorema Central del Límite (TCL), para n grande implica que X tiene una distribución aproximadamente normal sin importar la naturaleza de la distribución poblacional (Devore).

Los valores totales obtenidos se muestran en la siguiente tabla:

[1] Se llama intervalo de confianza a un intervalo de confianza es un rango de valores, derivado de los estadísticos de la muestra, que posiblemente incluya el valor de un parámetro de población desconocido. (Minitab 18, 2017)

Tabla 8.- Métricas de JMeter con 50 usuarios concurrentes

Framework	#Muestras	Media	95% Line	Min	Max	%Error	Peticiones/ sec	Kb/sec	Sent KB/sec	Tiempo de Respuesta
ASP.NET Core	50	895	1674	57	2148	0,00%	20,6/sec	12072,82	2,72	0:00:02
Express.js	50	745	1244	45	1273	0,00%	22,1/sec	12794,46	2,89	0:00:02

Nota: Elaboración propia

Tabla 9.- Métricas de JMeter con 100 usuarios concurrentes

Framework	#Muestras	Media	95% Line	Min	Max	%Error	Peticiones/ sec	Kb/sec	Sent KB/sec	Tiempo de Respuesta
ASP.NET Core	100	2905	5011	821	5281	0,00%	20,3/sec	10594,56	2,38	0:00:05
Express.js	100	2415	4230	182	4356	0,00%	21,7/sec	10809,73	2,44	0:00:05

Nota: Elaboración propia

Tabla 10.- Métricas de JMeter con 500 usuarios concurrentes

Framework	#Muestras	Media	95% Line	Min	Max	%Error	Peticiones/ sec	Kb/sec	Sent KB/sec	Tiempo de Respuesta
ASP.NET Core	500	13492	24503	1529	24968	0,00%	20,0/sec	11461,93	2,58	0:00:25
Express.js	500	11292	20329	341	21263	0,00%	21,2 / sec	12968,09	2,93	0:00:22

Nota: Elaboración propia

Tabla 11.- Métricas de JMeter con 1000 usuarios concurrente

Framework	#Muestras	Media	95% Line	Min	Max	%Error	Peticiones/ sec	Kb/sec	Sent KB/sec	Tiempo de Respuesta
ASP.NET Core	1000	26253	48029	388	50277	0,00%	19,8/sec	11596,95	2,61	0:00:50
Express.js	1000	23719	43605	907	45743	0,00%	20,9/sec	12352,86	2,79	0:00:46

Nota: Elaboración propia

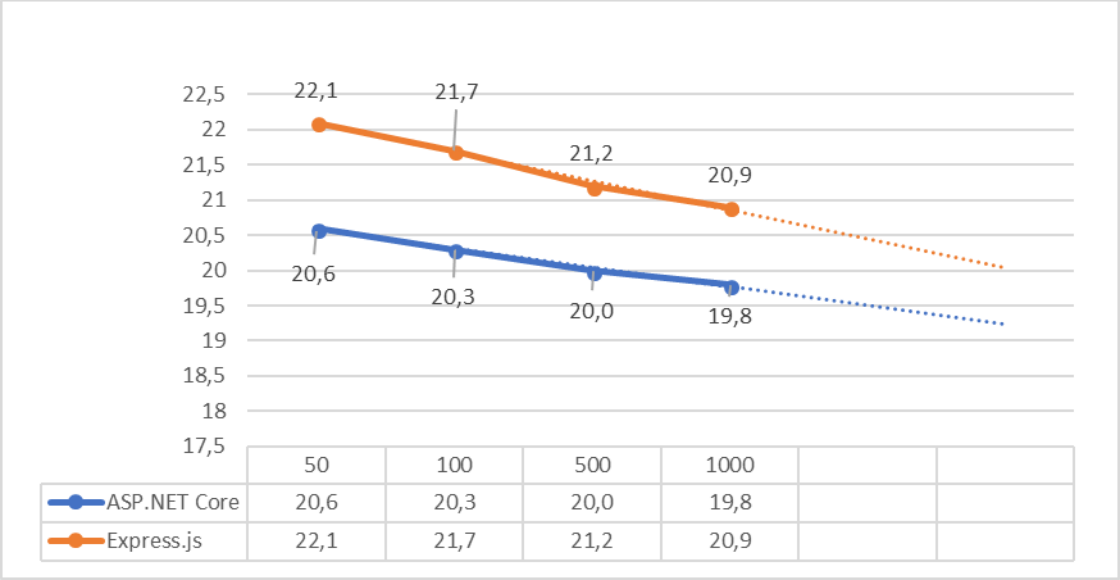


Figura 23.- Rendimiento en el consumo de la API REST.

