CTES
Revista Electrónica sobre
Ciencia, Tecnología y Sociedad
ISSN: 2448 - 6493

Artículos científicos

Arquitectura REST para el desarrollo de aplicaciones web empresariales

REST architecture for enterprise web application development

Luis Miguel Alamilla Hernández
Instituto Tecnológico de Villahermosa, México
alamilla-96@hotmail.com

Viviano Alonso Pérez Romero
Instituto Tecnológico de Villahermosa, México
alonsoperom@gmail.com

Sergio Arturo Sosa González
Instituto Tecnológico de Villahermosa, México
sergioarsosa95@gmail.com

José Adrián Valentín Rodríguez
Instituto Tecnológico de Villahermosa, México
joadva15@gmail.com

Resumen

Las empresas dedicadas a desarrollar aplicaciones distribuidas tienden a enfrentarse con ciertas incertidumbres al momento de elegir la arquitectura para implementar sus aplicaciones de desarrollo, por lo tanto, en sus inicios implementaron una arquitectura tradicional y creen que algo que les funcionó anteriormente les dará resultado. Entonces llevan su proyecto con una arquitectura monolítica, pero tiempo después la aplicación tiene la necesidad de crecer y se dan cuenta que el tipo de arquitectura seleccionada no fue la adecuada, ya que no pueden manejar los fallos del sistema eficientemente y no permite escalabilidad y modularidad.

Ante esta situación surgió SOA como una mejora de las aplicaciones monolíticas. Aunque en muchos aspectos SOA es más sencilla que una arquitectura monolítica, conlleva a un riesgo de cambios en cascada en todo el entorno si las interacciones de los componentes no se comprenden claramente, situación que para una organización conlleva a cuantiosas pérdidas si no se planea bien.

CTES
Revista Electrónica sobre
Ciencia, Tecnología y Sociedad
ISSN: 2448 - 6493

Con respecto a REST en una organización, incorpora mejoras sustanciales lo que permite descomponer las arquitecturas tradicionales en partes más pequeñas, los servicios que implementa una arquitectura de micro servicios usando un marco de mensajería común, es API RESTful. En donde cada servicio es independiente, permitiendo que un servicio pueda ser reemplazado, mejorarlo o abandonarlo, sin afectar a los demás servicios de la arquitectura.

Palabras claves: REST, RESTful, SOA, Aplicaciones monolíticas, servicio web.

Abstract

Companies dedicated to developing distributed applications tend to face certain uncertainties when choosing the architecture to implement their development applications, therefore, in the beginning they implemented a traditional architecture and believe that something that worked previously will work for them. Then they take their project with a monolithic architecture, but sometime later the application has the need to grow and they realize that the type of architecture selected was not the right one, since they cannot handle system failures efficiently and does not allow scalability and modularity.

SOA emerged as an improvement on monolithic applications. Although in many ways SOA is simpler than a monolithic architecture, it carries a risk of cascading changes throughout the environment if the interactions of the components are not clearly understood, a situation that for an organization leads to heavy losses if it is not well planned.

With respect to REST in an organization, it incorporates substantial improvements that allow traditional architectures to be broken down into smaller parts, the services that implement a micro services architecture using a common messaging framework, is API RESTful. Where each service is independent, allowing a service to be replaced, improved or abandoned, without affecting the other services in the architecture.

Keywords: REST, RESTful, SOA, monolithic applications, web service.

Fecha Recepción: Junio 2020 **Fecha Aceptación:** Diciembre 2020

Introducción

En la actualidad la tecnología avanza a pasos agigantados, exigiendo con ello nuevas formas de comunicación entre las diferentes aplicaciones de las empresas, ya que su principal objetivo es brindar un producto y servicio de calidad. Las empresas buscan optimizar las actividades en sus procesos con las nuevas tecnologías que la industria ofrece, no obstante, esto impone una inversión considerable para dicha implementación.

A nivel empresarial, tanto en el sector privado como público se realiza desarrollo de software para suplir las necesidades de automatización de procesos internos, este desarrollo ha seguido las tendencias impuestas por la plataforma, lenguaje de programación o por la experiencia del área de desarrollo. lo cual deviene en la implantación de sistema de construcción tradicional o monolítico. Una mala elección en la arquitectura en el momento de la entrega de los sistemas, representa insatisfacción en el cliente final y para las organizaciones pérdidas considerables no negociables.

