UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA (UNED)
E.T.S. DE INGENIERÍA INFORMÁTICA

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

Itinerario de Ingeniería del Software (Código 31105128)

TÍTULO: Análisis comparativo para aplicaciones web basados en servicios REST: stack MEAN y stack Java EE.

AUTOR: Jaime Sayago Heredia

DIRECTOR: Ismael Abad Cardiel

Curso Académico 2017-18

Convocatoria de Junio

Máster Universitario de Investigación En Ingeniería de Software y Sistemas Informáticos

ITINERARIO: Ingeniería del Software

CÓDIGO DE ASIGNATURA: 31105128

TÍTULO DEL TRABAJO: Análisis comparativo para aplicaciones web basados en servicios REST: stack MEAN y stack Java EE.

TIPO DE TRABAJO: Tipo B, proyecto específico propuesto por el alumno.

AUTOR: Jaime Sayago Heredia

PASAPORTE: 0301435012

DIRECTOR: Ismael Abad Cardiel

Miembros del Tribunal
Fecha de Lectura y Defensa
Calificación

DECLARACIÓN JURADA DE AUTORÍA DEL TRABAJO CIENTÍFICO, PARA LA DEFENSA DEL TRABAJO DE FIN DE MÁSTER

Fecha: 17/06/2018

Quien suscribe:

AUTOR: Jaime Sayago Heredia

PASAPORTE: 0301435012

Hace constar que es el autor del trabajo: "Análisis comparativo para aplicaciones web basados en servicios REST: stack MEAN y stack Java EE".

En tal sentido, manifiesto la originalidad de la conceptualización del trabajo, interpretación de datos y la elaboración de las conclusiones, dejando establecido que aquellos aportes intelectuales de otros autores se han referenciado debidamente en el texto de dicho trabajo.

DECLARACIÓN:

✓ Garantizo que el trabajo que remito es un documento original y no ha sido publicado, total ni parcialmente por otros autores, en soporte papel ni en formato digital.

✓ Certifico que he contribuido directamente al contenido intelectual de este manuscrito, a la génesis y análisis de sus datos, por lo cual estoy en condiciones de hacerme públicamente responsable de él.

✓ No he incurrido en fraude científico, plagio o vicios de autoría; en caso contrario, aceptaré las medidas disciplinarias sancionadoras que correspondan.

Firmado: Jaime Sayago Heredia

Impreso TFdM05_Autor. Autorización de publicación

y difusión del TFdM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los Autor/es

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Resumen

El uso de los servicios web REST (Representational State Transfer Protocol) ha aumentado y se ha convertido en la implementación más utilizada en la actualidad. La parte más crucial en un proyecto de desarrollo web basado en servicios REST es la elección de las herramientas correctas para el front-end, back-end y entorno de base de datos. El objetivo principal de este trabajo es realizar la comprobación del stack MEAN como solución efectiva para una aplicación web basada en servicios REST. JavaScript ha sido el lenguaje de programación del lado del cliente que se ejecuta en el navegador. La stack MEAN está conformada por MongoDB (base de datos), Node.js. (servidor web), Express (back-end) y Angular (front-end) para construir aplicaciones web. Comparamos la stack MEAN con una stack muy popular como lo es Java EE que agrupa a MongoDB (base de datos), Apache/Tomcat (servidor web), Spring boot (back-end) y JSP/HTML/CSS (front-end), con respecto a sus componentes, características y configuración del entorno. Desarrollamos dos aplicaciones similares construidas con las stacks MEAN y Java EE. Comparamos y analizamos varias características y criterios para pruebas como el tiempo de respuesta, rendimiento y carga de datos.

Palabras clave

Arquitectura software, Javascript, REST, Servicio Web, Java EE

INDICE DE CONTENIDOS

Lista de figuras

Lista de tablas

1. Introducción

La sociedad ha cambiado hacia una mayor dependencia de las tecnologías web que han llegado a convertirse en parte de nuestra vida diaria. Motivando a la academia y a la industria a buscar soluciones para mejorar su disponibilidad, rendimiento y escalabilidad.

De esta evolución surgieron los llamados servicios web que se podrían definir como un módulo de software ofrecido por un proveedor de servicios, disponible a través de la web y que es un elemento clave en la integración de sistemas de diferentes plataformas, lenguajes de programación y tecnologías [1].

Una de las implementaciones más utilizadas en la actualidad para servicios web es REST (Protocolo de transferencia de estado representacional) o RESTful. REST fue creado por Roy Fielding y propone un estilo de arquitectura orientada al diseño de software basado en la red [2] . Se puede definir como una arquitectura cliente/servidor. El cliente envía la solicitud al servidor, el servidor procesa la solicitud y devuelve la respuesta. Estas solicitudes y respuestas se basan en la transferencia de representaciones de recursos que se identifica a través de URIs [3].

El uso de servicios web REST han ido en aumento, dando surgimiento a varias herramientas, frameworks y stacks para desarrollo, integración y ejecución de aplicaciones web, las más utilizadas en la actualidad son la stack MEAN con Javascript y la stack de Java EE.

El problema radica que al momento de elegir la arquitectura apropiada de tecnologías para desarrollo, integración y ejecución de aplicaciones web basados en tecnologías conocidas y probadas como son los servicios web REST. La elección del stack correcto no es fácil, ya que muchos parámetros tienen que considerarse en la base de datos, en el lado del servidor o el lado del cliente, etc.

Diversos autores presentan a la stack MEAN como la mejor solución [4] [5] [6]. En este trabajo se plantea una comparación entre la stack MEAN y la stack Java EE y proponer cuál stack es la solución más efectiva para el desarrollo de aplicaciones web basados en servicios web REST.

Se realizó la implementación de una aplicación web basada en servicios REST para el manejo de la hoja de vida y gestión docente para el departamento de talento humano de la Pontificia Universidad Católica del Ecuador PUCE, Sede Esmeraldas – Ecuador, que permite almacenar, gestionar y analizar la información de los docentes y su gestión docente semestral. Esta aplicación web fue construida utilizando las dos stacks para probar y analizar los pros y contras de las dos stacks, junto con algunas pruebas de evaluación comparativa.

Las características y los criterios para pruebas como el tiempo de respuesta, rendimiento y carga de datos se obtuvieron del análisis de estudios previos para adquirir las herramientas y criterios más óptimos para la comparativa entre las dos aplicaciones web construidas.

Los resultados de las pruebas obtenidas y características analizadas evidenciaron que stack, es la mejor opción para el desarrollo de aplicaciones web basadas en servicios REST.

1.1. Objetivos

El objetivo principal es:

Realizar la comprobación del stack MEAN como solución efectiva para una aplicación web basada en servicios REST.

Los objetivos específicos son:

- Desarrollo de una aplicación web con servicios REST utilizando las stacks MEAN y Java EE.
- Selección y uso de herramientas de evaluación de calidad para una aplicación web basada en servicios REST
- Propuesta de un marco de métricas de calidad de servicios web REST.
- Análisis de resultados de aplicación de pruebas a las aplicaciones web desarrolladas.

1.2. Estructura

El documento está dividido en los siguientes bloques:

- Capítulo 1. Introducción, objetivo general junto con los específicos, la estructura del documento y metodología a utilizar.
- Capítulo 2. Estado del arte sobre arquitecturas web, servicios web, tecnología REST y los componentes y arquitecturas de la stack MEAN y la stack Java EE.
- Capítulo 3. Analizamos las características, ventajas y desventajas de cada una de las stacks MEAN y Java EE y sus respectivas arquitecturas.
- Capítulo 4. Configuración, ambientes y comentarios del desarrollo de una aplicación web utilizando las dos stacks para manejo de la hoja de vida y gestión docente para el departamento de talento humano de la PUCE, Sede Esmeraldas – Ecuador.
- Capítulo 5. Comparamos y analizamos los stacks, utilizando diferentes pruebas como el tiempo de respuesta, rendimiento y carga de datos.
- Capítulo 6. Interpretación de resultados de pruebas, obteniendo conclusiones, recomendaciones y limitaciones.

1.3. Metodología

Entendiendo la metodología como el conjunto de procedimientos racionales utilizados para alcanzar los objetivos. En este documento se realiza un estudio descriptivo de características cuantitativas de las tecnologías MEAN y Java EE, aplicando el método lógico inductivo, obteniendo valores medibles a través de métricas que brinden resultados para obtener el objetivo principal que conocer que tecnología es una solución efectiva para una aplicación web basada en servicios REST. Esta investigación también es experimental, debido a que se manejan las variables necesarias para producir un resultado, y a partir de esto, comprobar el objetivo principal que es buscar la mejor solución para una aplicación web basada en servicios REST. En este caso, con las herramientas de evaluación seleccionadas se obtuvo información que fue procesada y analizada para determinar si es factible considerar que stack de desarrollo es óptimo para una aplicación web basada servicios REST.

2. Estado del arte

2.1. Arquitecturas de software

Cada sistema informático, grande o pequeño, está formado por piezas que están unidas entre sí. Puede haber un pequeño número de estas piezas, o tal vez solo una, o puede haber docenas o cientos; y este vínculo puede ser trivial, o muy complicado, o en algún punto intermedio, es decir cada sistema tiene una arquitectura [7]. El estudio de la arquitectura de software, que formalmente aparece desde 1990, se basa en gran parte en un estudio de la estructura del software que comenzó en 1968 con el documento de sistema operativo histórico de Edsger Dijkstra que señala que vale la pena preocuparse por cómo se particiona y estructura el software, en lugar de simplemente programar para producir un resultado correcto [8]. Los bases de este campo de la Ingeniería de software y sus principales precursores como son David Parnas, Fred Brooks, Dijkstra, y entre otros [9], todo el trabajo en el campo de la arquitectura de software puede verse como evolucionando hacia un paradigma de desarrollo de software basado en principios de construcción de sistemas a gran escala que interactúan y logran el objetivo del sistema.

Al tratar de definir el concepto de Arquitectura de Software existen varias definiciones alternativas o contrapuestas. Pero hay algunas que son reconocidas, a continuación, las citamos. Se puede definir la arquitectura de software como la descomposición de un sistema de nivel superior en cada uno de sus componentes principales, las interfaces y su comunicación [10]. Además, se tiene: “Una arquitectura es el conjunto de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y sus interfaces que componen el sistema, junto con su comportamiento tal como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y comportamentales en subsistemas progresivamente más grandes, y el estilo arquitectónico que guía dicha organización” [11]. Igualmente Clements se refiere a grandes rasgos, a una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema [8]. La IEEE la define de la siguiente manera: Es la organización fundamental de un sistema encarnada en sus componentes,

las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución [12].

Dentro de las ventajas de la arquitectura de software, podemos mencionar que: simplifica la capacidad de comprender sistemas complejos planteando limitaciones en el diseño, reutiliza componentes en múltiples niveles, separa elementos, lo que permite mayor facilidad en el cambio y mantenimiento del desarrollo, aplica utilización de patrones [13]. El uso de arquitectura de software puede realizarse de una manera eficiente en función del tiempo y ser rentable ya que permite la reutilización de los componentes y patrones de diseño en los proyectos [14].

Al implementar una arquitectura de software es fundamental el uso de los patrones arquitectónicos que brindan un principio general de estructura. Un patrón arquitectónico expresa un esquema de organización estructural fundamental para los sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para organizar las relaciones entre ellos [15].

2.2. Patrones arquitectónicos

Uno de los aspectos más importantes en el desarrollo de sistemas es la experiencia. En este sentido, uno de los mayores avances de que se dispone, es la información, técnicas y modelos recopilada en el tiempo sobre problemas que se han ido presentando a diferentes investigadores para formular la solución a realizar y que sirve para el diseño de la arquitectura [16]. El propósito de un patrón de software es compartir una solución probada y ampliamente aplicable a un problema de diseño particular en una forma estándar que permita que se reutilice fácilmente [15]. Los patrones de software generalmente se organizan en tres grupos: estilos arquitectónicos que registran soluciones para organización a nivel de sistema, patrones de diseño que registran soluciones a problemas detallados de diseño de software y expresiones idiomáticas que capturan soluciones útiles para problemas específicos del lenguaje [7].

2.3. Estilos arquitectónicos

Un estilo de arquitectura se define como como una familia de sistemas con patrones comunes de una organización estructurada [17]. Un estilo arquitectónico expresa un esquema de organización estructural fundamental para los sistemas de software y proporciona un conjunto de tipos de elementos predefinidos, especifica sus responsabilidades e incluye reglas y pautas para organizar las relaciones entre ellos [7]. Un estilo arquitectónico proporciona un conjunto de principios organizacionales para el sistema como un todo, en vez de que tratar a cada componente como una parte del sistema. La solución descrita por un estilo arquitectónico generalmente se define en términos de tipos de elementos arquitectónicos y sus interfaces, tipos de conectores y restricciones sobre cómo se deben combinar los elementos y los conectores [7].

Son útiles durante el análisis y diseño de un sistema ya que forman parte fundamental de un software, porque puede determinar el estilo que más se ajuste para su construcción [15]. Están compuesto por: Un conjunto de tipos de bloques, una topología, un conjunto de conectores que interactúan para la comunicación y coordinación de estos bloques y un conjunto de restricciones semánticas que determinan cómo los elementos coordinan a través de la topología establecida [18].

El uso de estilos puede mejorar algunos atributos de calidad definidos para un sistema, pero disminuir otros, un ejemplo es el uso de un estilo en capas, aumenta la flexibilidad del software, pero generalmente disminuye el rendimiento, por eso es importante implementar y adecuar un estilo que se adapte con las necesidades del sistema [7]. Los estilos arquitectónicos, definen importantes decisiones sobre los elementos arquitectónicos y su relación con la construcción de un sistema. Estos estilos se pueden usar para restringir la arquitectura como para coordinar a los arquitectos que interactúan en su desarrollo [19].

Garlan y Shaw propusieron la primera taxonomía de los estilos de arquitectura [17]. Rozanski y Woods realizaron una recopilación de los estilos de uso más frecuente [7]. A continuación, se consideran los siguientes tipos:

- Tuberías-filtros.
- Organización de abstracción de datos y orientación a objetos

- Invocación implícita, basada en eventos
- Sistemas en capas
- Repositorios
- Procesos distribuidos, ya sea en función de la topología (anillo, estrella, etc.) o de los protocolos entre procesos. Una forma particular de proceso distribuido es, por ejemplo, la arquitectura cliente-servidor.
- Organizaciones programa principal / subrutina. ·
- Arquitecturas software específicas de dominio.
- Sistemas de transición de estados.
- Sistemas de procesos de control.
- Estilos heterogéneos.

Se puede apreciar que existen una gran variedad en la taxonomía de los estilos de arquitecturas, no se describen todos los estilos mencionados, debido a que la tesis se enfoca en los estilos de arquitectura orientados a la web, que en los siguientes apartados se van a tratar con mayor detalle.

2.4. Estilos arquitectónicos orientados a la web

Chacón y Cárdenas en su estudio proponen una taxonomía de los estilos de arquitectura software orientados a la web [20], que a continuación detallamos:

2.4.1. Cliente / servidor

El modelo cliente/servidor, también documentado como un estilo arquitectónico [10], tiene una importancia significativa en la arquitectura de software moderna, el usuario opera programas de aplicación (clientes) en su ordenador y estos programas acceden a los recursos del servidor que son compartidos y puestos a disposición centralmente, su modelo se basa en un esquema simple de solicitud-respuesta, esto significa que los archivos ya no tienen que ser transferidos como un todo [18]. Por ejemplo, el servidor a menudo accede a una base de datos y ejecuta una consulta especificada en la solicitud del cliente que puede ser una petición a través de una página web.

Esta arquitectura implica la existencia de una relación entre procesos que solicitan servicios (clientes) y procesos que responden a estos servicios (servidores) [10]. Estos

dos tipos de procesos pueden ejecutarse en el mismo procesador o en distintos ya que la arquitectura cliente/servidor permite la creación de aplicaciones distribuidas, la principal ventaja de esta arquitectura es que facilita la separación de las funciones según su servicio, permitiendo situar cada función en la plataforma más adecuada para su ejecución [21]. Además permite que múltiples procesadores ejecuten partes de una misma aplicación a través de red, junto con escalabilidad horizontal, escalabilidad vertical y posibilita el acceso a los datos independientemente de donde se encuentre el usuario [21].

2.4.2. Arquitectura orientada a servicios (SOA)

La arquitectura orientada a servicios (SOA) es una tecnología neutral, independiente de la plataforma, que proporciona reutilización, agilidad, acoplamiento e interoperabilidad con la ayuda de un conjunto de servicios. La principal ventaja del desarrollo de servicios es que proporcionan una forma común de interacción. Los servicios pueden interactuar independientemente de las distintas plataformas [22]. SOA es el producto de la evolución de plataformas de tecnología que se solían utilizar con frecuencia, preservando las características exitosas de las arquitecturas tradicionales. Podemos entender a las arquitecturas SOA como un estilo de diseño que guía los aspectos de creación y uso de servicios de negocio a través de su correspondiente ciclo de vida. Además, define y aprovisiona la infraestructura de tecnologías de la información que permite a diferentes aplicaciones intercambiar datos y participar en los procesos de negocio, independientemente del sistema operativo o de los lenguajes de programación con los cuales estos servicios (y aplicaciones) fueron desarrollados e implementados [23]. Una aplicación SOA la podemos dividir en tres capas. La capa de recepción de peticiones (servicios controladores: encargados de recibir las peticiones de los clientes y realizar las llamadas necesarias a otros servicios, en la secuencia adecuada, para devolver una respuesta), la capa de tareas (servicios de negocio: representan una tarea de negocio) la capa de lógica reutilizables (servicios de utilidad: representan tareas muy reutilizables como tareas de negocio, acceso a datos...) [23].

2.4.3. Broker

El estilo arquitectónico Broker se puede utilizar para estructurar sistemas de software distribuidos con componentes desacoplados que interactúan mediante invocaciones de

servicios remotos. Es responsable de coordinar la comunicación, como por ejemplo el reenvío de solicitudes. así como para la transmisión de resultados y excepciones [15]. Broker se utilizó para especificar Common Object Request Broker Architecture (CORBA) definida por el Object Management Group [24]. CORBA es una tecnología orientada a objetos para la distribución de objetos en sistemas heterogéneos [15] . Un lenguaje de definición de interfaz está disponible para soportar la interoperabilidad de los objetos cliente y servidor [25]. El estilo arquitectónico Broker tiene algunos beneficios importantes tales como transparencia, manejabilidad, extensibilidad de componentes y reutilización, también presenta algunos inconvenientes como que la aplicaciones sean más lentas y con baja tolerancia a fallos [15].

2.4.4. Peer to peer

Cliente/servidor es un estilo arquitectónico que es una especialización del estilo de invocación explícita general [10]. Esto significa que el cliente se comunica con el servidor a través de llamadas directas y explícitas y recibe respuestas a estas llamadas del servidor. Una alternativa es el estilo peer-to-peer [10], que también se especializa en la invocación explícita, pero se centra en la comunicación directa entre clientes en lugar de la comunicación a través de un servicio central [18]. La Arquitectura Peer-to-peer (P2P) usa una serie de pares iguales, en la estructura de arquitectura P2P pura no hay servidor central [7]. Cada par puede ofrecer y consumir servicios en la red, todo el estado del sistema se reparte entre sus pares [15]. En la arquitectura P2P, puede añadir o eliminar un servicio en cualquier momento, por lo tanto, los clientes deben averiguar qué servicios están actualmente disponibles antes de utilizar un servicio [18]. No todos los sistemas P2P son sistemas P2P "puros, la mayoría de los sistemas son híbridos y utilizan servidores centrales para determinados servicios, por ejemplo, como puntos de acceso a la red [18]. Como ventajas, los sistemas P2P eliminan el posible punto de fallo que representa un servidor central, se puede hacer muy escalable, y son resistentes a fallos parciales en la red y como desventaja de los sistemas P2P incluyen la posible división de la red si no se dispone de una lista central de pares y la dificultad de garantizar una respuesta particular del sistema en cualquier momento [7].

2.4.5. REST

El estilo de Arquitectura REST utiliza identificador de recursos uniforme URI para acceder y manipular los estados de los recursos en cual se proporciona un identificador único para un recurso web [3]. REST normalmente tiene muchos recursos y es una arquitectura orientada a los recursos [23]. En REST se acceden a los recursos mediante una interfaz genérica la cual reduce dramáticamente la complejidad de la semántica de la interfaz durante la interacción de servicios. Las necesidades de ancho de banda y recursos de REST son bajos, por lo que se considera ligero [26]. Las aplicaciones desarrolladas utilizando REST son llamadas aplicaciones RESTful que proporcionan más flexibilidad y menos gastos generales, lo que resulta en una mejor solución para la mayoría de las implementaciones [27]. A continuación, profundizaremos en la descripción de esta arquitectura.

2.5. Servicios web

El término servicios web según la W3C (World Wide Web Consistorium) proporcionan un medio estándar de interoperabilidad entre diferentes aplicaciones de software que se ejecutan en una variedad de plataformas y/o marcos de trabajo [28]. Un servicio web es un sistema de software diseñado para soportar la interacción entre máquinas a través de una red. Tiene una interfaz descrita en un formato procesable por máquina. Otros sistemas interactúan con el servicio web de una manera prescrita por su descripción utilizando mensajes, típicamente transmitidos usando HTTP con una serialización en conjunción con otros estándares relacionados con la web [28]. Se podría entender como un conjunto de aplicaciones o tecnologías que pueden interoperar en la web, su principal función es la de intercambiar datos entre sí; tanto los proveedores ofrecen sus servicios y los usuarios solicitan el servicio llamando a estos procedimientos utilizando la web [29]. Estos servicios proveen de mecanismos de comunicación estandarizados entre las distintas aplicaciones, que pueden interactuar entre ellas para entregar la información al usuario. La interoperabilidad y extensibilidad de las aplicaciones y su combinación para operaciones complejas requiere la utilización de un estándar [28]. Existen diversos estilos de servicios web a continuación los describimos.