Tomado: Elaboración propia.

En esta prueba se puede observar que en promedio se obtiene que el framework Express.js tiene un mejor rendimiento que ASP.NET Core, llegando a ser un 8% mejor en el rendimiento. En la Figura 22 muestra que Express.js con los diferentes usuarios concurrentes realizados ejecuta mayores peticiones sobre segundo, en cambio ASP.NET Core ejecuta una menor cantidad de peticiones sobre segundo, y se muestra una tendencia de diferencia de 1.6 peticiones por segundo.

Se puede determinar que solo en el consumo de la API REST es mejor en rendimiento si se desarrolla con el framework Express.js.

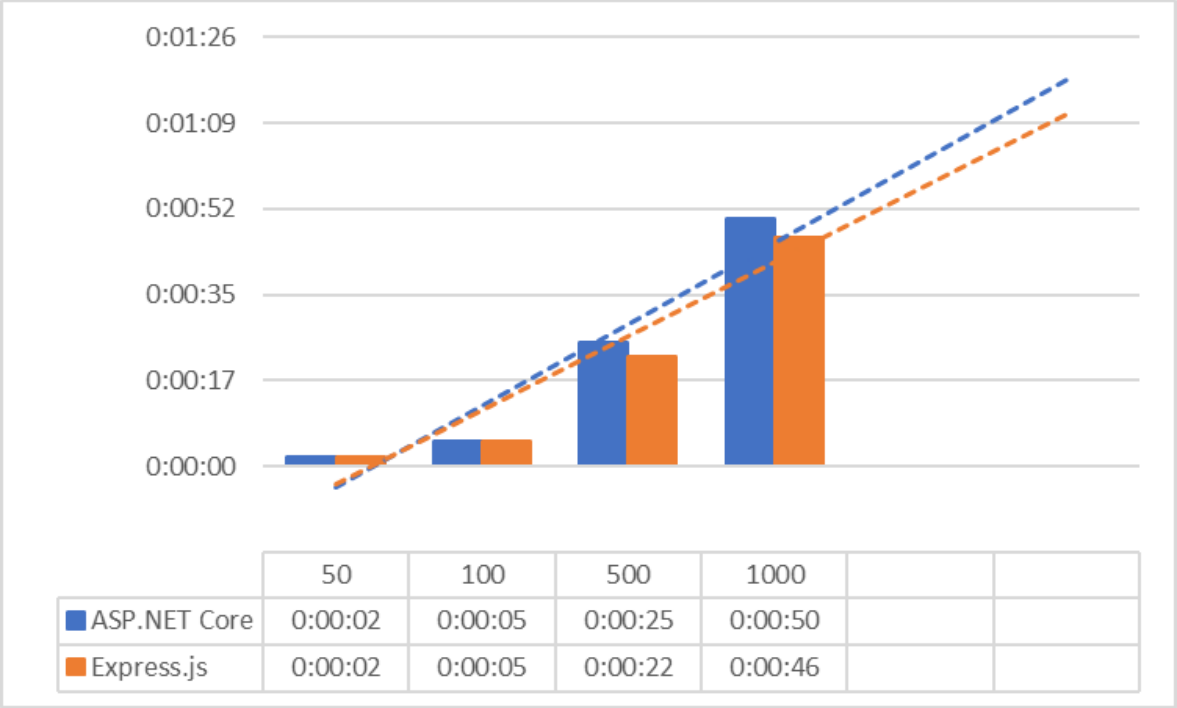


Figura 24.- Tiempo de respuesta en el consumo de la API REST.

Tomado: Elaboración propia.

En la Figura 23 muestra que la diferencia del tiempo de respuesta de Express.js y ASP.NET Core depende de los usuarios concurrentes, así mostrando una notable diferencia en la cantidad de usuarios superior a 500. La tendencia son líneas divergentes lo que implica que mientras sea una mayor cantidad usuarios concurrentes mayor será la diferencia en los tiempos de respuesta.