Las organizaciones están interesadas en proveer sistemas robustos y escalables. Como los procesos de negocios son realizados en espacios de información tecnológica compleja, la integración de los sistemas de información existentes se convierte en una base importante para la implementación técnica de los procesos de negocio.

Metodología

Se consideraron como población las empresas del sector privado de la república mexicana, se utilizó un método de búsqueda exhaustiva. Para el muestreo de la población se dividió en dos vertientes, considerando que las empresas del sector privado tienen ciertos esquemas de privacidad. Se utilizó la información de manera objetiva y parcial para el cálculo de las muestras que se ilustran a continuación.

Análisis de aplicaciones monolíticas para su migración a arquitecturas basadas en SOA

Una aplicación informática es un software que permite al usuario realizar una o más tipos de trabajos, existen tipos de aplicaciones:

- Aplicaciones Monolíticas.
- Aplicaciones Cliente/Servidor.
- Aplicaciones De 2, 3, N capas.
- Aplicaciones Distribuidas.

Las aplicaciones Monolíticas son aquellas que las tres partes forman un todo y se ejecutan en el mismo ordenador: Interfaz de usuario, Lógica o reglas de negocios y Gestión de datos. [1].

Una arquitectura monolítica es una única unidad, que se compone de tres partes: acceso a datos también llamado la capa del Modelo, interfaz de usuario como la vista y el controlador el cual comunica la capa de la base de datos con la capa de la interfaz. [8]

Las aplicaciones monolíticas emplean una tecnología que limita la disponibilidad de herramientas indispensables para desplegar el sistema.

Dicha arquitectura tiene un diseño para ser considerada autónoma, sus componentes ligados en el código son desplegados dentro de un solo compilador en el lenguaje de programación donde se realiza la aplicación, obligado a estar interconectado y ser dependiente de sí mismo para su funcionalidad.

Desde un punto de vista empresarial, el diseño se consideró como una de las primeras aplicaciones de software que se desarrolló con importantes ventajas como: ejecución de un solo archivo, despliegue único y facilidad de desarrollo. Al implementar esta arquitectura, el desarrollo de aplicaciones monolíticas era menos costoso, por lo que se concentre en la funcionalidad de un solo archivo de despliegue.

Para una empresa al momento de elegir implementar una arquitectura monolítica debe tener en cuenta la lógica de negocio y considerar que no es conveniente atar a una solo unidad la aplicación, sino más bien decidirse por una arquitectura que permita escalabilidad, interoperabilidad y ser de fácil implementación.

Para poner en contexto hoy en día, desarrollar un CRM(Customer Relationship Management) con arquitectura monolítica donde se tiene funcionalidades complejas y lógica de negocio en crecimiento, en donde se debe realizar un despliegue de todo el desarrollo por más mínimo cambio a un componente, implica un alto riesgo de errores y/o fallas en el despliegue y actualización de la aplicación, por ende se realiza menos frecuente, ya que requiere más ciclos de pruebas y la comunicación entre los equipos de desarrollo. Además, la calidad del mantenimiento del código no es garantizado y es muy complicado encontrar dependencias entre diferentes módulos.

En concreto, una empresa, al querer realizar un cambio funcional, se topará con el problema del despliegue y tendrá sus desarrolladores atentos a las fallas, además de convivir con las contingencias hasta que se realice las correcciones y se despliega una actualización.

Usar una arquitectura alternativa que permita escalabilidad, flexibilidad, interoperabilidad, autonomía y una eficiencia a la gestión multifuncional, es posible ya que la Arquitectura Orientada a Servicios (SOA) tiene mucho que aportar.

Arquitectura orientada al servicio (SOA) para diseñar e implementar Web Services

Actualmente, con el gran apogeo en que se encuentra el internet es muy común el término Web Services como artefactos de software a que a partir de estos se pueden construir aplicaciones más complejas.