2.6. Protocolos y estándares de servicios web

Un servicio web para comunicarse implica una determinada tecnología para que pueda interoperar entre cualquier aplicación. Para esto encontramos algunos protocolos y estándares que definen como se comunican, la estructura de la información que se envía y que se recibe, así como su interoperabilidad, etc. Se describen a continuación los protocolos y estándares básicos para que un servicio web puede utilizar:

- XML (eXtensible Markup Language), es un formato o lenguaje de marcado para describir contenido de forma separada de su formato. Dentro de este estándar hay estándares como el DTD o XSD para la definición del lenguaje y el XSL-FO y XSLT para la transformación y presentación [30].
- JSON (JavaScript Object Notation), es un formato o lenguaje ligero de intercambio de datos más simple de leer y escribir por humanos [30].
- SOAP (Simple Object Access Protocol), este protocolo complejo define una gran ática XML con el formato de los documentos que se deben intercambiar entre quien realiza la solicitud (cliente) y quien la responde (servicio) por lo general se utiliza en los servicios web [31].
- REST (Representational State Transfer), es un estándar de transferencia de representación de estado el cual no tiene estado (en inglés stateless), lo que quiere decir que, entre dos llamadas cualesquiera, el servicio pierde todos sus datos [32].
- WSDL (Web Services Description Language), este estándar define la gramática XML pero de la interfaz del servicio web o sea los métodos y parámetros que se exponen tanto a la entrada como salida necesarios para invocar los servicios[31] .

El uso de REST frente a SOAP nos proporciona una mayor escalabilidad del servicio [2]. REST proporciona una arquitectura mucho más fácil, así como peticiones más simplificadas, permitiendo todo ello un mayor rendimiento y velocidad; y permite interpretar las URI de manera visual [22]. REST se ha convertido en la implementación más utilizada en la actualidad [30].

2.7. Arquitectura REST

REpresentational State Transfer (REST) es un estilo arquitectónico propuesto por Fielding [34]. REST es para sistemas hipermedias distribuidos a gran escala y que logra que la World Wide Web (WWW) sea escalable. Fielding argumenta que en REST es la existencia de recursos (elementos de información), que pueden ser accedidos utilizando un identificador global (un Identificador Uniforme de Recurso). Para manipular estos recursos, los componentes de la red (clientes y servidores) se comunican a través de una interfaz estándar (HTTP) e intercambian representaciones de estos recursos (los ficheros que se descargan y se envían). El cliente puede navegar esencialmente a través de una amplia gama de recursos existentes, siguiendo los enlaces de un recurso a recurso [35]. Un principio clave de REST es la interacción sin estado entre los participantes en una conversación. Un estado en este caso significa el estado de la aplicación/sesión. El estado se mantiene como parte del contenido transferido del cliente al servidor/servicio y viceversa [36]. Más concretamente, en el caso de los servicios, los clientes que desean utilizar un servicio acceden a una representación particular de los recursos que representan el servicio mediante la transferencia de contenido de la aplicación utilizando un conjunto pequeño y definido globalmente de métodos remotos [36]. Estos métodos describen la acción a realizar sobre los recursos. Los métodos HTTP para crear, leer, actualizar y borrar recursos, cada uno identificado por un URI, son (PUT, POST, GET, y DELETE en HTTP 1.0, mientras que HTTP 1.1 permite extensiones) [37]. En REST, cada solicitud enviada a un objeto resulta en la transferencia de una representación de este objeto por ejemplo, texto, XML, JSON, etc. [35]. A continuación describe los objetivos del estilo REST [34]:

a) Escalabilidad de las interacciones de los componentes. La web ha crecido exponencialmente sin degradar su rendimiento. Una prueba de ellos es la variedad de clientes que pueden acceder a través de la web: estaciones de trabajo, sistemas industriales.
b) Generalidad de las interfaces. Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial.
c) Despliegue independiente de componentes. Los clientes y servidores pueden ser puestas en funcionamiento durante años. Por tanto, los servidores deben ser capaces de entenderse con clientes actuales y viceversa. Diseñar un protocolo que

permita este tipo de características resulta muy complicado. HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs, a través de la habilidad para crear nuevos métodos y tipos de contenido

d) Componentes intermediarios para reducir la latencia de interacción, reforzar la seguridad y encapsular sistemas heredados.

Los componentes del sistema REST deben cumplir con las siguientes restricciones [34]:

a) Identificación de recursos. Esto se consigue mediante el uso de URI.
b) La manipulación de los recursos mediante representaciones. HTTP es un protocolo centrado en URI. Los recursos son los objetos lógicos a los que se le envían mensajes. Los recursos no pueden ser directamente accedidos o modificados. Más bien se trabaja con representaciones de ellos. Cuando se utiliza un método DELETE para eliminar información.
c) Mensajes autodescriptivos. REST dicta que los mensajes HTTP deberían ser tan descriptivos como sea posible. Esto hace posible que los intermediarios interpreten los mensajes y ejecuten servicios en nombre del usuario. Uno de los modos que HTTP logra esto es por medio del uso de varios métodos estándares, muchos encabezamientos y un mecanismo de direccionamiento.
d) Hypermedia como motor del estado de aplicación. El estado actual de una aplicación web debería ser capturada en uno o más documentos de hipertexto, residiendo tanto en el cliente como en el servidor. El servidor conoce sobre el estado de sus recursos, aunque no intenta seguirles la pista a las sesiones individuales de los clientes. En la Figura 1. Podemos observar una vista del proceso de una arquitectura basada en REST

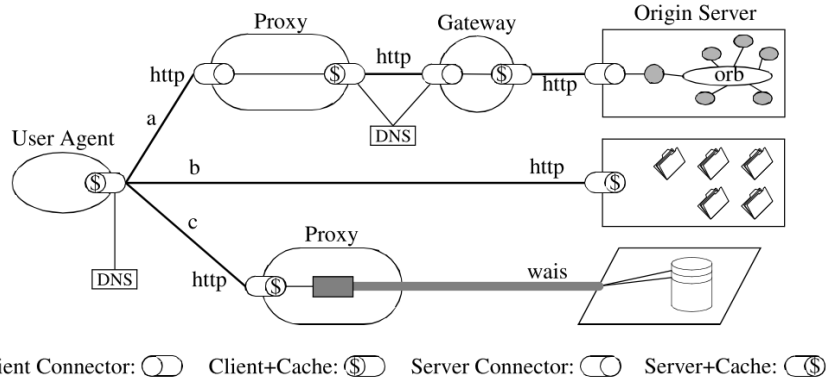

Figura 1. Vista de proceso de una arquitectura basada en REST.

2.8. Servicios web RESTful

RESTful Style Web Services REST es una arquitectura de software distribuida que utiliza todas las funciones web. Puede reducir la complejidad del desarrollo del sistema, proporcionar escalabilidad para el sistema [38]. En el estudio de Hongjun Li, tenemos los siguientes son varios principios clave para los servicios RESTful web [38]:

- Definir el ID para todos los recursos;
- Vincular todos los recursos;
- Utilización de métodos estándar;
- Recursos para múltiples enunciados;
- Comunicación sin estado;

Si las aplicaciones pueden contar con los cinco principios en el diseño del sistema, obtendrán un sistema de alta calidad.

- El ID de los recursos en los servicios RESTful web (ya sea un único recurso, o una colección de recursos; si los recursos virtuales, o recursos físicos) pueden utilizar un concepto unificado y único URI para identificar.
- Todos los recursos de los servicios RESTful web, están disponibles a través de hiperenlaces en forma de URI de referencia dinámica, en cualquier caso, para lograr la interconexión de los recursos de red.
- Todos los recursos en los servicios web RESTful deben ser alcanzados correctamente a través de HTTP, utilizando una interfaz HTTP común, uniforme y no específica. HTTP no sólo permite un posicionamiento único de los recursos, sino que también indica cómo operar el recurso.
- Los servicios web RESTful pueden utilizar HTTP para permitir el procesamiento de datos y llamar la relación entre la separación de características operativas, proporcionando recursos a las múltiples sentencias hechas entre el cliente y el servidor. Los datos HTTP se pueden aplicar a la interacción de recursos.
- Con el fin de mejorar la escalabilidad del sistema, los servicios web RESTful deben ser puestos en el estado de los recursos requeridos, o guardados en el cliente. Esto hace que el cambio de servidor web no sea visible para el cliente.

En esta tesis nos centraremos en realizar la implementación de servicios web RESTful. Y utilizando como representación de recursos a JSON que es el esquema más utilizando en las aplicaciones actualmente por encima de XML que es el estándar por defecto [33] .

2.9. Stacks de desarrollo

Un web stack de desarrollo, también denominado web application stack o conjunto de soluciones (solution stack), define a un paquete de software necesario para el desarrollo de páginas y aplicaciones web. El término stack ("pila") hace referencia al método de apilamiento de los componentes de este conjunto de herramientas, uno encima del otro. La sencilla arquitectura de estos stacks para el desarrollo web se compone de un sistema operativo, un servidor web, una base de datos y un lenguaje de programación. Este conjunto de elementos se encarga, con la ayuda del hardware del servidor correspondiente, de que la información necesaria sobre el proyecto web en cuestión llegue al cliente que la solicita, por defecto el navegador de Internet, el cual procesa los textos en HTML, CSS y JavaScript y pone la web a disposición del usuario [39]. Los componentes de software de un web stack pueden combinarse de forma individual, por lo que existe un gran número de versiones, cada una para diferentes fines. La combinación más conocida, sin embargo, es la pila open source LAMP, resultado de la unión de Linux, Apache, MySQL y PHP [40] .

2.9.1. Componentes de un web stack

El stack tradicional usado como infraestructura para páginas y aplicaciones web se compone, como mínimo, de los cuatro componentes de software mencionados anteriormente: sistema operativo, servidor web, base de datos y un lenguaje de programación, interprete o de scripts. En ocasiones, este paquete primordial se puede completar con frameworks, lenguajes de programación adicionales, servidores proxy o diversos programas de análisis [40]. A continuación, aclaramos los componentes básicos de un stack para el desarrollo web [39]:

- Sistema operativo. Hace referencia al hardware fundamental del servidor (procesador, unidad de memoria, espacio web) y los otros componentes de software. Dicho de otra forma, el sistema operativo recurre a los recursos de hardware para garantizar la funcionalidad de las aplicaciones, en especial del

servidor web. Algunos de los sistemas sobre los que se puede levantar un web stack son Linux, Windows, Unix y Mac OS.

- Servidor web. Es un software instalado en el sistema operativo que entrega documentos al cliente que los solicita. La solicitud del cliente es comunicada al servidor web mediante el protocolo de transmisión HTTP. Si estos documentos solicitados son contenidos estáticos como archivos HTML o imágenes, el servidor web procesa esta petición directamente entregándolos al cliente, pero el contenido dinámico, por el contrario, solo lo puede transmitir con ayuda de bases de datos y módulos de programación. Las soluciones de servidor web más utilizadas incluyen a Apache, Microsoft IIS y nginx.
- Base de datos. Aquí se almacenan, de forma eficiente y a largo término, los datos necesarios para que el servidor web, en el momento requerido, pueda recurrir a ellos. Para ello, el servidor emite peticiones con ayuda de las extensiones del servidor, que la base de datos procesa y devuelve a las extensiones del servidor. La elección de la base de datos está estrechamente ligada al software del servidor web y al lenguaje de programación utilizado. Los más usados son, entre otros, Oracle, MySQL y MongoDB.
- Lenguaje de programación, intérprete de órdenes o script. Sin el lenguaje de programación, el software del servidor web no es capaz de generar páginas dinámicas ni aplicaciones web. Los lenguajes intérpretes actúan, por un lado, en la parte del cliente (en el navegador que realiza la petición) y, por otro, la programación en la parte del servidor hace posible los cálculos necesarios para representar las páginas. El intérprete de lenguaje de programación de un web stack por ejemplo es PHP, aunque actualmente se puede recurrir a JavaScript, Java, ASP.NET, Ruby, Perl o Python.

2.9.2. Stacks más utilizados

En el campo del desarrollo web, hay una serie de stacks de desarrollo que han demostrado su eficacia. Las crecientes exigencias en cuanto a usabilidad, estabilidad y escalabilidad de unas aplicaciones web cada vez más complejas han llevado en los últimos años a que el stack clásico se usara cada vez menos, dado que actualmente existen varios stacks válidos para una aplicación [39]. Presentamos aquí algunos stacks para el desarrollo web:

2.9.2.1. LAMP

Como ya se mencionó anteriormente, LAMP es el web stack más conocido. Como una de las primeras, representó durante mucho tiempo la mejor pila de desarrollo de proyectos web. Todos sus componentes son open source con licencia libre y, de esta manera, no solo modificables, sino también gratuitos [41]:

- Sistema operativo Linux,
- Servidor web Apache,
- Base de datos MySQL,
- Intérprete de lenguaje PHP.

Pese a su condición gratuita, LAMP supo mantenerse firme desde el principio frente a soluciones propietarias equiparable. La stack LAMP está especialmente indicado para la realización de páginas web dinámicas con varias subpáginas. Los usuarios se benefician, sobre todo, de la gran comunidad que lo apoya, de unos tutoriales muy buenos y de la posibilidad de poder implementar PHP o herramientas tan potentes como phpMyAdmin. Debido a los avances técnicos constantes y al desarrollo de nuevo software libre, hoy existen algunas variaciones de LAMP, en las cuales se han sustituido parcialmente los componentes tradicionales. Algunas variaciones de las más conocidas son [39]:

- WAMP (Sistema operativo Windows)
- MAMP (Sistema operativo Mac OS X)
- XAMPP (Sistema operativo a escoger, Perl y PHP como intérprete de lenguaje, además de un servidor FTP independiente de plataforma)
- LAPP (Base de datos PostgreSQL para proyectos empresariales de gran envergadura)

2.9.2.2. WISA

A diferencia de LAMP, cuyos componentes aislados no fueron en principio desarrollados para colaborar entre sí, Microsoft programó los componentes de WISA precisamente con este fin, lo que garantiza el funcionamiento óptimo de este web stack. El software incluido es, lógicamente, propietario, por lo que su uso requiere la previa adquisición de las correspondientes licencias de uso, pero, por otro lado, los usuarios pueden contar con el

amplio soporte prestado por el gigante informático [42]. WISA se compone de estos elementos:

- Sistema operativo Windows Server,
- Internet Information Services (IIS) como software del servidor web,
- Sistema de gestión de bases de datos relacional SQL Server,
- Biblioteca de lenguajes de programación ASP.NET

A través, de la biblioteca ASP.NET se pueden programar proyectos web con Visual C# o Visual Basic.NET, que el sistema puede ejecutar con ayuda de un compilador. De esta forma, WISA no presupone el uso de un intérprete de lenguaje, consiguiendo así una mejor performance frente a los stacks de su clase. Esto convierte al stack de Microsoft en una buena solución para proyectos web complejos y exigentes con cifras remarcables de visitas.

2.9.2.3. MEAN

La stack MEAN sigue un concepto completamente diferente al del web stack clásico. Debido a su composición, la necesidad habitual de diferentes lenguajes de programación para el servidor (PHP, etc.) y para el cliente (JavaScript) desaparece, ya que todos los elementos de este paquete de software soportan JavaScript. El sistema operativo y el software del servidor web pasan a una posición secundaria. Los componentes de un stack MEAN son [43]:

- Base de datos NoSQL MongoDB.
- El framework para back-end es Express.js.
- El framework para el front-end de parte del cliente AngularJS.
- La plataforma Node.js, basada en el entorno de ejecución de JavaScript como servidor web.

Este stack tiene la ventaja de facilitar y acelerar el proceso de desarrollo y de permitir una buena escalabilidad del proyecto web. La transmisión de los datos en formato JSON beneficia, en especial, a la implementación de aplicaciones web de una sola página en el lado del cliente y de aplicaciones móviles. También en este caso existen una serie de variaciones como MEEN, con Ember.js como framework para el front-end, o MERN, con REact.js [39].

2.9.2.4. Java EE

La stack Java EE es una pila perfectamente válida para aplicaciones web. Los años que se encuentra en el mercado brinda una madurez y una excelente productividad, ya que uno de los lugares de un stack donde la productividad es un problema es la vinculación de datos de web a servidor. Es muy versátil para escribir servicios web (basados en REST o API/SOAP) y llevar esos datos al servidor. Para la lógica de negocio, utilizan los estándares de facto de Oracle como EJB. Para la integración ocurre lo mismo con los Websockets (Tyrus) o REST (Jersey). Para el acceso a la base de datos, JPA es el estándar más utilizado. Para el IDE, Eclipse es preferida por la industria, pero igualmente se puede trabajar con IntelliJ, Netbeans, etc. [44]. Java EE tiene varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc. y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para Java EE para componentes. Estas incluyen Enterprise JavaBeans (EJB), servlets, portlets, JavaServer Pages, JPA, CDI y varias tecnologías de servicios web RESTful (JAX-RS) and XML-based web services (JAX-WS) [45]. Los componentes principales de la stack Java EE son [46] :

- Base de datos: Postgresql, MongoDB.
- El framework para back-end utilizado es Spring Boot.
- El framework para el front-end JSP/HTML/CSS.
- La plataforma Apache/Tomcat como servidor web.

Cabe recalcar que estas son unas de las stack actuales de Java EE, existen varias combinaciones de herramientas para formar el stack. Existen distintas herramientas desarrolladas por terceros como por ejemplo Spring Framework, JSF, GWT, Spark, Play Framework, Struts [47].

2.10. Stack MEAN

La stack MEAN es una solución potente y completa. Utiliza el lenguaje de programación para web JavaScript [48]. Comprende cuatro bloques principales: MongoDB como base de datos, Express como marco de trabajo del servidor web, AngularJS como marco de trabajo del cliente web y Node.js como plataforma del servidor. Estos componentes están desarrollados por diferentes equipos e involucran a una comunidad fuerte de

desarrolladores y defensores impulsando el desarrollo y documentación de cada componente. Sin embargo, un problema que podría afectar dramáticamente su proceso de desarrollo y presentar problemas de escalamiento y arquitectura es la conexión de estas herramientas [49].

La principal fortaleza de la pila MEAN radica en su centralización de JavaScript como el principal lenguaje de programación, ya que cada componente está escrito en JavaScript, incluso la base de datos almacena los datos en formato JavaScript Object Notation (JSON) que es el único script que JavaScript entiende completamente [43].

Por lo tanto, JavaScript no sólo se utiliza como lenguaje de scripting del lado del cliente, sino que también se utiliza a lo largo de la aplicación, es decir, en el lado del cliente, del servidor y de la base de datos [4]. El uso de JavaScript como lenguaje principal de programación tanto en el lado del cliente como en el lado del servidor hace que la pila MEAN sea más potente y reduce el tiempo en la construcción de la aplicación [5].
El uso de todo el JavaScript permite dividir la funcionalidad y las tecnologías utilizadas de la siguiente manera [43]:

- Base de datos objetos JSON utiliza MongoDB.
- Servidor web utiliza Node.js.
- Framework web (back-end) utiliza Express.
- Cliente (front-end) utiliza Angular.

2.10.1. MongoDB

MongoDB es un modelo de almacenamiento de documentos NoSQL potente, adaptable y escalable [50]. En MongoDB, los datos se almacenan en la base de datos en un formato JSON llamado BJSON, que significa JSON binario, en lugar de filas y columnas como en la base de datos relacional. MongoDB tiene una capacidad para escalar con varias características que la base de datos relacional proporciona como índices secundarios, clasificación y consultas [51], que proporciona alto rendimiento, alta disponibilidad y escalado automático [49]. Chodorow plantea seis conceptos principales de MongoDB [50] :

1. Se pueden crear cero o más bases de datos dentro de la instancia MongoDB.
2. Una base de datos puede tener cero o más colecciones, donde una colección es similar a una tabla de un sistema de base de datos relacional.
3. Una colección puede tener cero o más documentos, donde los documentos son similares a las filas de tablas.
4. Los documentos pueden tener uno o más campos, donde los documentos son similares a columnas de tablas.
5. Los índices de la MongoDB funcionan como contrapartida a las claves primarias de un sistema de base de datos relacional.
6. Los cursores se utilizan para apuntar el conjunto de los resultados de los datos.

2.10.1.1. Características de mongoDB

MongoDB proporciona muchas de las funcionalidades igual que lo hacen las bases de datos relacionales. Junto con un CRUD (Create, insert, update, delete), las siguientes son algunas características de MongoDB [50]:

- Indexación: MongoDB soporta índices secundarios genéricos, lo que permite una variedad de consultas rápidas y ofrece capacidades únicas, compuestas y de indexación de texto completo.
- Agregación: MongoDB soporta un "pipeline de agregación" que nos permite construir agregaciones complejas a partir de piezas simples y permite que se agilice la base de datos.
- Tipos especiales de recopilación: MongoDB soporta acumulaciones de “tiempo de vida” para los datos que deberían terminar en un momento determinado, como las sesiones. Soporta colecciones de tamaño fijo, que son útiles para almacenar datos recientes, como registros.
- Almacenamiento de archivos: MongoDB soporta un protocolo “fácil de usar” para almacenar archivos grandes y metadatos de archivos.

2.10.1.2. MongoDB dentro de stack MEAN

En el stack MEAN, en MongoDB se almacenan los datos de las aplicaciones. El servidor de Node.js soporta una variedad de bases de datos, así que, MongoDB pueda ser reemplazado por cualquier otra base de datos. Las razones por las que el MongoDB encaja bien en la pila MEAN son las siguientes [52]:

- Orientación de documentos: MongoDB está orientado a documentos y almacena datos en el formato JSON, que es muy similar a lo que se trata en scripts del lado del servidor y del lado del cliente. No requiere la necesidad de transferir datos de filas a objetos y viceversa [51]. Por ejemplo, para crear un documento que contenga información de contacto y una dirección de contacto, no es necesario crear dos documentos separados para la información básica y los detalles de dirección [53].
- Alto rendimiento: Es popular por su alto rendimiento y es capaz de lidiar con el tráfico pesado de usuarios en una aplicación [54].
- Alta disponibilidad: MongoDB mantiene el mismo conjunto de datos y proporciona redundancia y alta disponibilidad manteniendo intacto el alto rendimiento [54].
- Alta escalabilidad: MongoDB proporciona una alta escalabilidad en el rendimiento al considerar sus diversas dimensiones clave como hardware, patrón de aplicación, diseño de esquema e indexación [50]. Su modelo orientado a documentos lo hace más sencillo para que pueda dividir la información en varios servidores. Consecuentemente, MongoDB se ocupa de ajustar la información y las cargas sobre un grupo, redistribuir los documentos de forma natural y dirigir la solicitud del usuario a las máquinas apropiadas [54].
- No inyección SQL. La seguridad de la base de datos es uno de los aspectos más críticos de la seguridad de la información. Las organizaciones se enfrentan a una avalancha de nuevas clases de amenazas y actores de amenazas con phishing, ransomware y robo de propiedad intelectual que crece más del 50% año tras año. Con la inyección SQL, los hackers pueden controlar datos críticos a través de consultas maliciosas. En MongoDB, los datos almacenados son objetos JSON y

se manipulan a través de consultas JSON y no cadenas SQL. Por lo tanto, MongoDB no es susceptible a la inyección de SQL [54].