Se puede determinar que solo en el consumo de la API REST el tiempo de respuesta es menor se desarrolla con el framework Express.js.

4.2. Segunda Prueba

En la segunda prueba en la aplicación web se demuestra las diferencias en la carga de archivos:

Tabla 12.- Métricas de la herramienta para desarrolladores del Navegador Google Chrome

Framework	Archivos solicitados	Tamaño de transferencia	Tiempo de carga
Express.js	19	167 KB	5,80 s
ASP.NET Core	19	170 KB	5,85s

Nota: Elaboración propia

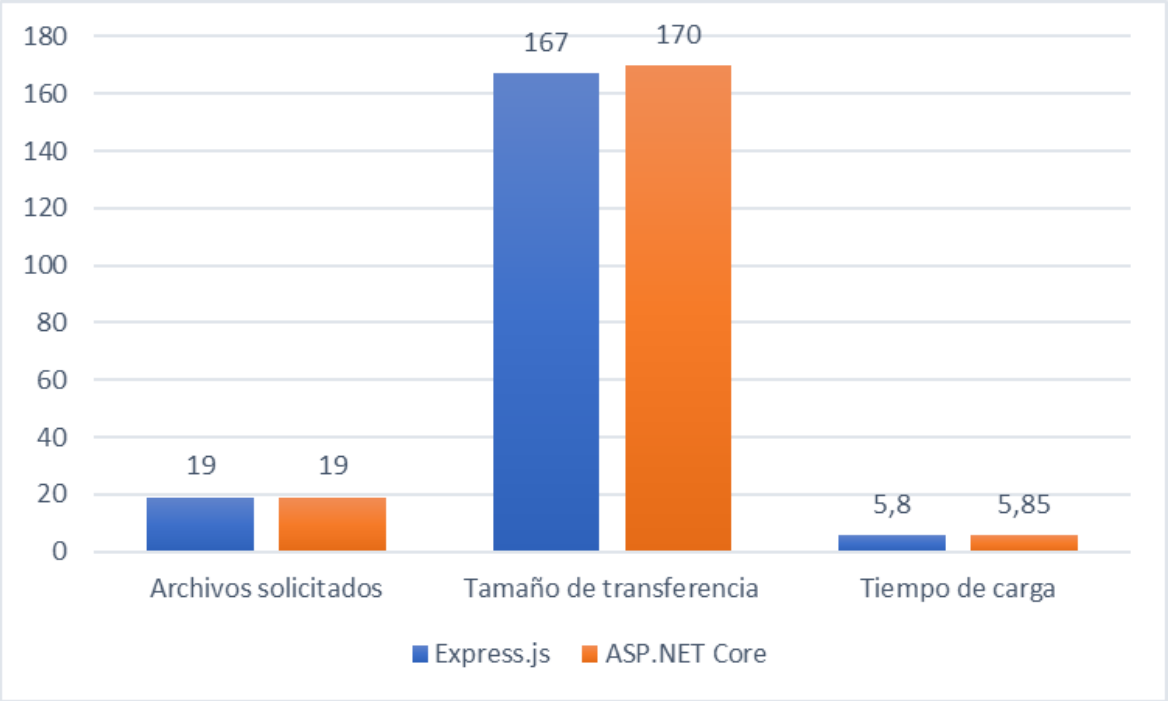


Figura 25.- Tiempo de carga de la aplicación web con SPA.

Tomado: Elaboración propia.

En esta prueba se puede observar que desde el Frontend el consumo de las API REST desarrolladas en los frameworks si tiene una afectación en el tiempo de carga y el tamaño de los archivos. Aunque comparando las diferencias con la prueba anterior no es tan notorio ya que la arquitectura de las aplicaciones web con SPA tiene un control en los archivos que transfiere desde el servidor para minimizar el tiempo y los componentes de carga. En la Figura 24 se puede diferenciar que existe una mejora del 2% en el tamaño de los archivos transferidos cuando se consume las API REST desarrolladas con el framework Express.js comparado con las API desarrolladas con el framework ASP.NET Core. En el tiempo de carga se puede diferenciar un 0.8% de diferencia que equivale a 0.05s, que ante la percepción del usuario no es notoria la diferencia.