Según el Instituto Nacional de Estándares y Tecnología (NIST), el Web Service es una arquitectura por capas que consiste en: (1) la capa de Web Services, (2) la capa del framework del Web Service y (3) la Capa Web Server. Según Mokbel y Jiajin (2008), el objetivo de diseñar la arquitectura de seguridad es resumir los detalles de seguridad a nivel del mensaje de la lógica de negocios. La ISO-WSP (Web Services Platforms) es una arquitectura de flujo de información que descompone la WSP en dos partes y que se ejecuta en dominios de protección diferentes: (1) el T-WSP: maneja la seguridad de datos confidenciales y (2) el U-WSP: que contiene amplio código que proporciona la normal funcionalidad del WS (Singaravelu, Wei y Pu, 2008). [2]

Según la W3C, un Web Service es: "Una aplicación de software identificada por una URI, cuya interfaz y enlaces son capaces de ser definidos, descritos y descubiertos como artefactos XML. Un web service soporta interacción directa con otros agentes de software usando mensajes basados en XML intercambiados a través de protocolos basados en internet".

Los Web Services (Web Services) permiten enlazar, mediante el uso de diferentes protocolos, cualquier tipo de tecnología o servicio en la web, logrando que aplicaciones legado se comuniquen con otras aplicaciones de una organización o de cualquier tercero.

La implementación de estos componentes, los Web Services, se ha masificado, siendo un importante punto de valor para las organizaciones, ya que allí confluyen varios núcleos de negocio por los cuales se transfiere información vital para las organizaciones, desde datos personales, información financiera, hasta todo tipo de información que pueda estar tipificada como confidencial. [7]

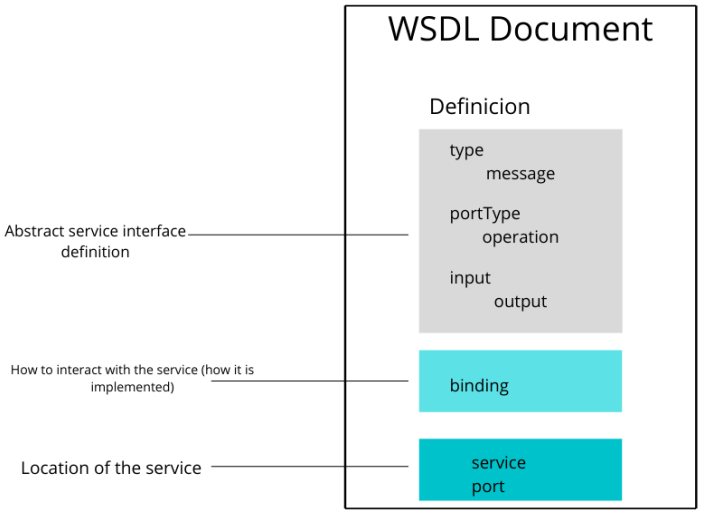
Para muchas empresas el construir sus sistemas con una arquitectura orientada a servicios web, facilita el publicar al mundo dicho sistema, además de ser utilizados de manera interna por sus desarrolladores. Esta afirmación permite que cada equipo de trabajo elija las herramientas de desarrollo que desee usar para construir sus proyectos sin afectar la vinculación con los demás sistemas. [7]

La intención de componer un Web Service para que sea utilizado en cualquier organización en el entorno o cualquier otra persona ha hecho posible la creación de lenguajes y estándares formales de definición de los mismos. No obstante, la alta abstracción para la fabricación y

especificación dificulta en gran medida su comprensión. Cabe recalcar que un Web Services es una interfaz al mundo exterior de una aplicación de software que puede ser atacado o vulnerado.

Todo Web Service posee una especificación que provee la información necesaria para invocarlo. Uno de los estándares de descripción más conocido es WSDL (Web Service Definition Language) [3]. La especificación WSDL es un idioma de XML, teniendo reglas definidas donde se especifica cada componente del Web Services.

Figura 1. Documento WSDL



Fuente: Elaboración propia

Como se muestra en la ilustración 1, así como el archivo WSDL es de utilidad para un desarrollador de software o cualquier persona que pueda implementarlo para hacer uso del servicio web que describe, además tiene la posibilidad de dar información a agentes no deseados o inclusive exponer vulnerabilidades del Web Services. Considerando que existen herramientas que generan WSDL de forma automática para un Web Services, con el alto nivel de atención a la información que es publicada no está exenta de un bajo control. Dicho control es muy importante para aquellos Web Services que pertenecen a bancos, servicios de compra online, entre otros.

Empresas competidoras pueden aprender el know-how y conseguir copiar el diseño para ofrecer servicios similares y competitivos. Pero no solo se trata de competencia, los ataques de seguridad como espionaje de información, suplantación de clientes, inyección de comandos y denegación de servicio también son posibles ya que los atacantes pueden aprender sobre los datos intercambiados y los patrones de invocación de los documentos WSDL. Si bien la

legibilidad de las descripciones de los servicios hace que los servicios web sean reconocibles, también contribuye a la vulnerabilidad del servicio [4].