2.10.1.3. Express

Express es un framework web maduro y flexible para construir aplicaciones web sobre el ecosistema Node. Por defecto, el framework Express utiliza el motor Pug para soportar plantillas [55]. Express es un framework relativamente pequeño que se encuentra en la parte superior de la funcionalidad del servidor web de Node para simplificar sus APIs y añadir nuevas funciones útiles. Facilita la organización de la funcionalidad de su aplicación con middleware y enrutamiento; agrega utilidades útiles a los objetos HTTP de Node y el renderizado de vistas HTML dinámicas; define un estándar de extensibilidad fácilmente implementado [56] .

Express utiliza el módulo HTTP de Node y conecta componentes que se denominan middleware. El middleware es un patrón que permite la reutilización del código dentro de las aplicaciones e incluso lo imparte a otros como módulos NPM. El punto clave del middleware es una capacidad con tres parámetros: request (petición), response (respuesta) y next [55]. La Figura 2 muestra el código de ejemplo de un "Hola mundo" utilizando Express.

```
1  var express = require("express");
2  var app = express();
3  app.get("/", function(request, response) {
4  response.send("Hola, Mundo!");
5  });
6  app.listen(3000, function() {
7  console.log("Express app started on port 3000.");
8  });
```

Figura 2. Hola mundo en express.

Express es fácil de configurar, implementar, controlar y proporcionar varios componentes clave para manejar las solicitudes web. Express ayuda en la creación de aplicaciones web y servidores HTTP simples ya que es un framework mínimo y flexible [56]. En MEAN, Express funciona como un medio para transferir las solicitudes de un cliente a una base

de datos y envía las respuestas de la base de datos al cliente [43]. Express proporciones las siguientes características [56]:

- Framework Ligero para aplicaciones web
- Manejo de rutas específicas
- Soporta Motores de Plantillas
- Soporta carga de archivos
- Permite desarrollar SPA (Singe Page Web Applications)
- Permite desarrollo de aplicaciones en tiempo real
- Manejo de errores

2.10.1.4. AngularJS

AngularJS es una librería escrita en JavaScript para el desarrollo de aplicaciones web, mantenida por Google, es un framework JavaScript de código abierto y aborda los retos de las single-page aplications (SPAs) [57]. Una aplicación web AngularJS sigue el patrón de diseño MVC, que resulta en el desarrollo de aplicaciones web ampliables, mantenibles, comprobables y estandarizadas [57]. La unión de datos AngularJS y la inyección de dependencias lo convierten en un socio ideal para cualquier tecnología de servidor, ya que elimina gran parte del código que de otro modo tendrías que escribir, y todo sucede dentro del navegador [43].

2.10.1.4.1. Características de AngularJS

A continuación se describen los componentes y características de AngularJS [58].

- Módulo: Un módulo es básicamente un contenedor que contiene servicios, controladores, filtros, directivas, etcétera. Cada módulo de AngularJS tiene su propia estructura de carpetas para controladores, directivas, etcétera. Cada página de vista en AngularJS tiene un módulo.
- Scope: Es sólo una representación JavaScript de los datos utilizados para completar una vista en una página web. Estos datos pueden provenir de cualquier fuente, como una base de datos o un servidor web remoto.

- View: Las vistas con plantillas y directivas, son componentes de AngularJS para construir una vista HTML que se presenta al usuario.
- Expression: La capacidad de añadir una expresión dentro de una plantilla HTML es una gran característica. Una expresión está básicamente vinculada a un ámbito de aplicación. Una expresión ayuda a vincular los datos de la aplicación a HTML.
- Controller: Un controlador es un componente de la estructura MVC. Un controlador contendrá su lógica de negocio principal. Su función principal es exponer los datos a la página de vista utilizando el visor.
- Data bindingi: Una característica importante es la encuadernación de datos. Este es el proceso de vincular datos del modelo a la vista y viceversa, además soporta la unión de datos en dos direcciones.
- Services: Los servicios son objetos singleton que proporcionan funcionalidad para una aplicación web.
- Dependency injection: Es el proceso de inyectar la dependencia en tiempo de ejecución (es decir, poner a disposición componentes dependientes para el acceso dentro del componente de código inicializado). Se utiliza para consumir servicios. Por ejemplo, si un módulo requiere acceso a un recurso a través de una solicitud HTTP, entonces el servicio HTTP puede ser inyectado en el módulo para que la funcionalidad esté disponible en el código del módulo.

2.10.1.4.2. Ciclo de vida de AngularJS

El ciclo de vida de AngularJS, que tiene tres fases: bootstrap, compilación y runtime. Estas fases de se producen en el momento de cargar la página web en el navegador del cliente. Para diseñar e implementar el código en una aplicación desarrollada con AngularJS, es importante entender el ciclo de vida. En la primera fase la librería se descarga en el navegador e inicializa el módulo de la aplicación, luego en la segunda fase compila el código y lo enlaza a JavaScript con la biblioteca de AngularJS y la tercera fase se ejecuta siempre y cuando el usuario recargue o navegue fuera de una página web [57]. Utiliza una arquitectura MVC para crear aplicaciones web. La arquitectura MVC es una metodología de programación que tiene como objetivo dividir una aplicación en tres componentes principales: un modelo, una vista y un controlador. Estos tres componentes se combinan para formar su solicitud. La Figura 3 muestra la arquitectura del Model-View-Controller [57].

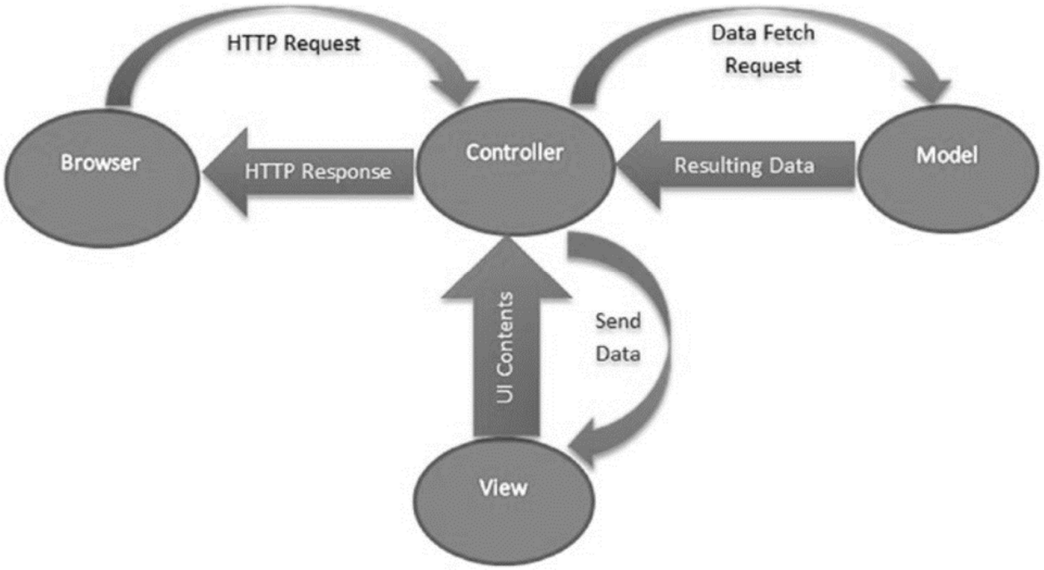

Figura 3. Arquitectura MVC en AngularJS [57].

Cuando un usuario envía una solicitud HTTP a través de un navegador, el controlador recibe la solicitud. El controlador procesa que solicita y envía la solicitud al modelo para proporcionar los datos adecuados. El modelo en respuesta proporciona la matriz de datos resultante al controlador de nuevo. El controlador procesa los datos de nuevo al formato deseado y los envía a la vista. La vista representa los datos a través del contenido de la interfaz de usuario y los envía al controlador. Finalmente, el controlador envía la respuesta HTTP al navegador [57].

- View: Se utilizan para generar una representación de salida de información, como un gráfico o un diagrama, al usuario en un navegador web. AngularJS construye las vistas en el DOM extrayendo todas las plantillas definidas para una aplicación [59].
- Model: Contiene el objeto $scope que se utiliza para almacenar el modelo de aplicación, por lo que no es necesario crear una clase de modelo JavaScript como con otros marcos de trabajo del lado del cliente JavaScript. Los alcances se adjuntan al DOM, lo que ayuda a simplificar considerablemente el problema de JavaScript [59].

- Controller: Es el lugar donde se define toda la lógica de negocio específica de una determinada vista. El controlador mantiene el modelo y la vista juntos [59].

2.10.1.5. Node

Node es un framework de desarrollo desarrollado originalmente en 2009 por Ryan Dahl, basado en el motor JavaScript V8 de Google [60]. Node es una plataforma basada en el tiempo de ejecución JavaScript de Chrome para crear aplicaciones en red rápidas y escalables [49]. Node utiliza un modelo controlado por eventos y de no bloqueo de E/S que lo hace ligero y eficiente, perfecto para aplicaciones que requieren grandes cantidades de datos y que se ejecutan en dispositivos distribuidos [43].
Node es un lenguaje de scripting del lado del servidor que puede ser usado en el lado del servidor, lado del cliente, e incluso puede ser un servidor web. Antes de que existiera el Node, JavaScript se usaba simplemente para la interacción del usuario como script del lado del cliente [61]. Para comunicarse con el servidor, requirió el soporte de otro lenguaje de scripting del lado del servidor como PHP/Perl. Node puede actuar independientemente por sí solo en el lado del servidor. Node incluso ayuda a reducir el tiempo y los recursos de desarrollo y también reduce la interacción entre el servidor web y el script del servidor web, el código del lado del cliente y el código del lado del servidor pueden estar en el mismo idioma [62].

2.10.1.5.1. Características Node

A continuación se presentan algunas de las características importantes que hacen de Node como excelente opción de los arquitectos de software [63]: Todas las APIs de la librería Node son asincrónicas, significa que un servidor basado en Node nunca espera que una API devuelva los datos. El servidor se mueve a la siguiente API después de llamarla y un mecanismo de notificación de eventos de Node ayuda al servidor a obtener una respuesta de la llamada anterior de la API. Es muy rápido al estar construido en el motor JavaScript V8 de Google Chrome, la biblioteca de Node es muy rápida en la ejecución de código. Node usa un modelo simple de hilos con bucle de eventos. El mecanismo de eventos ayuda al servidor a responder de una manera que hace que el servidor sea altamente escalable en comparación con los servidores tradicionales que crean hilos limitados para manejar las solicitudes. Node usa un solo programa que puede proporcionar servicio a un

número mucho mayor de peticiones que los servidores tradicionales como Apache HTTP Server. Además, en Node las aplicaciones nunca almacenan en búfer ningún dato. Estas aplicaciones simplemente emiten los datos en segmentos.

2.10.1.5.2. NPM y paquetes Node

Es el gestor de paquetes para Node. Es un módulo Node que se instala globalmente con la instalación inicial del Node. Por defecto, busca y carga paquetes desde el registro npm [64]. Además de ser un registro de paquetes, también actúa como un recurso fácil de usar para que los desarrolladores descubran nuevos módulos. npm es el corazón del ecosistema del módulo de Node [64]. NPM resuelve las dependencias y el conflicto de versiones en varios módulos. El objetivo principal es facilitar el uso e instalación de módulos que son componentes reutilizables disponibles a través de un repositorio online. NPM actúa como una infraestructura en Nodo para gestionar paquetes descargables desde la web [55].

2.11. Stack Java EE

Java que es uno de los lenguajes de programación más utilizados y que tiene una variedad de herramientas para diferentes propósitos [65]. En este caso se combina varias herramientas y componentes la conformar la stack Java EE. En una aplicación web los componentes están en el front-end o en el back-end [57]. La stack Java EE que es muy utilizada en para el desarrollo de aplicaciones modernas [66]. Estas tecnologías deben trabajar de una manera fácil y armoniosa. En el Front-End se usa JSP/HTML/CSS, utilizando el patrón MVC. En el back-end Spring Boot y la base de datos Postgresql, MongoDB o cualquier otro motor de base de datos. La siguiente figura muestra la comunicación del cliente y el servidor a través de una API RESTful y HTTP.

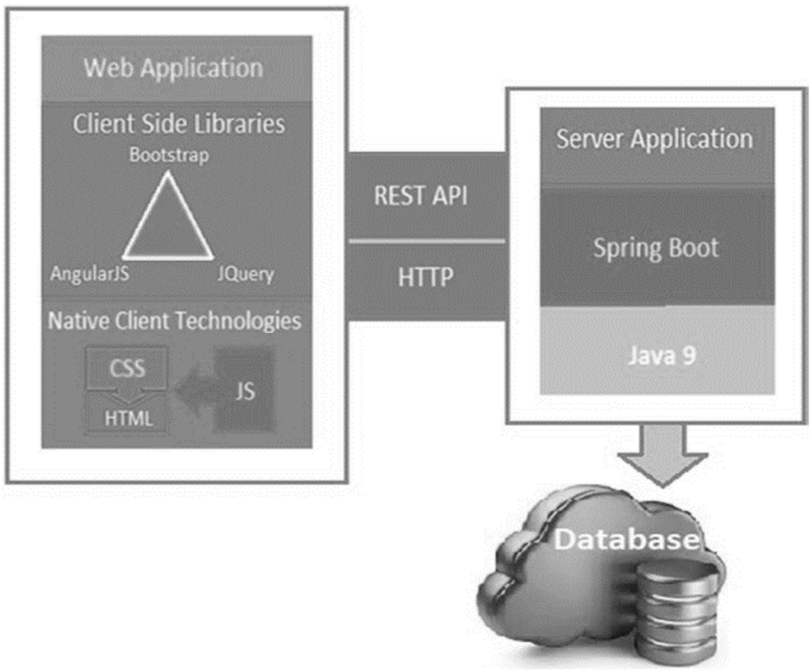

Figura 4. Comunicación entre cliente y servidor [57].

El uso de toda la stack Java EE permite dividir la funcionalidad y las tecnologías utilizadas de la siguiente manera, que conforman la stack seleccionada:

- Base de datos MongoDB.
- Spring boot para el (back-end).
- Cliente (front-end) utiliza JSP/HTML/CSS.
- La plataforma Apache/Tomcat como servidor web.

A continuación, se procede a describir cada una de las tecnologías utilizadas.

2.11.1. Spring boot

Spring Boot es un framework que posee las bibliotecas necesarias para la creación de aplicaciones ejecutables basadas en Spring. Los componentes que posee Spring Boot permiten simplificar los pasos de la selección de las dependencias necesarias para el proyecto a desarrollar y su despliegue en el servidor. No requiere de configuración usando archivos en XML/JSON y permite la creación de servicios REST en Java de manera fácil, centrando al desarrollador en los elementos críticos de desarrollo especifico de su aplicación [67]. Spring boot posee las siguientes características [67]:

- Proporciona una gama de características no funcionales que son comunes a grandes clases de proyectos (servidores integrados, seguridad, métricas, controles de estado, configuración externalizada).
- No hay generación de código y ningún requisito para la configuración XML.
- Proporciona una experiencia de inicio radicalmente más rápida y ampliamente accesible para todo el desarrollo de Spring.
- Proporciona una forma sencilla de configurar y manejar sus dependencias utilizando poms al inicio. Es decir, si va a crear una aplicación web, sólo necesita incluir la dependencia en el archivo start web den su pom de Maven o en el archivo de configuración de Gradle.
- Proporciona anotaciones que le ayudan a incluir, configurar y utilizar tecnologías como bases de datos (SQL y NoSQL), almacenamiento en caché, programación, mensajería, integración de Spring, procesamiento por lotes y más.
- Soporta contenedores de servlets a partir de la versión 3.0 y superior para los servidores de aplicaciones: Tomcat 8.5, Jetty 9.4 y Undertow 1.3.
- El uso de anotaciones para la definición de los servicios y los controladores REST facilita el desarrollo rápido de los servicios aplicando un bajo acoplamiento, esto permite separar las diferentes funcionalidades y descentralizar la solución.
- Por otra parte, provee de forma completa los beneficios asociados al framework de desarrollo Spring MVC

2.11.1.1. Spring security

La seguridad es el proceso de proteger los recursos de usuarios no autenticados y no autorizados y permitir que usuarios específicos (autenticados y autorizados) accedan a estos recursos protegidos. La seguridad es diferente de los cortafuegos, la detección de intrusos, la seguridad de JVM, o cualquier otra cosa [57]. Spring Security está dirigido principalmente a aplicaciones basadas en Spring. Spring Security Seguridad a comenzó inicialmente como Acegi Security y más tarde fue adoptado por Spring como un subproyecto. Se ha convertido en el estándar de facto para la seguridad de las aplicaciones desarrolladas con Spring. Admite autenticación y autorización en el nivel de invocación del método de solicitud HTTP [68] . Además de utilizar JDBC o uno de sus servicios o repositorios para autenticar a los usuarios, Spring Security viene con sistemas integrados para autenticar y (si se desea) autorizar el uso de Microsoft's Active Directory, Jasig's

Central Authentication Service (CAS), Java Authentication and Authorization Service o JAASAS (un Módulo de Autenticación Conectable), o PAM, implementación), LDAP y OpenID, todos los cuales son servicios basados solicitudes. Puede proteger aplicaciones basadas en web y aplicaciones del lado del cliente; sin embargo, está diseñado para aplicaciones web y requiere configuraciones adicionales para su uso en aplicaciones del lado del cliente [69].

2.11.1.2. Spring data rest

Spring MVC ofrece una base sólida para construir los tipos de servicios web REST que son el medio más utilizado para la integración de aplicaciones en la web, pero la implementación de la funcionalidad básica del servicio web REST puede ser tediosa y dar como resultado una gran cantidad de código [70]. Spring Data REST se basa en los repositorios de Spring Data y los exporta automáticamente como recursos de REST. Aprovecha los hipermedios para permitir a los clientes encontrar la funcionalidad expuesta por los repositorios y permite integrar los recursos en la funcionalidad relacionada basada en hipermedios tan fácilmente como sea posible. Spring Data REST es en sí misma una aplicación Spring MVC y está diseñada de tal manera que debe integrarse con sus aplicaciones Spring MVC existentes con muy poco esfuerzo. Una capa de servicios existente (o futura) puede funcionar junto con Spring Data REST con cambios menores. Para su configuración es necesario utilizar Spring boot y para empezar a utilizarlo es necesario la siguiente configuración [71]:

```
dependencies {
    compile 'org.springframework.boot:spring-boot-starter-data-rest:1.4.0.RELEASE'
}
```

Figura 5. Dependencia de inicio de Spring Data Rest.

Spring Data REST brinda una gran ayuda haciendo que la configuración automática escaneará todo el código en busca del repositorio de interfaces y las agregara directamente como una API REST. Junto con esto, utiliza el principio HATEOAS (Hypermedia As The Engine Of Application State) y soporta HAL (Hypertext Application Language) como capa semántica para metadatos (como enlace) en la parte superior. Como

HATEOAS en sí mismo no tiene reglas de cómo se manejan cosas como la vinculación entre recursos, la paginación, las búsquedas y otras tareas relacionadas con los metadatos, existen algunas soluciones [71].

2.11.2. Gradle

Es una herramienta de compilación basada en JVM, usa un lenguaje potente y expresivo Domain Specific Language (DSL) implementado en Groovy en lugar de XML. Debido a que Gradle es nativo de JVM, permite escribir una lógica personalizada sea en el lenguaje Java o Groovy. Sin embargo, escribir los archivos de construcción de Gradle no requieren un conocimiento profundo de Groovy [72]. Gradle tiene mayores ventajas a diferencia de Maven o Ant, una de ellas es que Gradle no usa XML, en cambio Maven y ANT si lo usan, lo cual queda algo corto en expresar el flujo del programa y la lógica condicional. Es preciso mencionar que las mejores características de otras herramientas se combinan en Gradle. Gradle brinda una sofisticada gestión de dependencias robusta y potente y son muy adecuados para construir proyectos basados en Java, debido a que permite sin mayor esfuerzo el consumo de componentes o módulos desarrollados por terceros o definir su propia bibliotecas de dependencias por proyecto permitiendo construir aplicaciones escalables [73]. Brinda muchos beneficios, el más importante es su flexibilidad. Los desarrolladores pueden aprovechar toda la potencia de un lenguaje de programación para definir y personalizar potencialmente el proyecto construir y ejecuta un demonio en segundo plano, que está siendo reutilizado después de la primera compilación, para acelerar las ejecuciones posteriores de la construcción. También realiza un seguimiento de las entradas y salidas de la compilación, si se han realizado cambios desde la última ejecución de la compilación. Esto permite que el sistema almacene en caché los pasos y reduzca el tiempo de desarrollo. Sin embargo, dependiendo de la complejidad del proyecto y sus dependencias utilizadas esta optimización podría ni siquiera ser necesario [73]. Gradle incluye un plugin “war” para la construcción de aplicaciones web Java, y la comunidad proporciona un excelente plugin llamado gretty para probar y desplegar aplicaciones web en Jetty o Tomcat [74].

2.11.3. Maven

Es una herramienta para la gestión y configuración de proyectos Java EE, sus funciones son similares a la herramienta Apache ANT, pero poseen conceptos diferentes [75]. Maven se encuentra desarrollado por Apache Software Foundation, donde se utiliza la construcción de Modelos de Objetos de Proyectos (POM) para detallar si un proyecto se está desarrollando con dependencias externas a sus módulos y el orden de compilación [73] . Al utilizar Maven el proyecto se define de mejor manera donde se aplica la lógica transversal de complementos personalizados o compartidos [76]. Unas de las grandes ventajas de Maven es que este descarga dinámicamente todos aquellas librerías que se necesitan para el desarrollo del proyecto Java EE del Repositorio Centran Maven, sin olvidarse de la estructura basada en complementos que este posee [75]. Antes de la aparición de Maven existía un responsable del proyecto dedicado a administrar la compilación de este, por ello en 2001 diferentes desarrolladores comenzaron a aplicar nuevos enfoques para la construcción de proyectos donde existían mil preguntas y pocas respuestas denominándose así a esta era antes de Maven como la del Ingeniero de Construcción. En la actualizad Maven ofrece un diversos beneficios desde la administración en el ciclo de vida del proyecto y sus dependencias hasta la lógica de compilación [76].