Se puede determinar que 0.05s son la diferencia en el tiempo de respuesta de la aplicación web con SPA y consumo de API REST desarrolladas en los frameworks Express.js y ASP.NET

Core; y según (Miller R. B.) define que 1.0 segundo es aproximadamente el límite para que el flujo de pensamiento del usuario permanezca ininterrumpido, aunque el usuario notará la demora. Normalmente, no se necesita retroalimentación especial durante retrasos de más de 0.1 pero menos de 1.0 segundo, pero el usuario pierde la sensación de operar directamente sobre los datos.

CAPÍTULO 5. CONCLUSIONES Y RECOMENDACIONES

- En este documento, presentamos un estudio para analizar el impacto del WBAF (Web Backend Application Framework) de API REST en el rendimiento de aplicaciones web dinámicas con SPA (Single Page Application) para definir el WBAF que mejor se adapte al desarrollo de aplicaciones web del Gobierno Provincial de Imbabura se realizó un prototipo de una aplicación web, en el frontend con Angular2 y arquitectura SPA y el desarrollo de las mismas API REST en el backend con los frameworks planteados. Con base en nuestros hallazgos y análisis tratamos de responder las preguntas de investigación planteadas en la sección de introducción y al obtener las métricas a medir en el prototipo de la aplicación web y las API REST se utilizó herramientas de testing como JMeter y la herramienta para desarrolladores del Navegador Google Chrome. Al responder la formulación del problema, podemos decir que "SI" impacta el framework de desarrollo de API REST sobre el rendimiento de ejecución de aplicaciones web con SPA, solamente cuando la cantidad de usuarios concurrentes sean mayor a 500, caso contrario ante la percepción del usuario no impactaría el framework de desarrollo.
- En el desarrollo de las API REST en los frameworks Express.js y ASP. NET Core, lo más importante es conocer la estructura de las API REST porque estos frameworks y la mayoría de frameworks necesitan los mismos componentes como el modelo, el controlador y el router.
- El diseño de pruebas se enfoca en las métricas definidas por la pregunta de investigación *Throughput y Response Time,* y con las herramientas de test utilizadas se obtiene los valores para las métricas descritas.
- Se concluye que en el escenario de prueba propuesto tiene un mejor rendimiento del 8% las API REST desarrolladas con el framework Express.js que API REST desarrolladas con ASP. NET Core.

- Se recomienda realizar pruebas de concepto al momento de decidir el framework y la arquitectura de desarrollos de los proyectos de Software, aunque existen frameworks que ofrecen un mayor rendimiento y estabilidad de versiones, esto no garantiza que el resultado satisfaga los requerimientos de rendimiento de cada proyecto y también tener en cuenta la experiencia del equipo de desarrollo en los frameworks.
- Se recomienda para la realización de pruebas que los querys realizados a la base de datos sean iguales, ya que en algunos ORM tienen formas diferentes de generación de los querys.
- Se recomienda que la metodología de benchmark se realice en escenarios similares para tener una mayor precisión en los resultados de comparación de los frameworks.

CAPÍTULO 6. REFERENCIAS BIBLIOGRÁFICAS

Alonso Amo, F., Martínez Normand, L., & Segovia Pérez, F. J. (2005). *Introducción a la ingeniería del software.* Madrid: Delta Publicaciones.

Anderson, R., & Wasson, M. (08 de Agosto de 2017). *Create a web API with ASP.NET Core and Visual Studio for Windows.* Obtenido de docs.microsoft.com: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api

Angular. (2018). Architecture Overview. *Angular.io*. Obtenido de https://angular.io/guide/architecture

Aversano, G., Rak, M., & Villano, U. (2013). The mOSAIC Benchmarking Framework: Development and Execution of Custom Cloud Benchmarks. *Scalable Computing: Practice and Experience, 14*(1).

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice.* Addison Wesley.

Baya, V., Gruman, G., & Parker, B. (2012). Exploiting the growing value from information: Creating an operating model for permeability. *Technology Forecast, 2*, 07-23. Recuperado el 28 de Septiembre de 2016, de http://www.pwc.com.br/pt/publicacoes/setores-atividade/assets/technology-forecast/techforecast-2012-issue-2.pdf

Bixby, J. (2012). Latency 101: What is latency and why is it such a.