Para lograr la flexibilidad, interacción e integración, se requiere la existencia de una arquitectura que preste estos servicios de interacción en distintas capas, para lo cual existe la arquitectura orientada a servicios (SOA Service Oriented Architecture) [5] que hace uso de los Servicios Web o Web Services [6], para lograr el objeto de la integración y flexibilidad de las aplicaciones.

SOA frente a la arquitectura de microservicios

SOA es un estilo de Arquitectura de Software basado en la definición de servicios reutilizables, con interfaces públicas bien definidas, donde los proveedores y consumidores de servicios interactúan en forma desacoplada para realizar los procesos de negocio. Se basa en cuatro abstracciones básicas: servicios, Application Frontend, repositorio de servicios y bus de servicios. Un servicio consiste en una implementación que provee lógica de negocio y datos, un contrato de servicio, las restricciones para el consumidor, y una interfaz que expone físicamente la funcionalidad. Las Application Frontend consumen los servicios formando procesos de negocios. Un repositorio de servicios almacena los contratos de servicios y el bus de servicios interconecta las Application Frontend y los servicios. [10]

Los dos enfoques de arquitectura que más se utilizan para las API remotas son la arquitectura orientada al servicio (SOA) y la arquitectura de microservicios. La SOA es el más antiguo de los dos, y comenzó como una mejora de las aplicaciones monolíticas. En lugar de usar una sola aplicación que haga todo, se pueden usar varias aplicaciones que proporcionan diferentes funciones y que no tengan conexión directa, todo gracias a un patrón de integración, como un bus de servicios empresariales (ESB). [9]

No obstante, en numerosos aspectos SOA es mucho más fácil respecto a una arquitectura monolítica, llevando consigo un riesgo de cambios en cascada del ambiente siempre y cuando las interacciones de los componentes no son comprendidas rotundamente. Dicha dificultad adicional se puede presentar en varios de los problemas que SOA pretende resolver.

Sin embargo, dada la naturaleza compleja y cambiante de los requerimientos del negocio en el contexto de las Organizaciones actuales, es recomendable que se siga un enfoque iterativo incremental como forma de lidiar con estos cambios, y con fuerte enfoque en la generación de productos intermedios como forma de ir obteniendo productos claves en el avance del mismo que proveen visibilidad sobre el desarrollo.

Arquitectura orientada a microservicios

Las arquitecturas de microservicios se parecen a los patrones SOA en que los servicios son especializados y no tienen conexión directa. Pero, además, descomponen las arquitecturas tradicionales en partes más pequeñas. Los servicios de la arquitectura de microservicios usan un marco de mensajería común, como las API de RESTful. Utilizan API de RESTful para comunicarse entre sí, sin necesidad de operaciones complejas de conversión de datos ni capas de integración adicionales. Usar las API de RESTful permite e incluso fomenta una distribución más rápida de nuevas funciones y actualizaciones. Cada servicio es independiente. Un servicio se puede reemplazar, mejorar o abandonar, sin afectar los demás servicios de la arquitectura. Esta arquitectura liviana optimiza los recursos distribuidos o en la nube y admite la escalabilidad dinámica de los servicios individuales. [9]

Características que implementa una arquitectura REST

REST no es un estándar, claramente define unos principios de arquitectura a seguir para implementar aplicaciones o servicios web. No obstante, REST se basa en estándares para su implementación: HTTP, XML. Los servicios REST tienen las siguientes características:

- Las operaciones más importantes que permiten manipular los recursos son cuatro: GET para consultar y leer; POST para crear; PUT para editar; DELETE para eliminar.
- El uso de hipermedias para permitir al cliente navegar por los distintos recursos de una API REST a través de enlaces HTML.
- La respuesta del API REST son por lo general JSON independientemente del lenguaje que se utilice.

Restricciones que definen a las API RESTful

Las API son RESTful siempre que cumplan con 6 restricciones fundamentales de un sistema RESTful:

1) Arquitectura cliente-servidor: esta restricción mantiene a la arquitectura REST compuesta por clientes, servidores y recursos, administrando las peticiones o solicitudes con HTTP.
2) Sin Estado: para esta restricción el contenido de los clientes nunca se almacena en el servidor entre las peticiones, por consiguiente, la información sobre el estado de la sesión se queda en el cliente.
3) Capacidad de caché: el almacenamiento por parte del cliente en el caché logra eliminar la necesidad y consumo de memoria al hacer interacciones cliente-servidor.