2.11.4. JSP/HTML/CSS

HTML (HyperText Markup Language) que en español significa Lenguaje de Marcas de Hipertexto es considerado un lenguaje de marcado principal de la World Wide Web que originalmente se diseñó para detallar documentos científicos semánticamente, sin embargo, al paso de los años se ha podido adaptar para detallar y describir otros tipos de documentos como por ejemplo aplicaciones [77]. HTML se encuentra en su versión 5.2 que provee diferentes características como estructura, funcionalidad, adaptabilidad y estilo; al hablar de este último, CSS (Cascading Stylesheets) que en español significa Hojas de Estilos en Cascada es el mayor representante que se encarga de dar estilos visuales como tamaño, color, bordes, sombras, etc. Cabe destacar que estas tecnologías son altamente dependiente entre sí [78].

Java Server Page o mejor conocida como JSP es una página HTML que brinda la ventaja de poder incrustar código Java facilitando así la creación de páginas web dinámicas

basadas no solo en HTML sino también en XML y otros tipos de documentos, brindando así un mayor manejo de los datos [79]. Una de las principales ventajas que ofrece JSP ante otros lenguajes es que con él se pueden crear aplicaciones web que manejen una lógica de negocios y acceso a datos de manera fácil y sencilla, de tal manera que permita la separación de los niveles dentro de la aplicación. Otra ventaja es la posibilidad de heredar la portabilidad del lenguaje Java [79]. Al tocar el tema de JSP es casi indispensable no hablar de la tecnología Servlet que es equivalente al rendimiento de una página JSP, ya que compilan de la misma manera, haciendo que JSP tenga un mejor desempeño ante otras tecnologías web. Los Servlet y los JSP son diferentes métodos de creación de páginas web tanto como PHP, ASP, entre otros [80]. Un Servlet es una clase Java que corre en el servidor, tiene parentesco con la clase HttpServlet. Su historia comienza con la creación de los programas applet que eran escritos en lenguaje Java y corrían en el navegador, pero fueron descontinuados desde que Microsoft Explorer suspendió su mantenimiento y por ello llegaron los servlet que los sustituyeron. Un Servlet da servicio a aquellas peticiones que demanda el navegador web, las procesa y devuelve una respuesta [81].

3. Arquitecturas

En este capítulo vamos a realizar una descripción de las arquitecturas de las tecnologías utilizadas para el estudio:

3.1. Arquitectura MEAN

Al desarrollar una aplicación web la elección de la mejor arquitectura y que para el usuario sea una experiencia fácil, agradable y funcional es trascendental. MEAN trabaja con una arquitectura MVC model-view-controller, que comprende tres capas esenciales: datos, lógica y vista. Una arquitectura MVC funciona así [43]:

1. Una solicitud entra en la aplicación.
2. La petición se enruta a un controlador.
3. Si es necesario, el regulador realiza una petición al modelo.
4. El modelo responde al controlador.
5. El controlador envía una respuesta a una vista.
6. La vista envía una respuesta al solicitante original.

En la arquitectura MVC, la lógica, los datos y la presentación se separan tres tipos de objetos, cada uno se encarga de sus propias tareas. La vista maneja la parte visual, que trata de la interacción con el cliente. El controlador responde a las peticiones del sistema y del cliente, haciendo que el modelo y la vista cambien apropiadamente. El modelo maneja la información, respondiendo a las demandas de datos o cambiando su estado de acuerdo a las instrucciones enviadas desde el controlador [49].

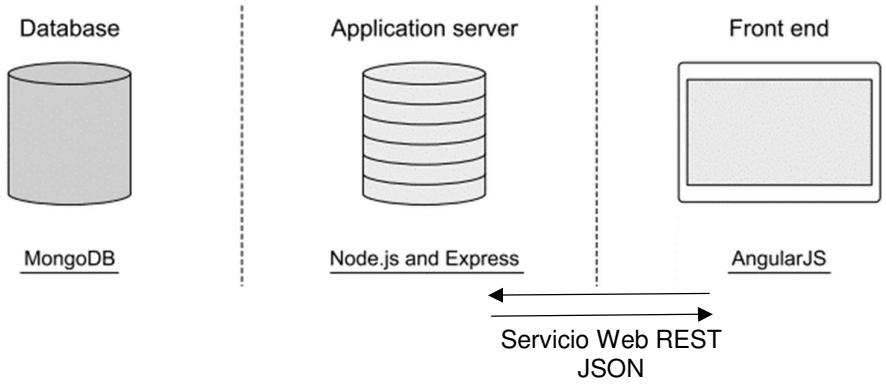

Figura 6. Arquitectura MEAN [43].

La figura 6 muestra la arquitectura de aplicación típica de una pila MEAN. AngularJS como Front-End con un MVC, que se comunica con el servidor Node a través de Express. Por cada solicitud de datos del Back-End, Node envía la solicitud a través del controlador nativo MongoDB o Mongoose. La respuesta del servidor es enviada al cliente a desde Express. El modelo de datos que se utilizado es JSON y siendo RESTful la estructura de servicio para conectar las rutas desde Express, permitiendo la transferencia de petición y respuesta de ida y vuelta entre el cliente y el servidor [49] .

3.2. Arquitectura Java EE

La arquitectura Java EE para la aplicación utiliza el modelo MVC en tres capas con el Front-End. En la arquitectura Java EE se genera un servicio web REST, para recibir y enviar las peticiones. En la figura 7 podemos observar la estructura de la tecnología Java EE utilizada.

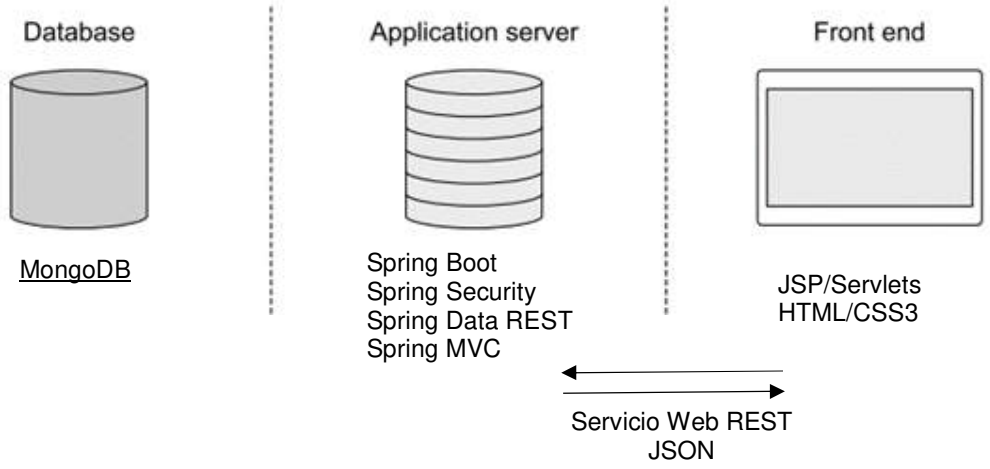

Figura 7. Arquitectura Java EE.

En el uso de Java se utiliza Spring por las ventajas que proporciona. Los módulos que forman el stack Spring como: IOC, AOP, MVC, Data Access & Transaction Management, Authentication & Authorization, Boot (para servicios web REST). Los datos del Sistema se guardan en gestor de base de datos No Relacional MongoDB. Al utilizar MongoDB simplifica y aprovecha Spring Data MongoDB de Spring que permite un acceso y configuración rápida los datos. Además, Spring Security se utiliza para la parte de Seguridades complementando a Spring MVC para gestión de usuarios y sesiones.

En la parte de la vista utilizamos JSP/Servlets junto HTML5/CSS3 junto con Spring. El formato de envío y recepción de peticiones por parte de los servicios web REST es JSON.

3.3. Stack MEAN y stack Java EE: Características, ventajas y desventajas

En este capítulo se muestran las características, ventajas y desventajas de cada una de las stacks. MEAN gana popularidad debido a las características de sus componentes: Node.js, MongoDB, AngularJS, junto con Express y todo utilizando JavaScript que es una de sus mayores ventajas. Aunque podemos afirmar que MEAN no se recomienda en sistemas pesados de procesamiento computacional, debido a que el componente Node.js es un entorno de un solo hilo y esto sería un inconveniente fuerte de esta tecnología. La madurez de la tecnología Java EE es la opción más correcta para este tipo aplicaciones pesadas de procesamiento computacional. Pero para una aplicación de una página (SPA), que requiere una alta interacción con el usuario, visualización y alta escalabilidad, la pila MEAN es la mejor opción. Cabe recalcar que en el ámbito empresarial Java EE es la tecnología más popular para el desarrollo de aplicaciones web por la seguridad que brinda esta tecnología. La elección de la stack depende principalmente de algunas características, que las vamos a analizar para conformar un conjunto de parámetros que permiten realizar la comparativa entre estas dos stacks, en definitiva, se trata de buscar un marco de evaluación de la arquitectura con los criterios elegidos. A continuación, se detalla los parámetros elegido y al final encontramos un resumen de las stacks a partir de la elección de parámetros efectuada.

3.3.1. Capa de servidor simplificada

La stack Java EE en su servidor simplemente empaqueta el artefacto en jar que viene ya con un servidor de aplicaciones embebido que puede ser Tomcat, Jetty o Undertow. En el estudio realizado se utiliza Tomcat. Esta tecnología es parte del Core de Spring que ayuda a configurar automáticamente algunos aspectos de la aplicación, lo que hace que nos enfoquemos en el desarrollo y por ende realizar más rápido la aplicación. También la forma de trabajar de esta manera es que se plantea que cada módulo sea lo más independiente posible para poder integrarse de una manera simple en cualquier entorno y que puede desplegarse sin necesidad de tener un contenedor de servlets externo, logrando una arquitectura orientada a microservicios. La stack MEAN simplifica la capa de servidor mediante el uso de Node, que proporciona el mejor entorno para ejecutar el

servidor. Node.js maneja todas las tareas, desde las solicitudes de ruta de aplicaciones hasta la reescritura de URLs o la construcción de mapeos con el uso de código JavaScript. Node.js evita la necesidad de tener un archivo de configuración para cada tarea. Dado que Node.js proporciona todo en una sola capa, simplifica la capa del servidor al reducir la cadena de errores. Java EE tiene un servidor web embebido, mientras que en MEAN, el servidor web puede ser construido localmente usando Node.js.

3.3.2. Código isomórfico

Como señala [82] isomórfico significa que cualquier línea de código (con algunas excepciones) de una aplicación web puede ser ejecutada en el cliente así como en el servidor. Por lo que la reutilización del código sería una realidad. Estas stack de tecnologías deberían permitir: Enrutamiento, obtención y persistencia de datos, renderizado de vistas, armado y empaquetado. JavaScript es el único lenguaje utilizado en MEAN, desde scripts del lado del servidor para Node.js y Express hasta scripts del lado del cliente usando AngularJS. Todo el proceso envió y recepción de consultas en formato JSON. El único lenguaje utilizado es JavaScript, esto hace que no se requiera diferentes expertos para el desarrollo del back-end y front-end. Igualmente, en la tecnología Java como lenguaje de programación único durante la configuración, en el desarrollo del back-end y en el front-end no se requiere de diferentes expertos. Aunque en este último podemos utilizar cualquier otra tecnología, framework o lenguaje para realizar el front-end.

3.3.3. Escalabilidad

Una aplicación es escalable, cuando pueda expandirse a medida que crece su uso y también si puede funcionar mejor después de la expansión. Un sistema escalable, debe poder aumentar el volumen de su base de datos, incorporar módulos, aceptar un mayor número de peticiones, sin dejar de funcionar y ni disminuir la calidad de su servicio [83]. En el stack Java EE, utilizamos Spring boot debido a que es muy adecuado para el desarrollo de aplicaciones web, ya que crea un servidor Tomcat embebido, utilizando spring- boot-starter-web [84]. Tomcat al ejecutar una petición esta es manejada a través de una distribución multihilos. Cuando existen varias peticiones se crean varios hilos ya que cada petición es un nuevo hilo. Depende de la configuración del servidor para el procesamiento de las peticiones, si estas se exceden, existe un tiempo de espera hasta que todas las peticiones sean procesadas [85]. Es realmente sencillo debido a que requiere de

configuraciones mínimas para tenerlo listo para el proyecto a realizar. Tomcat embebido permite incluir una configuración general de la aplicación y dejar en archivos externos parametrizados lo que se requiera, que proporciona un entorno estable para el desarrollo y para su uso en la producción [68]. En el caso de MEAN, Node utiliza un enfoque basado en eventos, asíncrono, es decir, sin bloqueo, por lo que los eventos se ejecutan libremente. En esencia significa que un servidor Node.js nunca espera a que una aplicación devuelva datos. El servidor se mueve a la siguiente, luego de llamarla y un mecanismo de notificación de eventos de Node.js ayuda al servidor a obtener una respuesta para continuar con la secuencia principal de la aplicación. Una vez que el evento esté listo, el programa volverá a él con una función de llamada de retorno, ejecutará el código y luego regresará al flujo principal del programa [86].

3.3.4. Arquitectura sin bloqueo

En MEAN, una característica que hace a Node.js más potente es su arquitectura sin bloqueo. Node usa un modelo de un solo hilo con bucle de eventos. El mecanismo de eventos ayuda al servidor a responder de forma no bloqueada y hace que el servidor sea altamente escalable, a diferencia de los servidores tradicionales, que crean hilos de discusión limitados para gestionar las solicitudes. Node utiliza un único programa enhebrado y el mismo programa puede proporcionar servicio a un número mucho mayor de peticiones que los servidores tradicionales que le proporcionan un rendimiento y una escalabilidad muy elevadas [86]. En el stack Java EE, al utilizar Apache Tomcat embedido el bloqueo se resuelve a través de múltiples hilos de ejecución mientras que en MEAN, Node maneja el uso de bucles de eventos sin bloqueo en un solo hilo.

3.3.5. Tiempo de desarrollo

En MEAN, JavaScript se utiliza en cada componente de la stack, por lo que es el único lenguaje de programación utilizado. Esto reduce el esfuerzo y el tiempo de desarrollo. En el caso del stack Java EE, igualmente en todo el proceso, es utilizado este el lenguaje de programación, salvo en el Front-End en el que se adicionalmente se agrega HTML/CSS, pero podemos asegurar que la utilización de un marco de trabajo maduro como lo es eclipse, ayuda de gran manera el desarrollo de una aplicación debido a que permite a los desarrolladores tener a su disposición los complementos necesarios para integrarlos y construcción del software mejorando el tiempo de desarrollo de la aplicación.

3.3.6. Transformación de datos y extensibilidad

Los datos manejados para el intercambio de información es JSON, que es el formato estándar más utilizado para el intercambio de datos junto como XML [33]. Ambas tecnologías tanto en JavaScript como Java la soportan y permiten realizar intercambio de datos. En cuanto a extensibilidad MongoDB proporciona flexibilidad para cambiar la estructura de la base de datos. En el caso de estudio utilizamos la misma base datos. En el stack Java EE con una configuración muy simple soporta cualquier motor de base de datos.

3.3.7. Rendimiento

Node.js en varios estudios ofrece un mejor rendimiento debido a su arquitectura basada en eventos. Pero igualmente existen estudios en que Java EE, brinda un excelente rendimiento. Lo cual, vamos a comprobar en las siguientes secciones.

3.3.8. Resumen de características

La siguiente tabla muestra un resumen de las características de cada stack.

Característica	MEAN	Java EE
Capa de servidor simplificada	Construcción de servidor localmente	Empaquetamiento de artefacto en un servidor embebido
Código isomórfico	Código puede ser ejecutado en el front-end como el back-end	Código puede ser ejecutado en el front-end como el back-end
Escalabilidad	La aplicación puede expandirse los eventos se ejecutan libremente de manera asíncrona	La aplicación se expande a través del servidor embebido con una distribución multihilos.
Arquitectura sin bloqueo	Maneja el uso de bucles de eventos sin bloqueo en un solo hilo.	El bloqueo se resuelve a través de múltiples hilos de ejecución
Tiempo de desarrollo	Al utilizar el mismo lenguaje JavaScript se reduce el esfuerzo y tiempo de desarrollo	Al utilizar el mismo lenguaje Java se reduce el esfuerzo y tiempo de desarrollo. Existen varios marcos de trabajo para mejorar el tiempo de desarrollo
Transformación de datos y extensibilidad	Los datos manejados para el intercambio de información es JSON.	Los datos manejados para el intercambio de información por estándar son XML, pero se puede trabajar con JSON.

Tabla 1. Resumen de características de las stacks.

Como conclusión de este capítulo, podemos decir que ambas stacks tanto MEAN y Java EE, cumplen con las características evaluadas manejándolas de distinta forma a través de distintas tecnologías. La stack MEAN resuelve de mejor manera la mayoría de las características evaluadas antes que la stack Java EE.

4. Configuración, ambientes y comentarios del desarrollo de una aplicación sobre los dos stacks

Las dos stacks tienen distintas configuraciones y ambientes para el desarrollo de una aplicación web e igualmente para desarrollar un servicio REST. En este capítulo nos enfocamos en la descripción de las configuraciones iniciales para el ambiente, además la instalación e integración de los componentes para las dos stack MEAN y Java EE, que nos permita realizar la implementación una aplicación web basada en servicios REST. El caso de estudio a realizar con las dos tecnologías de una aplicación web para manejo de la hoja de vida y gestión docente para el departamento de talento humano de la PUCE, Sede Esmeraldas.

4.1. Configuración y comentarios de desarrollo de una aplicación sobre el stack MEAN

Para la creación de una aplicación web utilizando la stack MEAN, es necesario únicamente un editor de texto, se escogió Atom que es un editor de código de fuente abierta para macOS, Linux, y Windows con soporte para plug-ins escrito en Node.js, Incrustando Git Control, desarrollado por GitHub. También puede ser utilizado como un entorno de desarrollo integrado (IDE) [87]. Se utilizó en su versión más reciente en su versión 1.27.2 para 64bits el cual ofrece características más potentes descritas en su página oficial (https://atom.io/) [88]. Una vez instalado el IDE podremos utilizar todos sus paquetes, librerías y herramientas. Para proceder a instalar los componentes de MEAN es sencillo y puede instalar en cualquier plataforma sea Linux, Windows, MAC Os X. Los componentes que tenemos que instalar son Node, MongoDB, AngularJS y Express. Para realizar la instalación seguimos los siguientes pasos:

- El primer paso es crear un directorio donde se alojará el proyecto. Utilizando el comando:

$ npm init

- Luego Index.js es el primer archivo donde se ejecuta el proyecto a partir de aquí se crea el archivo package.json en la carpeta. Este el punto de partida para cargar módulos o funciones. Este es un fichero básico para un proyecto en Node.JS. Este trabaja con Express así que lo instalamos con el comando:

$ npm install express -save

- Igualmente, instalamos la librería Bodyparse que nos permite manejar y recibir cualquier parámetro y lo convierte en objecto Javascript.

$ npm install body-parser –-save

- Para evitar el trabajo repetitivo de cuando hacemos algún cambio al código tenemos que ejecutar el script para observar los cambio. Para evitar ese trabajo repetitivo instalamos un demonio Nodemon que se encargara de comprobar los cambios en el proyecto y automáticamente lo actualiza y laza el servidor e igualmente no envía un mensaje si existe algún error. El comando para instalar es el siguiente:

$ npm i -D nodemon

- Los componentes y librerías instaladas se pueden revisar en package.json. Las podemos observar en la siguiente figura.

```
{
  "name": "api-restful",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jaime Sayago Heredia",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.18.2",
    "express": "^4.16.2",

  },
  "devDependencies": {
    "nodemon": "^1.17.1"
  }
}
```

Figura 8. Componentes y librerías instaladas para MEAN.

Vamos a comprobar si el servidor Node funciona junto con Express. Para incluir Express solamente se debe agregar las siguientes líneas:

```
1  //todas las configuraciones
2  'use strict'
3
4  var express = require ('express');
5  var app = express();
```

Figura 9. Servidor Node.

En la siguiente figura se muestra una parte del código necesario para probar que el servidor funciona

```
1  'use strict'
2  var express=require('express');
3  var app=express();
4  var bodyParser= require('body-parser');
5
6  var port=process.env.PORT || 3678;
7  app.use(bodyParser.urlencoded({extended:false}));
8  app.use(bodyParser.json());
```

Figura 10. Código para prueba de servidor.

Al ejecutar el código se muestra levantado el servidor, como se puede observar en la siguiente figura:

```
[nodemon] 1.12.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node index1.js`
API REST funcionando en http://localhost:3678
```

Figura 11. Servidor web en prueba.

Se encuentra correctamente configurado el servidor Node.js. Al observar el mensaje de consola del servidor al ejecutarlo el demonio, en este caso el mensaje es "API REST funcionando en http://localhost:3678". Además, en el navegador podemos ingresar a http://localhost:3678, y observar mensaje en el navegador.

4.1.1. Configuración y comentarios del back-end

Para el desarrollo de servicio web RESTful, en el capítulo dos se describió a fondo la arquitectura REST de los métodos HTTP. El modelo MVC será utilizado para la estructura del proyecto en el que creamos controllers, models, routes y luego configurarlas. Además de utilizar los distintos métodos HTTP como el Put o Delete. El proceso se realizó de la siguiente manera: En el controlador agregamos todos los métodos de crear, guardar, eliminar y listar los datos del proyecto. Como se puede apreciar en la siguiente figura un parte del código.

```
var express = require('express');
var Usuario = require('../Modelos/model');
var multer = require('multer');
var router = express.Router();

function getUsuarios(req, res) {
    Usuario.find({}).sort('-_id').exec((err, usuarios) => {
        if (err) {
            res.status(500).send({message: 'Error al devolver marcadores'});
        } else {
            if (!usuarios) {
                res.status(404).send({message: 'No hay marcadores'}); //devol
            } else {
                res.json(usuarios); //devolver un mensaje json
```

Figura 12. Método obtener usuario.

```
function deleteUsuario(req, res) {
    var usuarioId = req.params.id;
    Usuario.findById(usuarioId, function (err, usuario) {
        if (err) {
            res.status(500).send({message: 'Error al devolver el marcador'}); //devolver
        } else {
            if (!usuario) {
                res.status(404).send({message: 'No hay marcador'}); //devolver un mensaje
            } else
            {
                usuario.remove(err => {
                    if (err)
                    {
                        res.status(500).send({message: 'El marcado no se ha eliminado'});
                    } else {
                        res.status(200).send({message: 'El marcado se ha eliminado'}); //
                    }
                });
            }
        }
    });
}

function getUsuario(req, res) {
    var usuarioId = req.params.id;
    Usuario.findById(usuarioId, function (err, usuario) {
        if (err) {
            res.status(500).send({message: 'Error al devolver el marcador'}); //devolver
        } else {
            if (!usuario) {
                res.status(404).send({message: 'No hay marcador'}); //devolver un mensaje
            } else {
                res.json(usuario); //devolver un mensaje json
            }
```

Figura 13. Métodos CRUD.