Bondi, A. B. (1994). *Characteristics of Scalability and Their Impact on Performance.* New Jersey: AT&T Labs.

Casarin, H., & Casarin, S. (2012). *Pesquisa científica: da teoria à prática.* Brasil. doi:ISBN: 9788582123690

Collins, G., & Sisk, D. (2015). API economy. *Tech Trends 2015: The fusion of business and IT*, 21-33. Recuperado el 28 de Septiembre de 2016, de https://www2.deloitte.com/content/dam/Deloitte/no/Documents/technology/Tech-Trends-2015.pdf

Constitución de la República del Ecuador. (20 de Octubre de 2008). Registro Oficial 449. Quito.

Cuomo, A., Rak, M., & Villano, U. (2015). Performance prediction of cloud applications through benchmarking and simulation. *International Journal of Computational Science and Engineering, 11*(1), 46-55.

De Luca, V., Epicoco, I., Lezzi, D., & Aloisio, G. (Junio de 2012). GRB_WAPI, a RESTful Framework for Grid Portals. *Proceedings of the International Conference on Computational Science 2012*. doi:10.1016/j.procs.2012.04.049

Del Pino, J. (19 de 01 de 2018). *Equipo de Desarrollo de Mozilla.* Obtenido de Developer.Mozilla: https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction$history

Devore, J. L. (s.f.). *Probabilidades y Estadísticas para Ingeniería y Ciencias* (Sexta ed.).

Dosé, M., & Lilja, H. (2015). *A schematic for comparing web backend application frameworks With regards to responsiveness and load scalability.* Gotemburgo: Chalmers University of Technology. Recuperado el 10 de Julio de 2017, de http://studentarbeten.chalmers.se/publication/219826-a-schematic-for-comparing-web-backend-application-frameworks-with-regards-to-performance-and-scalabi

Echeverría Perez, D., & Abella Paumier, A. (s.f.). *Testing como Práctica para Evaluar la Eficiencia en Aplicaciones Web.* Habana: Centro Nacional de Calidad de Software.

Eloff, E., & Torstensson, D. (2012). An Investigation into the Applicability of Node.js as a Platform for Web Service. *Linköping: Institutionen för datavetenskap*.

Erb, B. (Abril de 2012). Concurrent Programming for Scalable Web Architectures. *Universität Ulm*.

Express official website. (s.f.). *Express official website.* Obtenido de Express official website: https://expressjs.com

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. PhD thesis. California: University of California.

Freitas, A., & Renata, V. (2014). An Ontology for Guiding Performance Testing. *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 401 - 407.

Gil, A. (2002). *Como elaborar projetos de pesquisa. In: Como elaborar projetos de pesquisa.* Atlas.

Haldar, M. (2017). *RESTful API Designing guidelines — The best practices.* Obtenido de hackernoon.com: https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, M. (2010). *Metodología de la Investigación.* Ciudad de México: McGRAW-HILL.

Herrera, L. E., Medina, A. F., & Naranjo, G. L. (2008). *Tutoría de la investigación científica: Guía para elaborar en forma creativa y amena el trabajo de graduación* (Primera ed.). Quito: Empresdane Gráficas.

Herron, D. (2013). Node Web Development. *Packt Publiching Ltd*.

Hungler, B. P., Tatano, C., & Polit, D. F. (2004). *Fundamentos de Pesquisa em Enfermagem Métodos, Avaliação e Utilização* (5 ed.). Porto Alegre: Artmed. doi:ISBN 85-7307-984-3

Ihrig, C., & Bretz, A. (2015). Full Stack Development with MEAN. *Cambridge, AUS: Site-Point Pty. Ltd*.

Islam, M., Islam, M., Islam, M., & Halim, T. (2011). A Study of Code Cloning in Server Pages of Web Applications Developed Using Classic ASP.NET and ASP.NET MVC Framework. *Proceedings of 14th International Conference on Computer and Infonnation Technology*, 22-24. doi:10.1109/ICCITechn.2011.6164840

Johnson, R. E., & Foote, B. (1988). Designing Reusable Classes. *Journal of Object-Oriented Programming, 1*, 22-35.