4) Sistemas en capas: las interacciones cliente-servidor logra mediaciones por capas adicionales, que permiten ofrecer otras funcionalidades, como el equilibrio de carga, los cachés compartidos o la misma seguridad.
5) Código de demanda: los servidores pueden extender las funciones de un cliente transfiriendo código ejecutable.
6) Interfaz uniforme: define una interfaz genérica para administrar cada una de las interacciones que se vayan produciendo entre el cliente-servidor de manera uniforme, lo cual separa y simplifica la arquitectura. No obstante, es fundamental para el diseño de las API RESTful incluir 4 aspectos:
    a) Identificación de recursos e n las solicitudes: los recursos se identifican en las solicitudes y estos son separados de las representaciones que se devuelvan al cliente.
    b) Administración de recursos mediante representaciones: los clientes pueden recibir archivos que representan recursos. Dichas representaciones deben tener la información suficiente como para poder ser modificadas o eliminadas.
    c) Mensajes autodescriptivos: Cada mensaje que se devuelve al cliente contiene la información suficiente para describir cómo debe procesar la información.
    d) Hipermedios es el motor del estado de la aplicación: después de acceder a un recurso, el cliente REST debe ser capaz de descubrir mediante hipervínculos todas las otras acciones que están disponibles actualmente.

Estas limitaciones pueden parecer demasiadas, pero son mucho más sencillas que un protocolo definido previamente. Por eso, las API RESTful son cada vez más frecuentes que las de SOAP. En los últimos años, la especificación de OpenAPI se ha convertido en un estándar común para definir las API de REST. OpenAPI establece una forma independiente del lenguaje para que los desarrolladores diseñen interfaces API de REST, que permite a los usuarios entenderlas con el mínimo esfuerzo. [9]

GraphQL, ¿El reemplazo de API REST?

GraphQL es una tecnología desarrollada por la compañía Facebook inicialmente en el año 2012 de forma interna y que deciden luego publicar para su uso gratuito en el mundo entero en el año 2015. Esta iniciativa se desarrolla debido a la necesidad identificada por la compañía de rediseñar el modelo de búsqueda de datos a una API que no tomara gran esfuerzo de parte del servidor para preparar la data y de parte del cliente para transformarla y adecuarla a un uso. [12]

GraphQL y REST son dos mecanismos arquitectónicos que se encuentran en niveles diferentes, puesto que GraphQL es una tecnología y REST es un estilo arquitectónico. Según Eizinger (2017), para hacer una comparación de estos dos mecanismos existen tres posibles acercamientos:

1) Generalizar GraphQL y subirlo al nivel de un estilo arquitectónico.
2) Especializar REST, es decir, crear una tecnología que siga las restricciones del estilo perfectamente.
3) Utilizar únicamente los principios arquitectónicos de cada mecanismo para su comparación.

GraphQL requiere una interfaz proporcionada por un servidor, el cual debe procesar consultas y devolver un conjunto de datos especificado por el cliente. Tiene como posibilidad usar una puerta de enlace API en la que se puede encapsular el acceso a los sistemas externos [14]. El servicio de consultas es una instancia que se ejecuta por separado del cliente y puede considerarse como un servicio web. El servidor GraphQL permite que el cliente envíe consultas más potentes de lo que sería posible con otro estándar como SOAP o RESTful.

Resultados

En la actualidad no existe aplicación o proyecto que no disponga de una API REST para la generación de servicios a partir de una aplicación. Hay numerosas empresas que gracias a REST y sus API generan negocio.

Organizaciones y empresas como Twitter, Amazon, Facebook, Google, entre muchas otras, trabajan con REST. Actualmente, pequeñas empresas están comenzando a integrarlas en sus plataformas, ya que se han dado cuenta de que les ayudan a incrementar sus beneficios, permitiendo mejorar la calidad y funcionalidad de sus plataformas.