Mientras tanto que los modelos son un esquema del tipo de documento que se va a guardar en la base de datos mongoDB. Como lo podemos apreciar en la siguiente figura parte de código de un modelo.

```
var mongoose = require('mongoose');
var CapacitacionImpartida = new mongoose.Schema({
    usuario:String,
    tipo_evento: String,
    institucion : String,
    n_horas : String,
    fecha_desde :String,
    fecha_hasta :String
})

module.exports = mongoose.model('CapacitacionImpartida', CapacitacionImpartida);
```

Figura 14. Código de un modelo.

Al existir múltiples rutas en el que el servidor debe saber a qué página dirigir las peticiones del cliente. Esto se lo hace a través de las rutas. En la siguiente figura se muestra una parte del código para realizar las rutas de las operaciones del controlador.

```
'use strict'
var express= require('express');
var UsuarioController =require('../Controladores/usuario');
var api=express.Router();

api.get('/usuario/:id',UsuarioController.getUsuario);
api.get('/usuarios',UsuarioController.getUsuarios);
api.post('/usuario',UsuarioController.saveUsuario);
api.put('/usuario/:id',UsuarioController.updateUsuario);
api.delete('/usuario/:id',UsuarioController.deleteUsuario);

module.exports=api;
```

Figura 15. Código de rutas de operaciones.

Por último, la base de datos de datos MongoDB, en que es necesario levantar el servidor a través del comando:

```
$ mongod --port 27017 -dbpath c:\data\db
```

Figura 16. Comando inicio base de datos.

Al ejecutar el comando, se muestra el mensaje “waiting for connections on port 27017”, que nos dice que estamos listo para trabajar con MongoDB. Luego necesitamos ejecutar el siguiente comando para tener acceso a la consola administrativa de MongoDB

```
$ mongo
MongoDB shell version v3.4.9
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.9
```

Figura 17. Comando para ejecución base de datos.

Esto nos permite tener acceso de las distintas bases de datos y colecciones de MongoDB. Podemos comprobar, en que base de datos nos encontramos con el comando

```
db
test
```

Figura 18. Comando ubicación de base de datos.

Por defecto nos muestra la base de datos “test”. Tenemos varios comandos para el acceso de las distintas bases de datos y sus respectivas colecciones. En nuestro caso vamos a utilizar la base de datos “Proyecto”. Para poder ver una colección se requiere del comando “db.<collection_name>.find().pretty()”. Como lo podemos observar en la siguiente figura ingresamos a la base de datos, su respectiva colección y mostramos los datos guardados en ella.

```
use proyecto
switched to db proyecto
db.usuarios.find().pretty()
{
        "_id" : ObjectId("5ab96146d759c14095e5d432"),
        "_class" : "com.example.model.User",
        "nombres" : "jaime",
        "apellidos" : "sayago heredia",
        "cedula" : "0301435012",
        "email" : "jaimepsayago@gmail.com",
        "fecha_nacimiento" : "1982-07-06",
        "estado_civil" : "Soltero/a",
        "tipo_sangre" : "A+",
        "nacionalidad" : "ecuatoriano",
        "lugar_nacimiento" : "azogues",
        "telefono" : "0984787662",
        "convencional" : "072248435",
        "direccion" : "calle 3 de noviembre y oriente"
```

Figura 19. Comando obtener colección.

Para trabajar con MongoDB junto con Node.js, se requiere de un módulo Mongoose. Este nos permite realizar las operaciones de la base datos. Además, nos presenta una capa de abstracción mucho más suave para MongoDB, lo que conlleva una mayor facilidad y sencillez al momento de trabajar con métodos y objetos. La instalamos con el siguiente comando:

```
$ npm install mongoose --save
```

Figura 20. Comando instalación mongoose.

Este comando instala todos los paquetes necesarios para Mongoose. Para utilizar la librería en el servidor únicamente requiere de las siguientes líneas de código:

```
'use strict'
var mongoose = require('mongoose');
var app = require('./app');
var port = process.env.PORT || 3000;
mongoose.connect('mongodb://localhost:27017/usuarios'
```

Figura 21. Configuración utilización de módulo mongoose.

Estos comandos nos dan acceso a la base de datos MongoDB y a sus respectivas colecciones.

4.1.2. Configuración y comentarios del front-end

El componente que vamos a instalar es AngularJS. En esta sección vamos realizar su configuración inicial y describir el proceso de la creación del front-end. El primer paso crear un directorio donde vamos a instalar el proyecto, en nuestro caso lo vamos a llamar "front-end". El siguiente paso es realizar la instalación de AngularJS. Se debe realizar la instalación de varios ficheros, componentes y librerías para tenerlo configurado y generar una aplicación. Los archivos mínimos requeridos son:

- tsconfig.json
- typings.json
- package.json
- systemjs.config.js

Cada uno de estos archivos debe contener las respectivas configuraciones y dependencias. Es necesario instalar algunos componentes a través de NPM, como son typings, utilizamos el siguiente comando:

$ npm run typings install

Luego se crea una carpeta "app", que es la ubicación de los archivos de configuración y dependencias para la aplicación web. Creamos los siguientes archivos

- app.component.ts, contendrá el template y la lógica de la plantilla hacia la aplicación
- app.module.ts, contendrá datos y funcionalidad como son los componentes, servicios y organización.
- index.html, sirve como página principal para inyección de las vistas. Como podemos observar en la siguiente figura:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Sistema de docentes</title>
  <base href="/">
<link rel='shortcut icon' type='imagenes/x-icon' href='/src/favicon.ico' />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <script type="text/javascript" src="/src/js/jquery-1.12.2.min.js"></script>
   <script type="text/javascript" src="/src/js/bootstrap.min.js"></script>
    <link rel="stylesheet" href="/src/css/styles.css"/>


</head>
<body style="font-size: 12px;">
  <app-root>Cargando...</app-root>
</body>
</html>
```

Figura 22. Página principal para inyección de vistas.

Luego de la configuración realizada esta lista la estructura para proceder a la realización de la aplicación web. Es necesario crear los siguientes componentes:

- Modelos, que son el esquema de los distintos atributos en la base datos.
- Servicios, que contiene la ruta respectiva y la codificación necesaria para que se consuma el servicio web REST. Como se puede observar en la figura 23 parte del código.

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import 'rxjs/add/operator/map' ;
@Injectable()
export class UsuarioService {

  constructor( private http:Http) { }
  //Usuarios
  getUsuarios(){
    return this.http.get("http://localhost:3000/api/usuarios")
        .map(res => res.json());
  }
   addUsuario(info){
    return this.http.post("http://localhost:3000/api/usuario",info)
        .map(res => res.json());
  }
  getUsuario(id){
    return this.http.get("http://localhost:3000/api/usuario/"+id)
        .map(res => res.json());
  }
  deleteUsuario(id){
    return this.http.delete("http://localhost:3000/api/usuario/"+id)
        .map(res => res.json());
  }
```

Figura 23. Código componentes para consumo de servicio web REST.

Componentes, es necesario crear los distintos componentes de la aplicación web que consuma los servicios web y realice las operaciones CRUD para ser agregados en la vista que se presenta al usuario final. En la figura 24, podemos observar parte del código para generar un componente.

```
import { Component, OnInit } from '@angular/core';

import { UsuarioService } from '../servicios/usuarioservice'
import { Usuarios } from '../modelos/usuarios'

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {

  constructor(
    public usuarioService:UsuarioService
  ) { }

  ngOnInit() {
    this.getUsuarios();
  }
  usuarios:Usuarios;
  getUsuarios(){
    this.usuarioService.getUsuarios()
        .subscribe(usuarios=>{
          this.usuarios = usuarios;
        })
  }
```

Figura 24. Código para generar un componente.

4.2. Configuración y comentarios de desarrollo de una aplicación sobre la stack Java EE.

Para la creación de una aplicación web utilizando el stack Java EE, fue necesario la utilización del IDE Eclipse que es un software plataforma compuesto con diversas herramientas de programación con la característica principal de ser código abierto multiplataforma. Esta plataforma es usada para el desarrollo de Entornos de Desarrollo Integrado (IDE) proporcionando una serie de ayudas que permiten escribir código Java de manera rápida y sencilla sin necesidad de buscar documentación [89][90]. Se utilizo en su versión más reciente denominada Oxygen de 64bits el cual ofrece características más potentes descritas en su página oficial (www.eclipse.org/oxygen). Una vez instalado el IDE podremos utilizar todos sus componentes y herramientas.

4.2.1. Configuración y comentarios del back-end

Para el desarrollo del servidor que proveerá del servicio en formato JSON fue necesario la creación de un proyecto Gradle que ofrece beneficios antes expuestos en un punto anterior. Una vez procesado la petición al IDE se procederá a instanciar las dependencias en el archivo 'build.gradle' que serán de suma importancia para el desarrollo correcto del Servidor.

```
dependencies {
    api 'org.apache.commons:commons-math3:3.6.1'

    implementation 'com.google.guava:guava:21.0'

    testImplementation 'junit:junit:4.12'

    compile 'org.springframework.boot:spring-boot-starter-web:1.5.6.RELEASE'

    compile 'org.springframework.data:spring-data-mongodb:1.10.6.RELEASE'

    compile 'org.glassfish.jersey.containers:jersey-container-servlet:2.25.1'

    compile 'javax.servlet:javax.servlet-api:4.0.0-b07'

    compile 'org.glassfish.jersey.media:jersey-media-json-jackson:2.25.1'
}
```

Figura 25. Código de dependencias.

Dependencias como Spring Framework y Servlet se pueden observar en la figura 25 donde se inicia con la palaba reservada 'compile' y luego el nombre y versión de esta para su utilización. La segunda parte importantes para el desarrollo de este Servidor es la

configuración de la conexión a la Base de Datos NoSQL MongoDB; para ello es necesario ir al recurso 'application.properties' que se encuentra en el paquete/carpeta 'src/java/ resources':

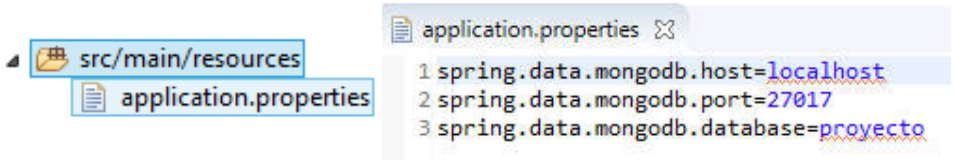

Figura 26. Configuración base de datos.

Dentro de este archivo se utiliza la clase 'Spring.data.mongodb' para declarar el host, el puerto y el nombre de la base de datos que se encuentra en MongoDB. Utilizando el administrador de archivos de Eclipse, se crean cuatro diferentes packages llamados 'models' que contendrán todos los modelos, 'controllers' donde se programará el servicio web, 'repositories' que son un conjunto de interfaces de comunicación entre los modelos y los controllers, quedando estructurada de la siguiente manera como se muestra en la figura 27:

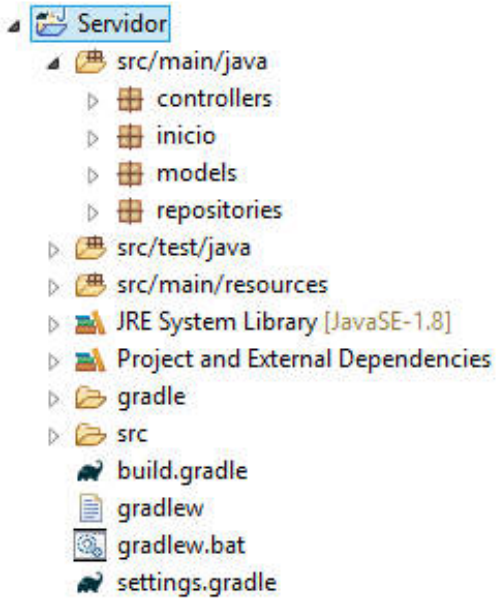

Figura 27. Estructura de la aplicación.

Modelos: Los modelos constan de un ID y la información que se quiere guardar en la base de datos, dentro de esta clase Java se necesitará importar dos clases correspondientes a Spring:

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
```

Figura 28. Código importación clases.

Que servirán para el correcto manejo de la información en formato JSON; en la siguiente Figura 29 se muestra la estructura de la clase 'User.java' la cual tiene similitud con las demás clases.

```
@Document(collection = "usuarios")
public class User {

    @Id
    private String id;
    private String nombres;
    private String apellidos;
    private String cedula;
    private String email;
    private String fecha_nacimiento;
    private String foto;
    private String estado_civil;

    private String tipo_sangre;
    private String nacionalidad;
    private String lugar_nacimiento;;
    private String telefono;
    private String convencional;
    private String direccion;

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getNombres() {
        return nombres;
    }
    public void setNombres(String nombres) {
        this.nombres = nombres;
    }
```

Figura 29. Código clase Usuario.

Repositories: Dentro de las interfaces repositories se pueden colocar métodos/funciones adicionales que deseemos colocar para el manejo de los datos; de la misma manera que en models se necesita importar una clase correspondiente a Spring.

```
import org.springframework.data.mongodb.repository.MongoRepository;

import models.User;

public interface UserRepository extends MongoRepository<User, String> {

    public User findOneByNombres(String nombres);

}
```

Figura 30. Código de importación de clase.

Controllers: Las clases controllers pueden ser consideradas una parte importante para la conclusión del proyecto Servidor, ya que es aquí donde se realizan las funciones de GET, POST, PUT y DELETE a través de rutas. Dentro de ellas se requiere importar diferentes clases como se muestra en la figura 31:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

Figura 31. Código importación varias clases.

Además, de importar las clases del modelo y repositories para el correcto funcionamiento del proyecto, para que en conjunto se realice un Mapeo a las peticiones recibidas y así determinar qué acción realizar.

```
import models.User;
import repositories.UserRepository;

@RestController
@RequestMapping("/api")
@CrossOrigin(origins="http://localhost:4200", allowedHeaders="*")
public class UserController {

    @Autowired
    UserRepository userRepository;
```

Figura 32. Código mapeo de peticiones.

Cuando se realiza una petición de POST se ejecuta una función parecida a la presentada en la siguiente figura:

```
//crear usuario
@PostMapping("/user")
public void createUser(@RequestBody User user) {
    userRepository.save(user);
}
```

Figura 33. Código petición POST.

De la misma manera cuando se realiza una petición GET por Id:

```
//obtener un usuario
@GetMapping("/user/{id}")
public User getUser(@PathVariable String id) {
    return userRepository.findOne(id);
}
```

Figura 34. Código petición GET por id.

Cuando se realiza una petición GET ALL

```
//obtener varios usuarios
    @GetMapping("/users")
    public List<User> getUsers() {
        return userRepository.findAll();
    }
```

Figura 35. Código petición GET ALL.

Cuando se realiza una petición PUT:

```
@PutMapping("/user")
public void updateUser(@RequestBody User user) {
    userRepository.save(user);
}
```

Figura 36. Código petición PUT.

Y cuando se realiza una petición DELETE:

```
//borrar usuario
@DeleteMapping("/user/{id}")
public void deleteUser(@PathVariable String id) {
    userRepository.delete(id);
}
```

Figura 37. Código petición DELETE.

De esta manera el Servidor REST se encuentra correctamente configurado para recibir peticiones del Clientes, y totalmente desarrollado con la stack de Java EE. El servidor al ser ejecutado correrá bajo la dirección 'localhost:8080/api/'.

4.2.2. Configuración y comentarios del front-end

Para la creación del front-end REST se utiliza otra tecnología alternativa a Gradle como lo es Maven, de igual manera Maven se basa en dependencias para el correcto funcionamiento del proyecto, descargando las librerías necesarias a través de su repositorio central. Paralelo a esto es necesario la instalación de Glassfish el servidor del cliente REST, la descarga se puede realizar en el sitio oficial (javaee.github.io/glassfish/download). Para luego de esto, instalar glassfish tool en eclipse oxygen en la sección de 'Servers' como se muestra en la figura 38:

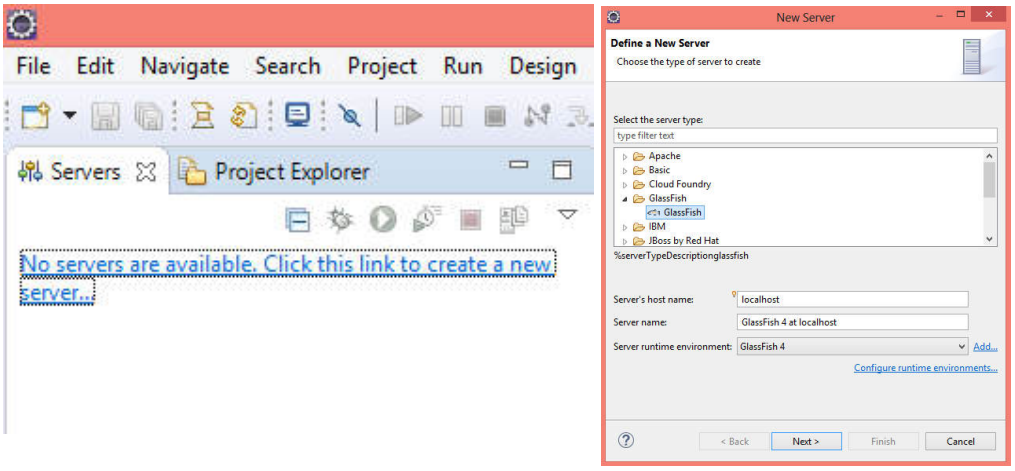

Figura 38. Instalación servidor.

De manera sencilla Eclipse ayuda a configurar el servidor glassfish para poder utilizarlo posteriormente con nuestro cliente REST. Cabe destacar que es necesario que el servidor glassfish corra por cualquier otro puerto que no se el 8080, ya que este se encuentra ocupado por el servidor REST. Para ello es necesario buscar el archivo 'domain.xml' e ir hasta donde se encuentra declarado por defecto el puerto 8080 y cambiarlo por otro que se encuentre disponible. Una vez configurado completamente el servidor glassfish, se deben colocar las dependencias que utilizará el proyecto en el archivo 'pom.xml', estas dependencias son: Spring Framework y sus subcomponentes, servlet, jackson, entre otras como se muestra en la siguiente figura 39:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    </dependency>
<!-- https://mvnrepository.com/artifact/org.springfra
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb</artifactId>
    <version>2.0.6.RELEASE</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.fasterxml
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.9.5</version>
</dependency>
```

Figura 39. Código dependencias.

Luego se procede a ir a las propiedades del proyecto y luego ir hasta 'Proyect Facets' para activar las funcionalidades de páginas web dinámicas, Java, JavaScript y REST web services.

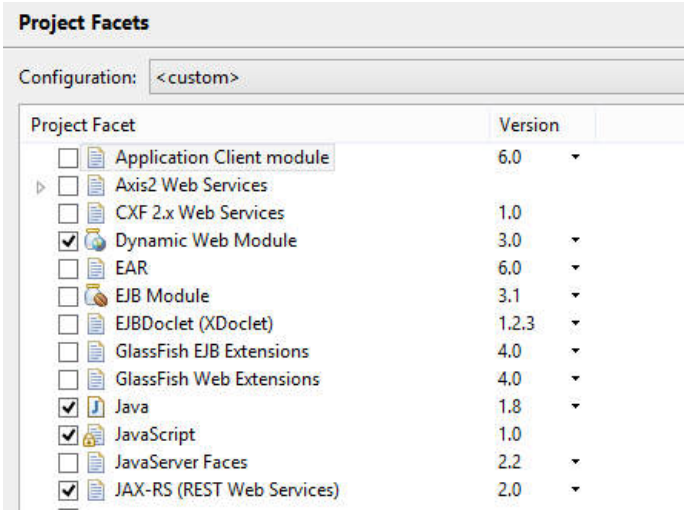

Figura 40. Activación funcionalidades.

Utilizando la arquitectura MVC se decidió separar el proyecto en tres partes, dos de ellas de tipo *.java (models - controllers) y la faltante *.jsp (views).

Models: Los modelos se encuentras declarados de la misma manera que en el servidor, para que la comunicación sea mejor procesada, adicional a esto se importó una clase '*com.fasterxml.jackson.annotation.JsonIgnoreProperties*' para que en caso exista un elemento faltante, este sea ignorado y la comunicación no se vea afectada.

```
@Document(collection = "usuarios")
@JsonIgnoreProperties(ignoreUnknown = true)
public class User {

    @Id
    private String id;
    private String nombres;
    private String apellidos;
    private String cedula;
    private String email;
    private String fecha_nacimiento;
    private String foto;
    private String estado_civil;

    private String tipo_sangre;
    private String nacionalidad;
    private String lugar_nacimiento;;
    private String telefono;
    private String convencional;
    private String direccion;
```

Figura 41. Modelo usuarios.

Controllers: Dentro de estas clases Java se declaran todos los métodos GET, POST, PUT y DELETE de la misma manera que en el servidor; de manera indispensable se utiliza la clase RestTemplate que es parte del paquete Spring para la comunicación con el servidor REST.

Para realizar la petición de GET y GET ALL:

```
//Obtener Todos los Usuarios
public User[] getUsers() {
    return restTemplate.getForObject(ruta+"users", User[].class);
}

//Obtener Usuario por ID
    public User getUser(String id) {
        return restTemplate.getForObject(ruta+"user/"+id, User.class);
    }
```

Figura 42. Código petición GET y GET ALL.

Para realizar la petición POST:

```
//Insertar Usuario
public void postUser(User user) {
    restTemplate.postForObject(ruta+"user", user, User.class);
}
```

Figura 43. Código petición POST.

Para realizar la petición PUT:

```
//Editar un Usuario
public void updateUser(User user) {
    restTemplate.put(ruta+"user", user);
}
```

Figura 44. Código petición PUT.

Para realizar la petición DELETE:

```
//Eliminar Usuario
public void deleteUser(String id) {
    restTemplate.delete(ruta+"user/"+id);
}
```

Figura 45. Código petición DELETE.

Para realizar las vistas es necesario la utilización de las JSP páginas en las que se pueden incrustar código Java, tema tratado en un punto anterior. Para la utilización de las clases java antes descritas, es necesario importarlas dentro de estos documentos *.jsp como se muestra en la siguiente figura 46:

```
<%@ page import="controllers.*" %>
<%@ page import="models.*" %>
```

Figura 46. Código JSP.