Kalita, M., Khanikar, S., & Bezboruah, T. (2011). Investigation on performance testing and evaluation of PReWebN: a Java technique for implementing web application. *IET Software, V,* 434 - 444. doi:10.1049/iet-sen.2011.0030

Kitchenham, B., & Charters, S. M. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. *Technical Report EBSE 2007-001*.

Koffel, W. (01 de Diciembre de 2013). *Choosing A Web Framework.* Recuperado el 23 de Septiembre de 2017, de Clearlytech: http://www.clearlytech.com/2013/12/01/choosing-web-framework/

Koffel, W. (1 de Diciembre de 2013). *clearlytech.* Recuperado el 10 de 07 de 2017, de clearlytech: http://www.clearlytech.com/2013/12/01/choosing-web-framework/

Kubala, J. (Diciembre de 2017). Progressive web app with Angular 2 and ASP.NET. JAMK University of Applied Sciences.

Li, A., Zong, X., Kandula, S., Yang, X., & Zhang, M. (2011). Cloudprophet: towards application performance prediction in cloud. *Proceedings of the ACM SIGCOMM 2011 conference*, 426 - 427.

Liew, S., & Su, Y. (2012). Cloudguide: Helping users estimate cloud deployment cost and performance for legacy web applications. *In Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th Int. Conf. on*, 90 - 98.

Linden, G. (2006). *Make data useful.* Obtenido de http://glinden.blogspot.se/2006/11/

Marconi, M. D., & Lakatos, E. M. (2003). *Fundamentos de Metodología Científica* (5ta ed.). São Paulo: ATLAS.

Mascarenhas, S. A. (2012). *Metodologia Cientifica.* Pearson Education do Brasil. doi:ISBN 8564574594

McMurray, R. (s.f.). *Installing IIS 8 on Windows Server 2012: The Official Microsoft*. Obtenido de http://www.iis.net/learn/get-started/whats-newin-

Mesbah, A., & van Deursen, A. (2007). *Migrating Multi-page Web Applications to Singlepage.* Amnsterdam: IEEE.

Microsoft. (s.f.). *IIS 8.0 WebSocket Protocol Support: The Official Microsoft IIS*. Obtenido de http://www.iis.net/learn/get-started/whats-new-iniis-8/iis-80-websocket-protocol-support

Mikowski, M., & Powell, J. (2013). *Single Page Web Applications JavaScript end-to-end.* Manning. doi:ISBN 9781617290756

Miller, R. (27 de Octubre de 2016). *Entity Framework Core Quick Overview.* Recuperado el 21 de Febrero de 2018, de Microsoft Docs: https://docs.microsoft.com/en-us/ef/core/#comments-container

Miller, R. B. (s.f.). Response time in man-computer conversational transactions. *Proc. AFIPS Fall Joint Computer Conference, 33*, 267-277.

Minitab 18. (2017). *Minitab Web Site*. Obtenido de https://support.minitab.com/es-mx/minitab/18/help-and-how-to/statistics/basic-statistics/supporting-topics/basics/what-is-a-confidence-interval/

Nodejs.org. (14 de 02 de 2018). *Nodejs.org*. Obtenido de Nodejs.org: https://nodejs.org/en/

Patil, S. S., & Joshi, S. (Agosto de 2012). Identification of Performance Improving Factors for Web Application by Performance Testing. *International Journal of Emerging Technology and Advanced Engineering, II*, 433-436.

Pautasso, C., & Wilde, E. (2009). Why is the Web Loosely Coupled?: A Multi-faceted Metric for Service Design. *In Proceedings of the 18th nternational Conference on World Wide Web* (págs. 911 - 920). New York: ACM.

Peacock, R. (2000). Distributed architecture technologies. IT Professional 2.

Petticrew, M., & Roberts, H. (2008). *Systematic Reviews in the Social Sciences: A Practical Guide.* Blackwell Publishing. doi:10.1002/9780470754887

Pressman, R. (2010). *Ingeniería del Software; un enfoque práctico* (Séptima ed.). México D.F., México: McGraw-Hill.

ProgrammableWeb. (2013). *ProgrammableWeb*. Recuperado el 28 de Septiembre de 2016, de http://www.programmableweb.com/api-research

Rak, M., Suri, N., Luna, J., Petcu, D., Casola, V., & Villano, U. (2013). Security as a service using an SLA-based approach via SPECS. *In Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th Int. Conf. on, 2*, 1-6.