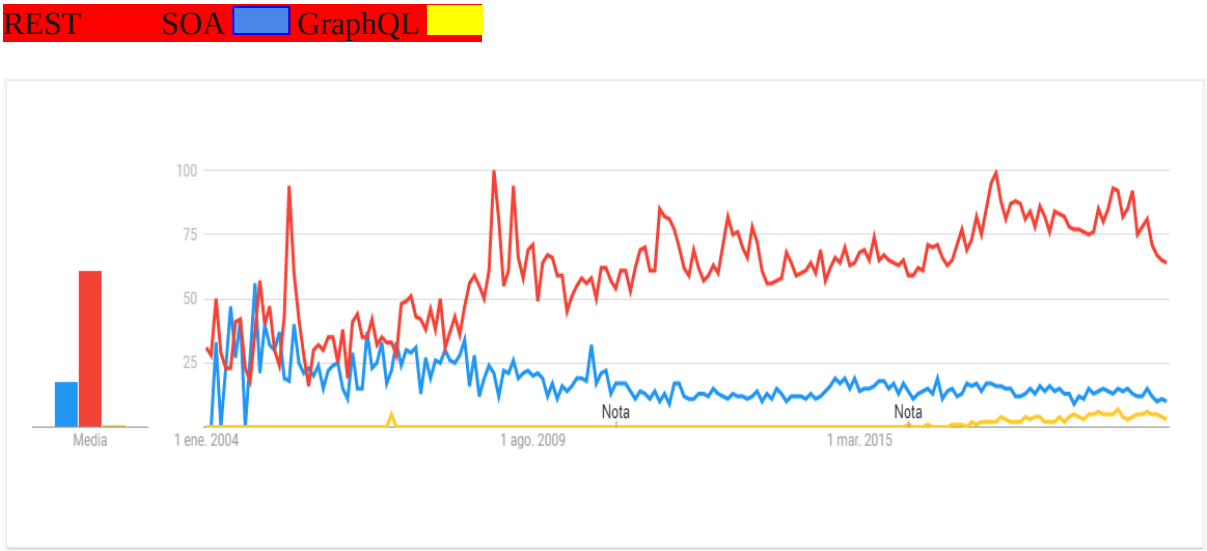
Puesto que, el uso de la arquitectura REST en las empresas, proporcionan múltiples beneficios a la hora de mejorar los procesos de la organización. Sin REST, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de API para servicios de internet.

Ahora bien, en la gráfica 1 se observar la contundencia que tiene REST respeto a SOA y GraphQL, no obstante, podemos decir a favor de GraphQL que es más reciente y por ello tiene menor popularidad. Pero en lo que concierne a SOA que es una arquitectura más antigua que REST y tomando como inicio el año 2004 hasta el día de hoy, objetivamente podemos decir que entre 2004 y 2008 estuvo muy igualado la utilización de REST y SOA, y claramente se observa que aunque REST era reciente y parecía la moda, no lo fue llegó para quedarse y lo ha

demostrado a partir del 2008 cuando se despegó de SOA y las empresas comenzaron a implementar REST a sus aplicaciones y desde aquella fecha REST se muestra y mostró como la mejor alternativa para el desarrollo de aplicaciones empresariales.

No obstante, cabe recalcar que hoy en día mientras REST se sigue manteniendo fuerte, SOA está perdiendo el interés, y más abajo tenemos a GraphQL que va tomando fuerza para la implementación en los desarrollos.

Figura 2. Grafica REST frente a SOA y GrapQL



Fuente: Elaboración propia

Cabe mencionar que REST ha estado tomando fuerza a una velocidad impresionante y más con la llegad de NodeJS y las base datos NoSQL como MongoDB. Sin embargo, con la llegada del internet de las cosas (IOT) cada vez se conectan más dispositivos a internet que necesitan ser integrados, haciendo esto una gran oportunidad para REST y seguir con esa contundencia que se mencionó anteriormente.

Discusión

REST como arquitectura, es muy prometedora y dando a conocer que es más fácil de entender que SOA y no por eso significa que sea menos confiable. Cada uno tiene su ventaja, pero REST muestra cómo debe comportarse una aplicación web bien diseñada.

Como podemos ver, el estilo de arquitectura REST intente ayudar a resolver concretamente el rendimiento, la facilidad de modificación, visibilidad, portabilidad, escalabilidad y simplicidad.

El estilo de arquitectura REST ayuda a resolver problemas que pueden dar dolores de cabeza desde un principio, ya que a una guía de cómo debe comportarse una aplicación para cumplir las cualidades mencionadas.