Utilizando HTML, CSS y JavaScript para el desarrollo y presentación del cliente REST. Una vez terminado su codificación se puede ejecutar en el servidor Glassfish y visualizar en la dirección 'localhost:8083/GestionDocenteProyecto/'. Cabe destacar que es necesario añadir las librerías utilizadas al Java Build Path dentro de las propiedades del proyecto.

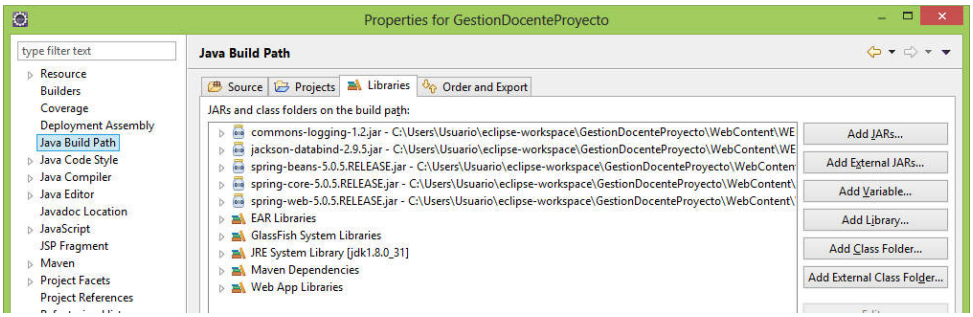

Figura 47. Librerías utilizadas para aplicación.

5. Caso de Estudio: Comparaciones, pruebas y análisis

En esta sección se procederá se presenta la aplicación web, las pruebas junto con la interpretación de resultados de comparativas y pruebas de la aplicación web de hoja de vida y gestión docente a través de servicios REST.

5.1. Aplicación objeto de estudio

Dependiendo de los requisitos de la aplicación, podemos elegir entre la stack MEAN o la stack Java EE para el desarrollo una aplicación web basada en servicios REST. Se propone que actualmente se prefiere la stack MEAN sobre la stack Java que en el Capítulo 2 se evidencia. Para probar y analizar los pros y contras se las dos stacks, se desarrolló la aplicación de la hoja de vida y gestión de docente para el departamento de talento humano de la Pontificia Universidad Católica del Ecuador, Sede Esmeraldas, que fue construido utilizando ambas pilas, seguido de algunas pruebas de evaluación comparativa. Algunos de los criterios como el desarrollo de aplicaciones en tiempo de respuesta, el rendimiento, la carga de datos son demostrados en las siguientes secciones. Las pruebas de la aplicación fue con la ayuda de tres herramientas de evaluación comparativa: Siege [91] , Apache Bench [92] y de tiempo de respuesta fue JMeter [93]. La aplicación está construida con la stack MEAN y la stack Java EE.

- JMeter

JMeter [93] es una herramienta de carga para llevar acabo simulaciones sobre cualquier recurso de Software. Inicialmente diseñada para pruebas de estrés en aplicaciones web, hoy en día, su arquitectura ha evolucionado no sólo para llevar a cabo pruebas en componentes habilitados en Internet (HTTP), sino además en Bases de Datos, programas en Perl, requisiciones FTP y prácticamente cualquier otro medio. Además, posee la capacidad de realizar desde una solicitud sencilla hasta secuencias de requisiciones que permiten diagnosticar el comportamiento de una aplicación en condiciones de producción. En este sentido, simula todas las funcionalidades de un Navegador ("Browser"), o de cualquier otro cliente, siendo capaz de manipular resultados en determinada requisición y reutilizarlos para ser empleados en una nueva secuencia [94].

Su instalación es sencilla únicamente es necesario descargarlo de su página oficial y luego crear un script o utilizando las pantallas de configuración de pruebas.

- Apache Bench

Apache Bench es una plataforma de línea de comandos. En el sistema operativo Windows, viene con la instalación predeterminada del servidor Apache. Esta herramienta realiza las pruebas de carga enviando el número de request (peticiones) HTTP (incluso de forma concurrente) que solicitemos contra una determinada página web. Una vez terminado el proceso, Apache Bench nos da información acerca de los tiempos mínimos, máximos y promedio de tiempo que ha tardado en llevar a cabo estas peticiones, así como la tasa de transferencia o pedidos por segundo que el servidor pudo llevar a cabo. ApacheBench puede resultar muy interesante para realizar un test de prueba que nos permita conocer el rendimiento de un servidor web. Únicamente es necesario ejecutar el comando en la consola [95]: ***Apache24\bin>httpd***

- Siege

Siege es una utilidad de pruebas de carga http y benchmarking. Fue diseñado para permitir a los desarrolladores web medir su código bajo presión, para ver cómo soportará la carga en Internet. Siege soporta autenticación básica, cookies, HTTP, HTTPS y protocolos FTP. Permite a su usuario golpear un servidor con un número configurable de clientes simulados. Esos clientes ponen al servidor "bajo asedio". La duración del asedio se mide en transacciones, la suma de usuarios simulados y el número de veces que cada usuario simulado repite el proceso de golpear al servidor. Por lo tanto, 20 usuarios concurrentes 50 veces es 1000 transacciones, la duración de la prueba. Las medidas de rendimiento incluyen el tiempo transcurrido de la prueba, la cantidad de datos transferidos (incluidas las cabeceras), el tiempo de respuesta del servidor, su tasa de transacción, su rendimiento, su concurrencia y el número de veces que devolvió correctamente. Estas medidas son cuantificadas y reportadas al final de cada corrida. Su significado se discute a continuación. El sitio tiene esencialmente tres modos de operación: regresión, simulación de Internet y fuerza bruta. Puede leer un gran número de URLs desde un archivo de configuración y ejecutarlas de forma incremental (regresión) o aleatoria (simulación de Internet). O el usuario puede simplemente escribir una sola URL con una configuración

en tiempo de ejecución en la línea de comandos ("fuerza bruta") [91]. Los procedimientos de instalación de Siege en Windows es necesario seguir dos pasos. Primero, descargue el último archivo comprimido de Siege. Y luego extraerlo para usarlo desde la línea de comandos [96].

5.2. Hoja de vida y gestión docente

La aplicación que se desarrolló es un CRUD, con varios datos del docente para llenar su hoja de vida por ejemplo datos personales, títulos, experiencia docente, experiencia no docente, capacitaciones, investigaciones, artículos, congresos, libros, etc. Otra parte de la aplicación es la gestión docente igualmente un CRUD en la que cada docente realiza un informe de actividades de fin de cada semestre con la información de alumnos tutorados, cursos, seminarios, congresos y proyectos de vinculación realizados. El desarrollo de esta aplicación fue de apreciar que tan fácil o complicado puede ser realizar una aplicación utilizando ambas stacks. El objetivo principal era para ilustrar la el tiempo de respuesta tanto en la stack MEAN como en la stack Java EE, como también medir su rendimiento y carga de datos. Para el propósito de la prueba, alrededor de 300 registros de docentes y de gestión fueron insertados en las dos aplicaciones.

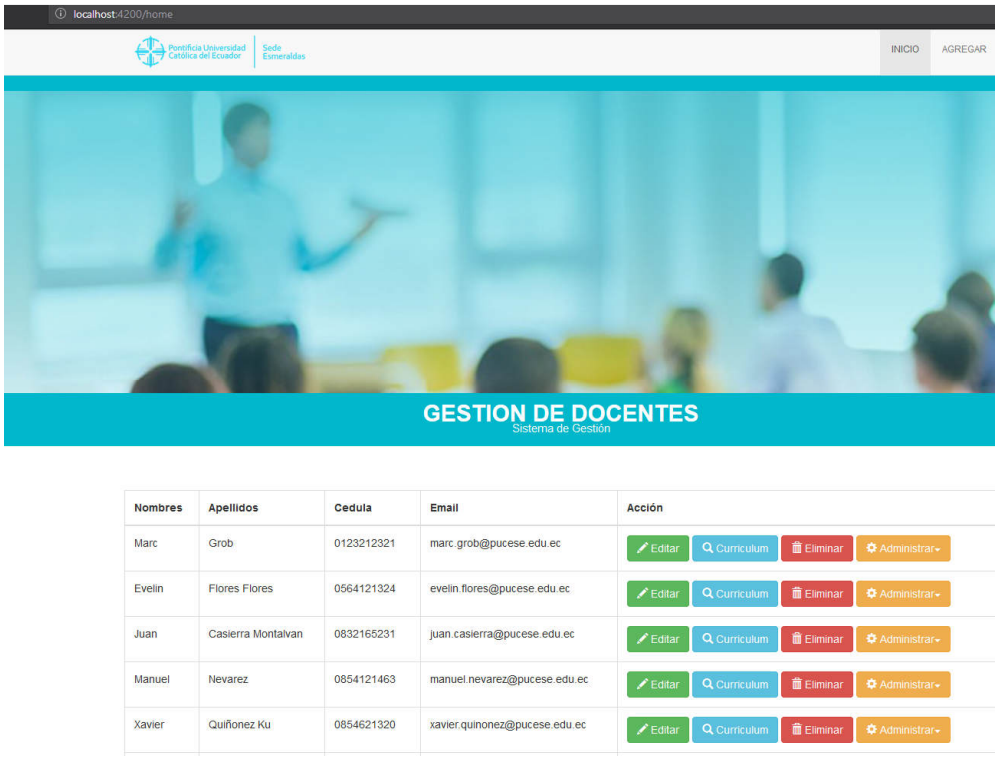

Figura 48. Captura de pantalla de sistema stack MEAN.

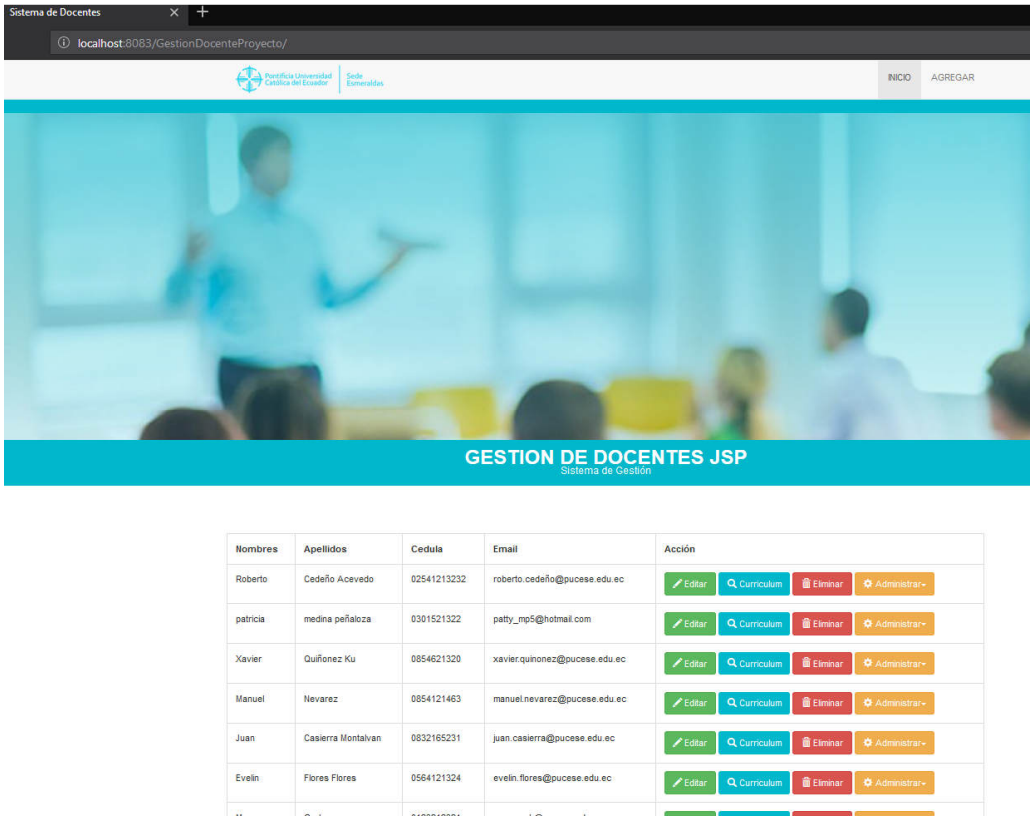

Figura 49. Captura de pantalla de sistema stack Java EE.

5.3. Pruebas y resultados

La máquina utilizada en las pruebas tiene una configuración Core i7 de 3.6GHZ y 16GB. Memoria RAM, usando Windows 10 de 64 bits como sistema operativo. En cualquier desarrollo de aplicaciones, el rendimiento y el equilibrio de carga son esenciales [97]. Para calcular el rendimiento y las pruebas de carga en el servidor, se utilizaron herramientas de evaluación comparativa. Para evaluar el rendimiento de ambas stacks, se realizaron pruebas usando JMeter, Apache Bench y Siege como se describe a continuación:

5.4. Prueba de tiempo respuesta de las aplicaciones

Las mediciones de tiempo de respuesta de las aplicaciones construidas con los dos stacks MEAN y Java EE, se han realizado mediante la herramienta Apache JMeter. El tiempo de respuesta significa el intervalo entre el momento en que se envía una solicitud y el momento en que JMeter recibe la respuesta [98]. Para esta prueba se desarrolló un script

en JMeter para la creación de la petición de datos. Básicamente, el script intenta acceder a la aplicación y realizar una función de búsqueda de información. El experimento se realizó en el entorno donde glassfish ejecuta la aplicación y se conecta al servicio REST para la petición, esto en la stack Java EE y para la stack MEAN el entorno Node.js ejecuta la aplicación al consumir el servicio REST. El script de JMeter se lo ejecuto por separado. La siguiente tabla muestran los resultados de las dos pruebas.

Etiqueta	Muestras	Promedio	Error %	Rendimiento	Kb/seg Recibidos
Java EE	1000	953	0,00%	350,30%	2351,28
MEAN	1000	2	0,00%	352,50%	412,12

Tabla 2. Tiempo de respuesta de las dos stacks MEAN y Java EE.

Como se aprecia en la tabla se hicieron 1000 solicitudes de muestras. El tiempo promedio transcurrido (columna Promedio) fue de 953 milisegundos y el porcentaje de error (columna Error %) fue de 0 para la stack Java EE. Para la stack MEAN el tiempo promedio transcurrido (columna Promedio) fue de 2 milisegundos y el porcentaje de error (columna Error %) fue de 0. El porcentaje de error es la columna que muestra el porcentaje de solicitudes rechazadas de esta tabla. En ambas stacks esta columna es la que no varía entre ejecuciones. El resto de las columnas muestran la rapidez con la que se manejaron las solicitudes (columna de rendimiento). Respecto al rendimiento hay una pequeña diferencia favorable para la stack Java EE 350.30% frente a 352.50% para la stack MEAN. Por último, la cantidad de kilobytes recibidos (columna de kb/segundo recibidos). En esta podemos apreciar que la stack Java EE su velocidad es de 2351.28 kb/segundo frente a 412.12 kb/segundo de la stack MEAN. A continuación, la gráfica de la prueba realizada con el script de JMeter.

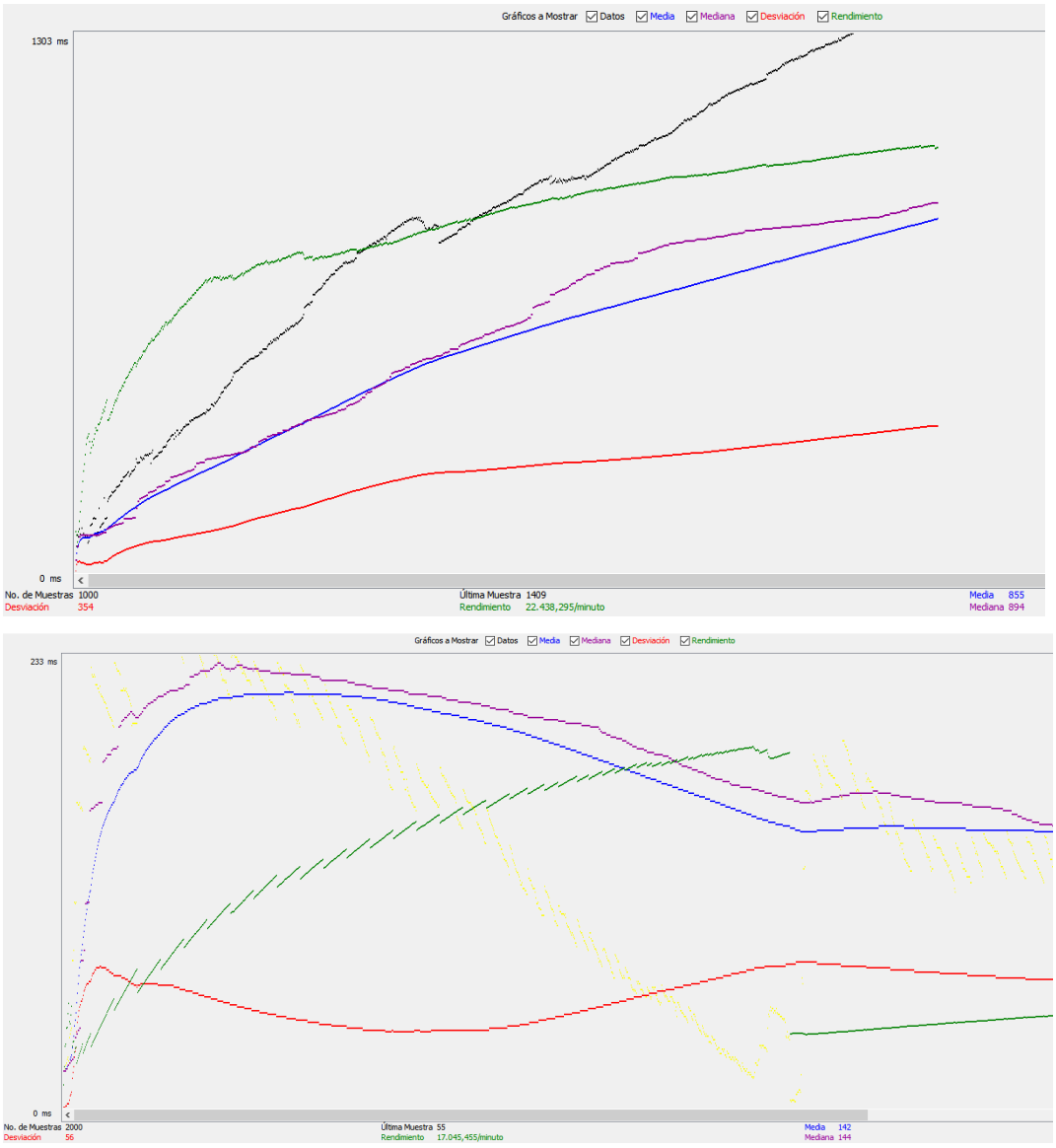

Figura 50. Gráfica JMETER prueba tiempo de respuesta.

La figura 50 muestra los gráficos de los resultados de la prueba realizada que representa el número de peticiones que un servidor web puede manejar.

5.5. Pruebas de rendimiento

Para esta prueba, usamos la herramienta Apache Bench para probar el tiempo de respuesta del sistema de la hoja de vida y gestión docente desarrollado en la stack MEAN y en la stack Java EE. La prueba se ejecutó con varios niveles de concurrencia y número de peticiones. La concurrencia es la medida de cuántas sesiones de usuario simultáneas están activas en una aplicación web en un momento dado. Con un aumento en la concurrencia,

podemos probar cuántas sesiones de usuarios concurrentes puede soportar una aplicación web en términos de tiempo de respuesta para realizar cualquier tarea. Este es un comando de ejemplo para comparar un servidor que se ejecuta localmente en el puerto 8080 con 10.000 solicitudes en total y 500 solicitudes concurrentes:

ab -n 10000 -c 500 http://localhost:8083/GestionDocenteProyecto/

```
Benchmarking localhost (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        GlassFish
Server Hostname:        localhost
Server Port:            8083

Document Path:          /GestionDocenteProyecto/
Document Length:        6582 bytes

Concurrency Level:      500
Time taken for tests:   19.016 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      68930000 bytes
HTML transferred:       65820000 bytes
Requests per second:    525.87 [#/sec] (mean)
Time per request:       950.796 [ms] (mean)
Time per request:       1.902 [ms] (mean, across all concurrent requests)
Transfer rate:          3539.90 [Kbytes/sec] received
```

Figura 51. Código para prueba.

Se realizaron varias pruebas con aumentos en la concurrencia y el número de solicitudes como se muestra en la Tabla 1. Las pruebas se realizaron para MEAN y Java EE con la concurrencia establecida en 10, 50, 100 y 500, y el número de solicitud como 100, 500, 1000, 5000, 10000, 50000 y 100000. La siguiente tabla muestra las solicitudes de respuesta con la concurrencia y su respectivo valor luego de ejecutar la prueba.

	Concurrencia	10		50		100		500	
#peticiones	stack	tiempo(ms)	req/seg	tempo(ms)	req/seg	tempo(ms)	req/seg	tempo(ms)	req/seg
100	JAVA EE	29.073	343.96	152.892	327.03	302.808	330.24	0	0
	MEAN	108.515	92.15	556.959	89.77	1.154.266	86.64	0	0
500	JAVA EE	27.954	357.73	124.687	401.01	256.297	390.17	1.189.665	420.29
	MEAN	30.201	341.93	155.569	362.65	290.660	344.04	1.692.777	295.57
1000	JAVA EE	20.585	485.79	94.854	527.13	198.829	502.95	1.129.968	442.49
	MEAN	19.634	509.32	91.478	546.58	178.723	559.53	1.287.704	388.29
5000	JAVA EE	16.820	594.54	84.694	590.36	163.043	613.33	856.288	583.92
	MEAN	12.398	979.50	46.720	1092.71	89.122	1179.20	801.606	688.06
10000	JAVA EE	16.240	615.78	80.102	624.20	157.729	634.00	846.325	590.79
	MEAN	9.371	1067.07	40.290	1241.01	75.367	1326.84	767.076	651.83
50000	JAVA EE	15.876	629.89	78.306	638.52	156.580	638.65	867.041	576.67
	MEAN	8.971	1114.66	34.834	1435.39	69.537	1438.09	704.577	709.65
	JAVA EE	64.216	165.31	180.072	277.67	224.170	446.09	1.451.031	344.58
100000	MEAN	8.395	1191.14	34.006	1470.35	68.301	1464.11	697.426	716.92

Tabla 3. Tiempo de respuesta con concurrencia stacks MEAN y JAVA EE.