Rak, M., Turtur, M., & Villano, U. (2015). Early prediction of the cost of HPC application execution in the cloud. *In Proceedings of SYNASC 2014,*, 409 - 416.

Richardson, L., & Ruby, S. (2007). *RESTful Web Services.* (M. Loukides, Ed.) Sebastopol, California, Estados Unidos: O'Reilly Media.

Roth, D., Anderson, R., & Luttin, S. (12 de 12 de 2017). *Microsoft.* Obtenido de Docs.Microsoft: https://docs.microsoft.com/en-us/aspnet/core/index

Secretaría Nacional de la Administración Pública. (2013). *Plan de Gobierno Electrónico 2014-2017.* Quito.

Shanker Maurya, L., & Shankar, G. (2012). MAINTAINABILITY ASSESSMENT OF WEB BASED APPLICATION. *Journal of Global Research in Computer Science, 3*. doi:ISSN: 2229-371X

Sharma, U., Shenoy, P., Sahu, S., & Shaikh, A. (2011). Kingfisher: Cost-aware elasticity in the cloud. *In 2011 Proceedings IEEE INFOCOM*, 206 - 210.

Smith, C. U., & Williams, L. G. (2002). *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Boston: Addison-Wesley.

Tuya Gonzalez, J., Ramos Román, I., & Dolado Cosín, J. (2007). *Técnicas cuantitativas para la gestión en la ingeniería del software: Técnicas y prácticas en las pruebas de software.* Madrid: Netbiblo.

Valdez, C. (2016). Why should I use Node.js: The Non-blocking Event I/O Framework? *RED HAT DEVELOPER PROGRAM*. Recuperado el 14 de Febrero de 2018, de RED HAT DEVELOPER PROGRAM: https://developers.redhat.com/blog/2016/08/16/why-should-i-use-node-js-the-non-blocking-event-io-framework/

Walther, S. (19 de Agosto de 2008). *Understanding Models, Views, and Controllers (C#).* Recuperado el 21 de Febrero de 2018, de Microsoft Docs: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/understanding-models-views-and-controllers-cs

Wilkhu, S. (2017). *Developing Single page application with best practices.* Aurburn: Auburn University.

Woodside, M., Franks, G., & Petriu, D. C. (2007). The Future of Software Performance Engineering. *Future of Software Engineering*. doi:10.1109/FOSE.2007.32

Yuni, J. A., & Urbano, C. A. (2014). *Técnicas para investigar: recursos metodológicos para la preparación de proyectos de investigación.* Córdoba: Brujas.

Zhu, K., Fu, J., & Li, Y. (2010). Research the performance testing and performance improvement strategy in web application. *2010 2nd international Conference on Education Technology and Computer (ICETC)*, 328 - 332.

ANEXOS

Historias de Usuario

Tabla 13.- Historias de Usuario

Nro.	Código Épica	Épica	Código Tema	Tema	Código HU	HU	Descripción
1	PYO	Personas y Organizaciones	DIR	Directorio de Actores Sociales	PYO-DIR-1	Registro de organizaciones sociales	Información de actores sociales, organizaciones sociales, personas.
2	PYO	Personas y Organizaciones	DIR	Directorio de Actores Sociales	PYO-DIR-2	Registro de personas	Información de personas
3	PYO	Personas y Organizaciones	DIR	Directorio de Actores Sociales	PYO-DIR-3	Tipos de mecanismo de contacto	Teléfono, Correo Electrónico Dirección Física
4	PYO	Personas y Organizaciones	DIR	Directorio de Actores Sociales	PYO-DIR-4	Información de contacto	El registro de direcciones, teléfono, correo, etc

Nota: Elaboración propia.

Servicios

Tabla 14.- Lista de Servicios del backend.

Routers	Acción	Operaciones Rest	Ruta	Cabecera	Campos	Descripción
Entidad	Read	Get	/entidades/			Lista todas las entidades.
Entidad	Read	Get	/entidades/:id			Busca la entidad según entidadId.
Entidad	Create	Post	/entidades/	Content-Type:application/json	{ entidadcreadoPor":"gvalencia" }	Crea la entidad.

Nota: Elaboración propia.