Conclusiones

Grandes empresas todavía implementan SOA y no invierten en una arquitectura con mayor rendimiento e integración de medidas como REST ya que el cambio trae consigo una serie de modificaciones y tiempo, y una empresa si sabe que hay algo que les funciona no ponen en riesgo su negocio, aun sabiendo que REST a lo largo trae consigo grandes beneficios ya antes mencionados.

La mayoría de clientes prefieren el estándar REST. Existen muchas empresas que han invertido y apostado por SOAP, pero es evidente que los desarrolladores prefieren, en muchos casos, una forma más sencilla de manipulación de datos como lo que ofrece REST.

Al aparecer REST tiene mayor aceptación, por eso va a depender de lo cambiante que resulta la tecnología. Además, es mucho más sencillo encontrar contenido y servicios web que se basen en la arquitectura REST. Aunque implementar un servicio web REST es un poco más fácil que utilizar SOA.

Una implementación siguiendo el estilo arquitectónico REST en una API posee una mayor rapidez de respuesta que una API creada con la tecnología GraphQL. Como se sabe REST impone que debe utilizarse el caché para no hacer peticiones innecesarias al servidor y así aumentar la velocidad de las interacciones entre el cliente y el servidor, mientras que GraphQL como tecnología no ofrece de forma natural este middleware.

No se descarte que en el futuro muy cercano se hable más de GraphQL a diferencia de REST ya que el rendimiento que nos proporciona GraphQL es considerado.

Referencias

Bernal, M. F. C. (2017). *Asegurar web services no es cuestión de costos. Ciencia, Innovación y Tecnología*, 3, 105-109.

Byron, L. (2015) *GraphQL: A data query language*. [Artículo oficial de Facebook acerca de GraphQL]. Recuperado en https://code.facebook.com/posts/1691455094417024/graphql-a-data-querylanguage/ el 07 de junio de 2017.

Delgado, A., González, L., & Piedrabuena, F. (2006). *Desarrollo de aplicaciones con enfoque SOA (Service Oriented Architecture)*. Reportes Técnicos 06-16.

Eizinger, T. (2017) *API Design in Distributed Systems: A Comparison between*

Fowler, M. (2003). *Patterns of enterprise application architecture*. Addison-Wesley.

GraphQL and REST. [Trabajo de Maestría para optar por el título de Gran Maestre de ciencia en ingeniería]. Universidad de ciencias aplicadas Fachhochschule Technikum Wien, Vienna, Austria.

GraphQL. (2016). *Introduction to GraphQL*. Recuperado el 07 de junio de 2020 de https://graphql.org/learn/

GraphQL. (2018). *GraphQL Specification Versions.* Recuperado el 07 de junio de 2020 de https://graphql.github.io/graphql-spec/June2018/.

Krafzig, D. Banke, K. Slama, D. *"Enterprise SOA, Service Oriented Architecture Best Practices"* Prentice Hall, 2005, ISBN 0-13-146575-9.

M. Vera, *Intelligence Bussines*, 2014. [En línea]. Available: http://www.i2btech.com/blog-i2b/tech-deployment/que-se-entiendepor-soa-y-cuales-son-sus-beneficios/.

Ma. Carmen Gastalver Robles (2017). UF1757 - *Información y gestión operativa de la compraventa internacional*, p 331.

P. Bianco, K. Rick y M. Paulo, *Software Engineering Institute*, 2007. [En línea]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8443

Pacheco Laje, J. L. (2018*). Estudio comparativo entre una arquitectura con microservicios y contenedores dockers y una arquitectura tradicional (monolítica) con comprobación aplicativa* (Doctoral dissertation, Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas. Carrera de Ingeniería En Sistemas Computacionales).

Pananya Sripairojthikoon, Twittie Senivongse. *"Concept-Based Readability Measurement and Adjustment for Web Services Descriptions".* ICACT Transactions on Advanced Communications Technology (TACT) Vol. 3, Issue 1, January 2014.

Red Hat. (2020). *Qué son las API y para qué sirven*. Recuperado el 07 de junio de 2020 de https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

Singaravelu, L., Wei, J., y Pu, C. (2008). *A Secure Information Flow Architecture for Web Services. SCC '08 Proceedings of the 2008 IEEE International Conference on Services*, p. 182-189.

WSDL Specification for W3C https://www.w3.org/TR/wsdl.