Como se aprecia en la tabla se hicieron las pruebas con varias solicitudes de respuestas que van desde 100 hasta 100000 solicitudes contra la concurrencia que va de 10 hasta 500 solicitudes. El tiempo medio que el servidor ha tardado en atender un grupo de peticiones (columna tiempo) en este caso de 100 respuestas y un nivel de concurrencia de 10 fue de 29.073 milisegundos y la media de peticiones atendidas por segundo durante la prueba (columna req/seg) fue de 343.96 segundos para la stack Java EE. Para la stack MEAN el tiempo medio que el servidor ha tardado en atender un grupo de peticiones (columna tiempo) en el caso de 100 respuestas y un nivel de concurrencia de 10 fue de 108.515 milisegundos y la media de peticiones atendidas por segundo durante la prueba (columna req/seg) fue de 92.15 segundos. Se observa que el procesamiento de peticiones es mejor en Java EE en el servidor al momento atender un grupo de menor cantidad respuestas es decir en las desde 100 hasta 1000. A continuación, la gráfica de la prueba realizada con 100000 respuestas y con un nivel de concurrencia de 10, 50, 100 y 500.

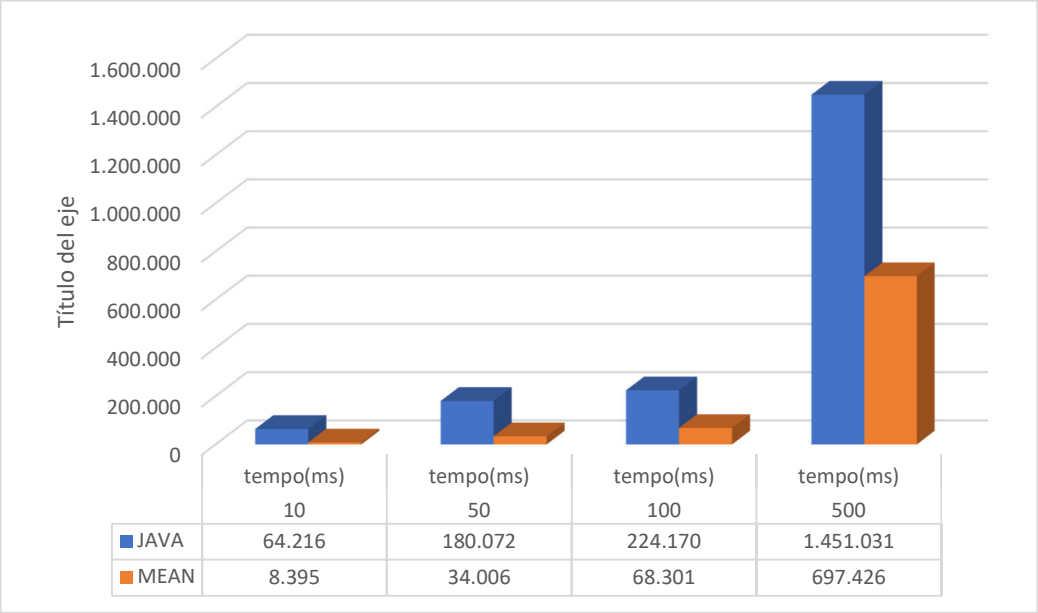

Figura 52. Gráfico de tiempo de respuesta de 100000 solicitudes de respuesta

Observamos en la figura 52 que es la prueba de 100000 respuestas y un nivel de concurrencia de 10, 50, 100 y 500. En la que para la stack Java EE el tiempo medio que el servidor ha tardado en atender un grupo de peticiones fue de 64.216, 180.072, 224.170 y 1.451.031 milisegundos respectivamente. Y para MEAN este tiempo de fue de 8.395, 34.006, 68.301 y 6097.426 milisegundos. Se puede apreciar que mientras aumenta el número de solicitud de respuesta y concurrencia mejora el rendimiento de la stack MEAN sobre el stack Java EE.

5.6. Pruebas de transferencia de datos y concurrencia

Prueba de transferencia de datos y concurrencia utilizando Siege. Para probar cargas en ambas aplicaciones construidas en las stacks MEAN y Java EE, la función Siege se utilizó un instrumento de evaluación comparativa. A continuación, se muestra un comando de ejemplo para comparar un servidor que se ejecuta localmente en el puerto 4200 con 150 peticiones simultáneas, durante un lapso de 5 minutos [99] para la stack MEAN:

- ***siege --concurrent=150 --time=5M http://localhost:4200***

Para la prueba la realización de la prueba para la stack Java EE, se utiliza la ejecución localmente hacia el puerto 8083 donde está alojada la aplicación web. Podemos observar a continuación el mismo ejemplo aplicado a la stack Java EE:

- ***siege --concurrent=150 --time=5M*** http://localhost:8083/GestionDocenteProyecto

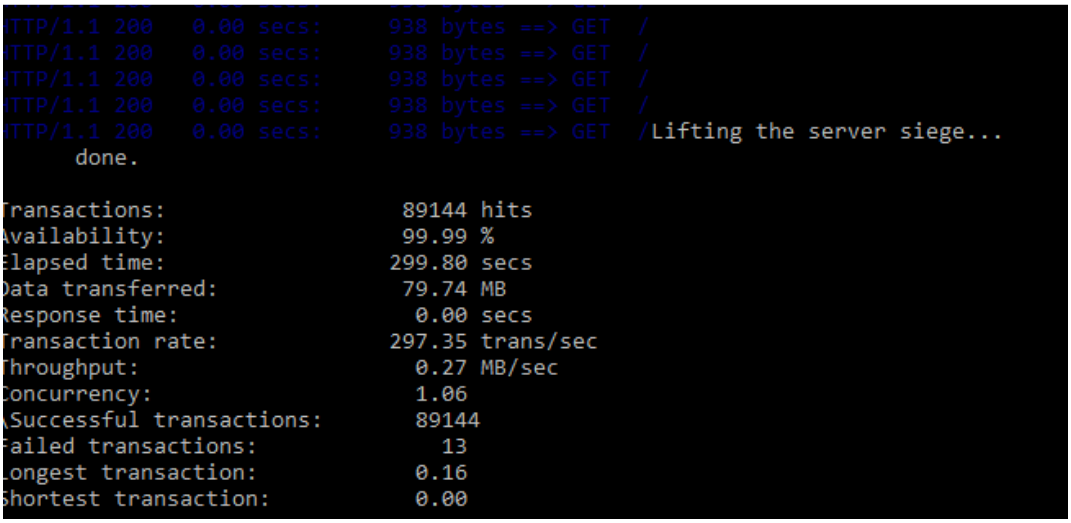

Figura 53. Ejecución de comando Siege para pruebas stack MEAN.

La prueba realizada por Siege proporciona el tiempo transcurrido, la cantidad de datos transferidos, tiempo de respuesta, rendimiento y tasa de transacciones. Estos datos fueron recolectados de Siege pasando el valor de concurrencia y el URL de la aplicación como entrada. Para ilustrar la velocidad de transferencia de datos en el sistema de hoja de vida y gestión docente se utilizó Siege para medir la cantidad de datos transferidos en las aplicaciones de las stacks MEAN y Java EE. Los datos transferidos son la suma de los datos transferidos que incluye la información de cabecera, así como el contenido. El resultado de la prueba de asedio para los datos transferidos para las stacks Java EE y MEAN se muestra en las siguientes tablas y sus respectivos análisis.

Concurrencia 5 usuarios por 5 minutos					
Stacks	Transacciones	Datos	Tasa transacción	Rendimiento	Concurrencia
MEAN	2979	2,66	9,96 trans/seg	0,01 MB/seg	0,03
Java EE	2944	18,48	9,84 trans/seg	0,06 MB/seg	0,08

Tabla 4. Datos transferidos MEAN y Java EE concurrencia de 5 usuarios.

Como se puede observar en la tabla 4 se hicieron las pruebas con el nivel de concurrencia con 5 usuarios por 5 minutos. Transacciones es el número de accesos al servidor. En la prueba, 5 usuarios simulados cada uno golpeó el servidor por 5 minutos con un total de 2979 transacciones para la stack MEAN y 2944 transacciones para la stack Java EE. El tiempo transcurrido es la duración de toda la prueba de asedio que en este caso lo

controlamos en 5 minutos. Los datos transferidos son la suma de los datos transferidos a cada usuario simulado que en el caso de la stack MEAN fue de 2.66 MB y para Java EE de 18.48 MB. La tasa de transacciones es el promedio de transacciones que el servidor fue capaz de manejar por segundo. Esta tasa fue de 9.96 trans/segundo para MEAN y de 9.84 trans/segundo para Java EE. El rendimiento es el número medio de bytes transferidos cada segundo desde el servidor a todos los usuarios simulados. El rendimiento en la stack MEAN fue de 0.01 MB/Segundo y para la stack Java EE 0.06 MB/Segundo. La concurrencia es el número promedio de conexiones simultáneas, un número que aumenta a medida que disminuye el rendimiento del servidor. Podemos ver que este valor para la stack MEAN es de 0.03 y para la stack Java EE es de 0.08. Se observa que los datos transferidos son menos en MEAN, pero su concurrencia es baja, mientras que Java EE los datos transferidos son mayores pero su concurrencia es alta.

Concurrencia 10 usuarios por 5 minutos					
Stacks	Transacciones	Datos	Tasa transacción	Rendimiento	Concurrencia
MEAN	5925	5,3	19,80 trans/seg	0,02 MB/seg	0,06
Java EE	5463	34,29	18,25 trans/seg	0,11 MB/seg	0,16

Tabla 5. Datos transferidos MEAN y Java EE concurrencia de 10 usuarios.

Como se puede observar en la tabla 5 se hicieron las pruebas con el nivel de concurrencia con 10 usuarios por 5 minutos. Transacciones es el número de accesos al servidor. En la prueba, 10 usuarios simulados cada uno golpeó el servidor por 5 minutos con un total de 5925 transacciones para la stack MEAN y 5463 transacciones para la stack Java EE. El tiempo transcurrido es la duración de toda la prueba de asedio que en este caso lo controlamos en 5 minutos. Los datos transferidos son la suma de los datos transferidos a cada usuario simulado que en el caso de la stack MEAN fue de 5.3 MB y para Java EE de 34.29 MB. La tasa de transacciones es el promedio de transacciones que el servidor fue capaz de manejar por segundo. Esta tasa fue de 19.80 trans/segundo para MEAN y de 18.25 trans/segundo para Java EE. El rendimiento es el número medio de bytes transferidos cada segundo desde el servidor a todos los usuarios simulados. El rendimiento en la stack MEAN fue de 0.02 MB/Segundo y para la stack Java EE 0.11 MB/Segundo. La concurrencia es el número promedio de conexiones simultáneas, un número que aumenta a medida que disminuye el rendimiento del servidor. Podemos ver que este valor para la stack MEAN es de 0.06 y para la stack Java EE es de 0.16. Se

observa que los datos transferidos son menos en MEAN, pero su concurrencia es baja, mientras que Java EE los datos transferidos son mayores pero su concurrencia es alta.

Concurrencia 20 usuarios por 5 minutos					
Stacks	Transacciones	Datos	Tasa transacción	Rendimiento	Concurrencia
MEAN	11856	10,61	39,57 trans/seg	0,04 MB/seg	0,13
Java EE	11829	74,25	39,49 trans/seg	0,25 MB/seg	0,34

Tabla 6. Datos transferidos MEAN y Java EE concurrencia 20 usuarios.

Como se puede observar en la tabla 6 se hicieron las pruebas con el nivel de concurrencia con 20 usuarios por 5 minutos. Transacciones es el número de accesos al servidor. En la prueba, 20 usuarios simulados cada uno golpeó el servidor por 5 minutos con un total de 11856 transacciones para la stack MEAN y 11829 transacciones para la stack Java EE. El tiempo transcurrido es la duración de toda la prueba de asedio que en este caso lo controlamos en 5 minutos. Los datos transferidos son la suma de los datos transferidos a cada usuario simulado que en el caso de la stack MEAN fue de 10.61 MB y para Java EE de 74.25 MB. La tasa de transacciones es el promedio de transacciones que el servidor fue capaz de manejar por segundo. Esta tasa fue de 39.57 trans/segundo para MEAN y de 39.49 trans/segundo para Java EE. El rendimiento es el número medio de bytes transferidos cada segundo desde el servidor a todos los usuarios simulados. El rendimiento en la stack MEAN fue de 0.04 MB/Segundo y para la stack Java EE 0.25 MB/Segundo. La concurrencia es el número promedio de conexiones simultáneas, un número que aumenta a medida que disminuye el rendimiento del servidor. Podemos ver que este valor para la stack MEAN es de 0.13 y para la stack Java EE es de 0.34. Se observa que los datos transferidos son menos en MEAN, pero su concurrencia es baja, mientras que Java EE los datos transferidos son mayores pero su concurrencia es alta.

Concurrencia 50 usuarios por 5 minutos					
Stacks	Transacciones	Datos	Tasa transacción	Rendimiento	Concurrencia
MEAN	29705	26,57	99,16 trans/seg	0,09 MB/seg	0,41
Java EE	29487	185,09	98.45 trans/seg	0,62 MB/seg	0,93

Tabla 7. Datos transferidos MEAN y Java EE concurrencia de 50 usuarios.

Como se puede observar en la tabla 7 se hicieron las pruebas con el nivel de concurrencia con 50 usuarios por 5 minutos. Transacciones es el número de accesos al servidor. En la prueba, 50 usuarios simulados cada uno golpeó el servidor por 5 minutos con un total de

29705 transacciones para la stack MEAN y 29487 transacciones para la stack Java EE. El tiempo transcurrido es la duración de toda la prueba de asedio que en este caso lo controlamos en 5 minutos. Los datos transferidos son la suma de los datos transferidos a cada usuario simulado que en el caso de la stack MEAN fue de 26.57 MB y para Java EE de 185.09 MB. La tasa de transacciones es el promedio de transacciones que el servidor fue capaz de manejar por segundo. Esta tasa fue de 99.16 trans/segundo para MEAN y de 98.45 trans/segundo para Java EE. El rendimiento es el número medio de bytes transferidos cada segundo desde el servidor a todos los usuarios simulados. El rendimiento en la stack MEAN fue de 0.09 MB/Segundo y para la stack Java EE 0.62 MB/Segundo. La concurrencia es el número promedio de conexiones simultáneas, un número que aumenta a medida que disminuye el rendimiento del servidor. Podemos ver que este valor para la stack MEAN es de 0.41 y para la stack Java EE es de 0.93. Se observa que los datos transferidos son menos en MEAN, pero su concurrencia es baja, mientras que Java EE los datos transferidos son mayores pero su concurrencia es alta.

Concurrencia 100 usuarios por 5 minutos					
Stacks	transacciones	datos	tasa transacción	rendimiento	concurrencia
MEAN	59533	53,26	198,7 trans/seg	0,18 MB/seg	0,72
Java EE	58222	365.46	194,46 trans/seg	1,22 MB/seg	2,17

Tabla 8. Datos transferidos MEAN y Java EE con concurrencia de 100 usuarios.

Como se puede observar en la tabla 8 se hicieron las pruebas con el nivel de concurrencia con 100 usuarios por 5 minutos. Transacciones es el número de accesos al servidor. En la prueba, 100 usuarios simulados cada uno golpeó el servidor por 5 minutos con un total de 59533 transacciones para la stack MEAN y 58222 transacciones para la stack Java EE. El tiempo transcurrido es la duración de toda la prueba de asedio que en este caso lo controlamos en 5 minutos. Los datos transferidos son la suma de los datos transferidos a cada usuario simulado que en el caso de la stack MEAN fue de 53.26 MB y para Java EE de 365.46 MB. La tasa de transacciones es el promedio de transacciones que el servidor fue capaz de manejar por segundo. Esta tasa fue de 198.7 trans/segundo para MEAN y de 194.46 trans/segundo para Java EE. El rendimiento es el número medio de bytes transferidos cada segundo desde el servidor a todos los usuarios simulados. El rendimiento en la stack MEAN fue de 0.18 MB/Segundo y para la stack Java EE 1.22 MB/Segundo. La concurrencia es el número promedio de conexiones simultáneas, un número que aumenta a medida que disminuye el rendimiento del servidor. Podemos ver

que este valor para la stack MEAN es de 0.72 y para la stack Java EE es de 2.17. Se observa que los datos transferidos son menos en MEAN, pero su concurrencia es baja, mientras que Java EE los datos transferidos son mayores pero su concurrencia es alta.

Concurrencia 150 usuarios por 5 minutos					
Stacks	Transacciones	Datos	Tasa transacción	Rendimiento	Concurrencia
MEAN	88716	79,36	296,07 trans/seg	0,26 MB/seg	1,04
Java EE	87194	547,32	299,65 trans/seg	1,83 MB/seg	3,97

Tabla 9. Datos transferidos MEAN y Java EE concurrencia de 150 usuarios.

Como se puede observar en la tabla 9 se hicieron las pruebas con el nivel de concurrencia con 150 usuarios por 5 minutos. Transacciones es el número de accesos al servidor. En la prueba, 150 usuarios simulados cada uno golpeó el servidor por 5 minutos con un total de 88716 transacciones para la stack MEAN y 87194 transacciones para la stack Java EE. El tiempo transcurrido es la duración de toda la prueba de asedio que en este caso lo controlamos en 5 minutos. Los datos transferidos son la suma de los datos transferidos a cada usuario simulado que en el caso de la stack MEAN fue de 79.36 MB y para Java EE de 574.32 MB. La tasa de transacciones es el promedio de transacciones que el servidor fue capaz de manejar por segundo. Esta tasa fue de 296.07 trans/segundo para MEAN y de 299.65 trans/segundo para Java EE. El rendimiento es el número medio de bytes transferidos cada segundo desde el servidor a todos los usuarios simulados. El rendimiento en la stack MEAN fue de 0.26 MB/Segundo y para la stack Java EE 1.83 MB/Segundo. La concurrencia es el número promedio de conexiones simultáneas, un número que aumenta a medida que disminuye el rendimiento del servidor. Podemos ver que este valor para la stack MEAN es de 1.04 y para la stack Java EE es de 3.97. Se observa que los datos transferidos son menos en MEAN, pero su concurrencia es baja, mientras que Java EE los datos transferidos son mayores pero su concurrencia es alta. A continuación, una tabla resumen de los datos transferidos en MEAN y Java EE, durante las pruebas de carga.

5.6.1. Resumen de datos transferidos

En la siguiente tabla se presenta el resumen de datos transferidos.

	Concurrencia					
Stacks	5	10	20	50	100	150
MEAN	2,66	5,3	10,61	26,57	53,26	79,36
Java EE	18,48	34,29	74,25	185,09	365.46	547,32

Tabla 10. Resumen de datos transferidos MEAN y Java EE concurrencia.

La tabla 9 resume la tasa de transacciones promedio de transacciones que el servidor fue capaz de manejar por segundo. Con los distintos niveles de concurrencia de 10, 20, 50, 100 y 500. Esta tasa fue de es menor de transferencias por segundo para MEAN y es superior los datos de transferencia por segundo para Java EE. Como lo demuestra el gráfico 54.

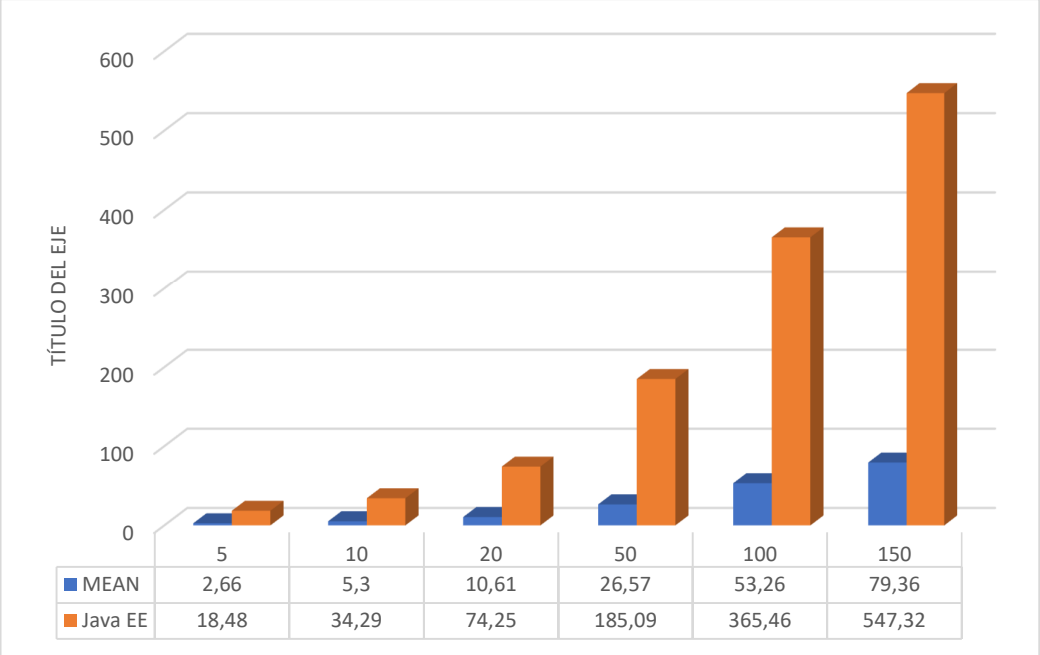

Figura 54. Gráfico de datos transferidos durante la concurrencia.

Como lo podemos observar en la figura 54, se apreciar de mejor manera la transferencia de datos que es mayor para la stack Java EE. Mientras que la stack MEAN es menor.

5.6.2. Resumen de concurrencia

En la siguiente tabla se presenta el resumen de prueba concurrencia.

Concurrencia						
Stacks	5	10	20	50	100	150
MEAN	0,03	0,06	0,13	0,41	0,72	1,04
Java EE	0,08	0,16	0,34	0,93	2,17	3,97

Tabla 11. Resumen de concurrencia en las stacks MEAN y Java EE.

La tabla 10 resume la concurrencia, es decir el promedio de conexiones simultáneas, un número que aumenta a medida que disminuye el rendimiento del servidor. Con los distintos niveles de concurrencia de 10, 20, 50, 100 y 500. Esta medida es menor para la stack MEAN y la medida fue superior para Java EE. Como lo demuestra el gráfico 55.

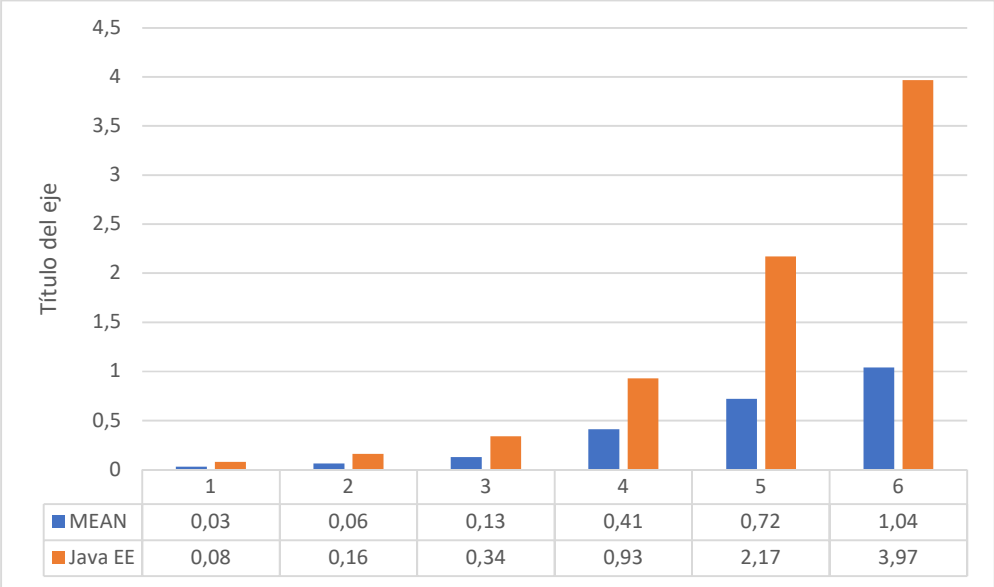

Figura 55. Gráfico de concurrencia de las stack MEAN y Java EE.

Como lo podemos observar en la figura 55, se observa de mejor manera la prueba de concurrencia que es mayor para la stack Java EE. Mientras que la stack MEAN es menor. Esta medida es un parámetro importante en el estudio. Por lo que podemos decir que el servidor tiene un mejor rendimiento en la stack MEAN que el servidor de la stack Java EE.

6. Resultados, conclusiones, recomendaciones y limitaciones

En el estado del arte se hizo una descripción de las tecnologías para arquitecturas y servicios web. Y los más utilizados en la actualidad. Igualmente se describió y analizó dos tipos de tecnologías de stacks para el desarrollo de web aplicaciones utilizando servicios REST: la stack MEAN y la stack Java EE. La stack MEAN es un paquete de software que combina MongoDB como la base de datos NoSQL, Express como un framework de Node.js para el scripting para el desarrollo del back-end, Angular como plataforma MVC para la construcción del front-end, construido con código JavaScript. La stack Java EE es la combinación de MongoDB como el sistema de base de datos, Spring Boot para el back-end y para el front-end se utiliza JSP/HTML/CSS, y Apache/Tomcat como servidor web. En la stack MEAN, JavaScript es el único lenguaje de programación tanto para el front-end y para el back-end. En la stack Java EE, utiliza el lenguaje de programación java del lado del back-end y del lado del front-end junto con HTML/CSS. La creciente popularidad del uso de JavaScript como secuencias de comandos del front-end y del back-end ha hecho que MEAN sea una las combinaciones de tecnologías más utilizadas para desarrollo aplicaciones web y servicios REST. Con MEAN, no necesitamos decidir sobre la combinación de que se utilizará para la creación de scripts de front-end a back-end y la conectividad de bases de datos. MEAN es la mezcla de componentes JavaScript que se presenta como un completo lenguaje de programación. La stack MEAN está construida exclusivamente en JavaScript, por lo que es un lenguaje para gestionar el lado del cliente, servidor y base de datos. Todos los componentes utilizados en la stack MEAN son completamente de código abierto y actualmente son soportados por desarrolladores corporativos como MongoDB y Google.

MEAN es fácil de aprender ya que tanto el front-end como el back-end están construidos usando un solo lenguaje. Todos los componentes de MEAN son relativamente ligeros. Con MEAN, front-end son capaces de entender el código de back-end y los desarrolladores de back-end son capaces de entender el código del front-end. Incluso las consultas a bases de datos en MEAN son más parecidas al código JavaScript. La stack MEAN es flexible y escalable. Angular es una muy buena opción para el desarrollo de una Simple Page Aplication y es responsiva. Node.js es escalable y hace un buen trabajo en el manejo de llamadas asíncronas y en el manejo de muchas tareas concurrentes en un solo hilo. El Node crea un entorno para aplicaciones que requieren tareas de E/S de alto

nivel, como por ejemplo para web para servir múltiples conexiones y peticiones con memoria limitada.

Comparamos la stack MEAN con otra stack muy popular, la stack Java EE, que tiene ya trabajando mucho tiempo con éxito durante décadas. Java EE también es muy fácil de aprender ya que tanto el front-end como el back-end están construidos usando un solo lenguaje de programación Java. Todos los componentes de Java EE son tecnologías ya maduras. En Java EE, front-end son capaces de entender el código de back-end y los desarrolladores de back-end son capaces de entender el código del front-end, utilizan Java como lenguaje principal y adicionalmente HTML/CSS que son lenguaje de marcado muy populares. JSP es utilizada para el desarrollo de aplicaciones web modernas y es responsiva. Apache/Tomcat es escalable y realiza en el manejo de llamadas asíncronas y en el manejo de muchas tareas concurrentes en un a través de multi-threading.

En este estudio, nos centramos en la probar la stack MEAN frente a la stack Java EE con el sistema de la hoja de vida y gestión docente, construida para comparar la stack MEAN contra la stack Java EE.

El sistema de la hoja de vida y gestión docente demostró el rendimiento, incluyendo el rendimiento con solicitud de respuesta con concurrencia, transferencia de datos y concurrencia. El rendimiento de la aplicación se midió con la ayuda de herramientas de evaluación comparativa. El tiempo de respuesta de la stack Java EE es mayor que en comparación con la stack MEAN. La transferencia de datos fue más baja en la stack MEAN en comparación con la stack Java EE. Y en la prueba de concurrencia se observa que la concurrencia es mejor en el stack MEAN ya que es menor en comparación con la stack Java EE.

En este estudio, nos centramos en la puesta en marcha en una aplicación real con las stacks MEAN y Java EE, dando como resultado que la stack MEAN sobresale en relación con la stack Java EE y en las que, la stack MEAN es preferible a la stack Java EE.

Falta hacer mención a situaciones en la que MEAN no debe ser utilizada, puesto que se requiere de mayor cantidad de criterios de análisis y variables de medición. Un análisis

de las deficiencias de la stack MEAN nos proporcionaría con una mejor comprensión MEAN y su aplicabilidad.

Sin embargo, este documento demostró cómo la stack MEAN es buena para el desarrollo de aplicaciones web basados en servicios REST, hay limitaciones en nuestro estudio de caso. Por ejemplo, nuestro estudio de caso no muestra cómo MEAN reacciona con las aplicaciones intensivas y de gran procesamiento en CPU y memoria RAM. Por lo tanto, este documento podría beneficiarse de la introducción de un estudio de caso que se ocupe del uso intensivo y gran procesamiento de la CPU y memoria RAM.

7. Referencias

[1] J. Tihomirovs and J. Grabis, “Comparison of SOAP and REST Based Web Services Using Software Evaluation Metrics,” *Inf. Technol. Manag. Sci.*, vol. 19, no. 1, pp. 92–97, 2016.

[2] R. T. Fielding and R. N. Taylor, “Principled design of the modern Web architecture,” *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, 2002.

[3] S. Mumbaikar and P. Padiya, “Web Services Based On SOAP and REST Principles,” *Int. J. Sci. Res. Publ.*, vol. 3, no. 5, pp. 3–6, 2013.

[4] M. Stajcer and D. Orescanin, “Using MEAN stack for development of GUI in real-time big data architecture,” *2016 39th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. MIPRO 2016 - Proc.*, pp. 524–529, 2016.

[5] A. J. Poulter, S. J. Johnston, and S. J. Cox, “Using the MEAN stack to implement a RESTful service for an Internet of Things application,” *IEEE World Forum Internet Things, WF-IoT 2015 - Proc.*, pp. 280–285, 2015.

[6] R. Salunkhe, S. Telang, P. Shrigondekar, and A. Tanpure, *Review of REST Ful Service Using MEAN Stack for Real Time Big Data Architecture*, vol. 3297, no. 11. Birmingham: Packt Publ, 2007.

[7] N. Rozanski and E. Woods, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2005.

[8] P. Clements, “Coming Attractions in Software Architecture,” 1996.

[9] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001.

[10] D. Garlan and M. Shaw, “An Introduction to Software Architecture,” *Knowl. Creat. Diffus. Util.*, vol. 1, no. January, pp. 1–40, 1994.

[11] P. Kruchten, “Architectural Blueprints — The ‘ 4 + 1 ’ View Model of Software Architecture,” vol. 12, no. November, pp. 42–50, 1995.

[12] S. Engineering and S. Committee, “IEEE Recommended Practice for Architectural Description of Software-Intensive Systems,” 2000.

[13] D. Garlan and D. Garlan, “Software Architecture : a Roadmap Software Architecture : a Roadmap,” 2000.

[14] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Second Edi. Addison Wesley, 2003.

[15] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996.

[16] R. S. Pressman, *INGENIERIA DEL SOFTWARE UN ENFOQUE PRACTICO*. McGraw-Hill Interamericana de España S.L., 2010.

[17] D. Garlan and M. Shaw, “An Introduction to Software Architecture,” *Knowl. Creat. Diffus. Util.*, vol. 1, no. January, pp. 1–40, 1994.

[18] O. Vogel, I. Arnold, A. Chughtai, and T. Kehrer, *Software Architecture: A Comprehensive Framework and Guide for Practitioners*. Springer Berlin Heidelberg, 2011.

[19] D. Perry, A. L. Wolf, T. B. Laboratories, and M. Hill, “Foundations for the Study of Software Architecture,” 1992.

[20] C. Chacón and J. Cárdenas, “Meta Análisis de los Estilos de Arquitectura de Software Orientados a la Web,” Universidad Católica de Colombia, 2017.

[21] S. L. Mora, “Programación de aplicaciones web: historia, principios básicos y clientes web.,” *Editor. Club Univ.*, vol. 53, no. 9, pp. 1689–1699, 2013.

[22] S. Kumari and S. K. Rath, “Performance comparison of SOAP and REST based

Web Services for Enterprise Application Integration," *2015 Int. Conf. Adv. Comput. Commun. Informatics*, pp. 1656–1660, 2015.
[23] F. Bocchio, "Estudio Comparativo De Plataformas Cloud Computing Para Arquitecturas Soa," vol. 1, no. 5, p. 114, 2013.
[24] P. Chung, Y. Huang, S. Yajnik, D. Liang, and J. Shih, "DCOM and CORBA side by side, step by step, and layer by layer," *C++ Rep.*, vol. 10, no. 1, pp. 18–29, 1998.
[25] R. Mateosian, "Enterprise computing," *IEEE Micro*, vol. 22, no. 3, pp. 71–72, 2002.
[26] S. Malik and D. H. Kim, "A comparison of RESTful vs. SOAP web services in actuator networks," *Int. Conf. Ubiquitous Futur. Networks, ICUFN*, pp. 753–755, 2017.
[27] A. Arcuri, "RESTful API Automated Test Case Generation," *2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur.*, pp. 9–20, 2017.
[28] D. Booth *et al.*, "Web Services Architecture," *W3C Note*, vol. 22, no. February, pp. 1--98, 2004.
[29] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web services architecture," *IBM Syst. J.*, vol. 41, no. 2, pp. 170–177, 2002.
[30] S. Patni, *Pro RESTful APIs*. 2017.
[31] D. Chappell and T. Jewell, *Java Web Services*. 2002.
[32] J. Sandoval, *RESTful Java Web Services*, First. Birmingham: Pack Publishing Ltd, 2009.
[33] G. Barbaglia, S. Murzilli, and S. Cudini, "Definition of REST web services with JSON schema," *Softw. - Pract. Exp.*, vol. 47, no. 6, pp. 907–920, 2016.
[34] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," *Building*, vol. 54, p. 162, 2000.
[35] M. Zur Muehlen, J. Nickerson, and K. Swenson, "Developing web services choreography standards - The case of REST vs. SOAP," *Decis. Support Syst.*, vol. 40, no. 1 SPEC. ISS., pp. 9–29, 2005.
[36] D. Fensel, F. M. Facca, E. Simperl, and I. Toma, *Semantic Web Services*, 1st ed. Springer Publishing Company, Incorporated, 2011.
[37] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. 'big'web services: making the right architectural decision," *Proceeding 17th Int. Conf. World Wide Web*, pp. 805–814, 2008.
[38] H. Li, "RESTful Web service frameworks in Java," *2011 IEEE Int. Conf. Signal Process. Commun. Comput. ICSPCC 2011*, no. 1, 2011.
[39] "Desarrollo web con stacks de software - 1&1." [Online]. Available: https://www.1and1.es/digitalguide/servidores/know-how/desarrollo-web-con-stacks-de-software/. [Accessed: 03-Jun-2018].
[40] Daria Khmelnitskaya, "Which Technology Stacks Are Most Popular Among Startups?," 2017. [Online]. Available: http://www.iamwire.com/2017/08/technology-stacks-popular-startups/156724. [Accessed: 03-Jun-2018].
[41] C. Wodehouse, "What is a Software Stack? Choose the right stack for your project," 2015. [Online]. Available: https://www.upwork.com/hiring/development/choosing-the-right-software-stack-for-your-website/. [Accessed: 03-Jun-2018].
[42] 1&1, "Desarrollo web con stacks de software," 2016. [Online]. Available: https://www.1and1.es/digitalguide/servidores/know-how/desarrollo-web-con-stacks-de-software/. [Accessed: 03-Jun-2018].

[43] S. Holmes, *Getting MEAN with Mongo, Express, Angular, and Node*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2015.
[44] E. Jendrock, R. Cervera-Navaroo, I. Evans, K. Haase, and W. Markito, "Java Platform, Enterprise Edition The Java EE Tutorial, Release 7," no. September, pp. 1–980, 2014.
[45] IBM, "IBM Java EE 7 Application Server," 2018. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSEQTP_9.0.0/com.ibm.websphere.base.doc/ae/covr_javaee7.html. [Accessed: 03-Jun-2018].
[46] JHipster, "Technology stack," 2017. [Online]. Available: https://www.jhipster.tech/tech-stack/. [Accessed: 03-Jun-2018].
[47] Daniel Ortego Delgado, "Los 7 mejores frameworks de Java de 2017," 2018. [Online]. Available: https://openwebinars.net/blog/los-7-mejores-frameworks-de-java-de-2017/. [Accessed: 04-Jun-2018].
[48] D. Flanagan, *JavaScript - The Definitive Guide*. 2011.
[49] A. Q. Haviv, *MEAN Web Development*, vol. 1. 2014.
[50] K. Chodorow, *Mongo DB: The Definitive Guide*. 2013.
[51] R. O. Obe and L. S. Hsu, *MongoDB in Action*. 2011.
[52] A. Morgan, "The MEAN Stack: MongoDB, ExpressJS, AngularJS and Node.js | MongoDB," 2013. [Online]. Available: https://www.mongodb.com/blog/post/the-mean-stack-mongodb-expressjs-angularjs-and. [Accessed: 26-Feb-2018].
[53] J. Alarcón, "Fundamentos de bases de datos NoSQL: MongoDB," 2014. [Online]. Available: https://www.campusmvp.es/recursos/post/Fundamentos-de-bases-de-datos-NoSQL-MongoDB.aspx. [Accessed: 26-Feb-2018].
[54] MongoDB, "MongoDB Architecture Guide," *MongoDB White Pap.*, no. June, pp. 1–16, 2016.
[55] E. Brown, *Web Development with Node and Express: Leveraging the JavaScript Stack*. O'Reilly Media, 2014.
[56] E. Hahn, *Express in Action: Node Applications with Express and Its Companion Tools*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2015.
[57] R. K. Soni, *Full Stack AngularJS for Java Developers*. 2017.
[58] Angular, "Angular - Architecture Overview," 2018. [Online]. Available: https://angular.io/guide/architecture. [Accessed: 27-Feb-2018].
[59] L. Ruebbelke, "5 Awesome AngularJS Features," 2012. [Online]. Available: https://code.tutsplus.com/tutorials/5-awesome-angularjs-features--net-25651. [Accessed: 27-Feb-2018].
[60] S. Davis, "Mastering MEAN: Introducing the MEAN stack," *IBM.com*, pp. 1–20, 2014.
[61] M. Cantelon, M. Harter, T. J. Holowaychuk, and N. Rajlich, *Node.Js in Action*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2013.
[62] S. Pasquali, *Mastering NodeJS: Expert Techniques for Building Fast Servers and Scalable, Real-Time Network Applications with Minimal Effort*. 2013.
[63] TutorialsPoint, "Node.js Quick Guide," 2016. [Online]. Available: https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm. [Accessed: 27-Feb-2018].
[64] C. J. Ihrig and A. Bretz, *Full Stack JavaScript Development With MEAN: MongoDB, Express, AngularJS, and Node.JS*. SitePoint, 2014.
[65] J. Bloch, *Effective Java (2Nd Edition) (The Java Series)*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008.
[66] S. Daschner, *Architecting Modern Java EE Applications: Designing lightweight,*

business-oriented enterprise applications in the age of cloud, containers, and Java EE 8. Packt Publishing, 2017.
[67] F. Gutierrez, *Pro Spring Boot*. 2016.
[68] R. Johnson, J. Höller, A. Arendsen, T. Risberg, and C. Sampaleanu, *Professional Java Development with the Spring Framework*. Wiley, 2007.
[69] N. S. Williams, *Professional Java for Web Applications*. Wiley, 2014.
[70] J. Brisbin, O. Gierke, and G. Turnquist, "Spring Data REST - Reference Documentation," 2015. [Online]. Available: https://docs.spring.io/spring-data/rest/docs/current/reference/html/. [Accessed: 06-Apr-2018].
[71] B. Varanasi and S. Belida, *Spring REST*. 2015.
[72] B. Muschko, *Gradle in Action*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2014.
[73] S. Daschner, *Architecting Modern Java EE Applications: Designing lightweight, business-oriented enterprise applications in the age of cloud, containers, and Java EE 8*. Packt Publishing, 2017.
[74] Gradle, "Building Java Web Applications." [Online]. Available: https://guides.gradle.org/building-java-web-applications/?_ga=2.55385899.1230000254.1523048007-1380569645.1523048007. [Accessed: 07-Apr-2018].
[75] F. P. Miller, A. F. Vandome, and J. McBrewster, *Apache Maven*. Alpha Press, 2010.
[76] H. Schildt, "The Complete Reference."
[77] W3C, "HTML 5.2 Recomendation," 2017. [Online]. Available: https://www.w3.org/TR/html52/. [Accessed: 06-May-2018].
[78] J. D. Gauchat, *El gran libro de HTML5. CSS3 y Javascript*, vol. 354, no. 9. 2012.
[79] B. Noel and A. Chopra, "Introduction To Javaserver Pages," 2001.
[80] J. C. Ojeda, *Introducción a la Programación Web con Java: JSP y Servlets , JavaServer Faces*. 2017.
[81] R. Mordani and S. Chan, "Java™ Servlet Specification," *Sun Microsystems Inc., version*, no. May, 2009.
[82] H. Quintana, "Implementación de aplicaciones isomórficas con JavaScript," *Rev. ULIMA*, no. ISSN 1993-4912, pp. 143–161, 2015.
[83] L. Castro, "¿Qué es escalabilidad?," 2016. [Online]. Available: https://www.aboutespanol.com/que-es-escalabilidad-157635. [Accessed: 13-Apr-2018].
[84] P. Webb *et al.*, "Spring Boot Reference Guide Table of Contents," 2018.
[85] K. T. Lam, Y. Luo, and C. L. Wang, "A performance study of clustering web application servers with Distributed JVM," *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS*, pp. 328–335, 2008.
[86] Tutorials Point, "Node.js," 2016.
[87] Diego Alvarez, "¿Qué es Atom? · GitBook," 2016. [Online]. Available: https://ull-esit-dsi-1617.github.io/estudiar-las-rutas-en-expressjs-alberto-diego/Diego/Atom/queesatom.html. [Accessed: 11-Jun-2018].
[88] GitHub, "Atom Basics." [Online]. Available: https://flight-manual.atom.io/getting-started/sections/atom-basics/. [Accessed: 11-Jun-2018].
[89] Eclipse, "Eclipse Documentation." [Online]. Available: http://help.eclipse.org/oxygen/index.jsp. [Accessed: 06-May-2018].
[90] W. Beaton and J. d Rivieres, "Eclipse platform technical overview," *Retrieved Novemb.*, vol. 2003, no. July 2001, pp. 1–20, 2006.
[91] "Siege Home," 2012. [Online]. Available: https://www.joedog.org/siege-home/.

[Accessed: 14-May-2018].
[92] “Apache HTTP server benchmarking tool.” [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html. [Accessed: 14-May-2018].
[93] “Apache JMeter - Apache JMeter™.” [Online]. Available: https://jmeter.apache.org/. [Accessed: 13-May-2018].
[94] “Load Testing Rails Apps with Apache Bench, Siege, and JMeter | Steve Grossi at Work.” [Online]. Available: https://work.stevegrossi.com/2015/02/07/load-testing-rails-apps-with-apache-bench-siege-and-jmeter/. [Accessed: 13-May-2018].
[95] “Install apache bench on windows 7,” 2014. [Online]. Available: https://stackoverflow.com/questions/7327099/how-to-install-apache-bench-on-windows-7. [Accessed: 14-May-2018].
[96] “Siege-windows,” 2016. [Online]. Available: https://github.com/ewwink/siege-windows. [Accessed: 14-May-2018].
[97] Zoran Antolovic, “Web App Performance Testing with Siege: Plan, Test, Learn — SitePoint,” 2017. [Online]. Available: https://www.sitepoint.com/web-app-performance-testing-siege-plan-test-learn/. [Accessed: 14-May-2018].
[98] “Prueba de rendimiento usando Jmeter.” [Online]. Available: https://www.guru99.com/jmeter-performance-testing.html. [Accessed: 14-May-2018].
[99] “Load Testing and Benchmarking With Siege.” [Online]. Available: https://www.serverwatch.com/tutorials/article.php/3936526/Load-Testing-and-Benchmarking-With-Siege.htm. [Accessed: 14-May-2018].