Centro de Ciencias Básicas

Departamento de Sistemas de Información

Definición de una API para mejorar la interoperabilidad entre las plataformas de sistemas informáticos de los departamentos de estadística y geografía del INEGI

Trabajo Práctico que presenta
Ángel Ricardo Castro Estrada
para optar por el grado de:
Maestría en Informática y Tecnologías Computacionales

TUTORES
MITC Jorge Eduardo Macías Luévano
Dr. Juan Muñoz López

INTEGRANTE DEL COMITÉ TUTORAL
Dra. Laura Arminda Garza González

Aguascalientes a 13 de junio de 2022

CARTA DE VOTO APROBATORIO
INDIVIDUAL

M. EN C. JORGE MARTÍN ALFÉREZ CHÁVEZ
DECANO DEL CENTRO DE CIENCIAS BÁSICAS

_____
P R E S E N T E

Por medio del presente como TUTOR designado del estudiante **ANGEL RICARDO CASTRO ESTRADA** con ID **180657** quien realizó el trabajo práctico titulado: : **DEFINICIÓN DE UNA API PARA MEJORAR LA INTEROPERABILIDAD ENTRE LAS PLATAFORMAS DE SISTEMAS INFORMÁTICOS DE LOS DEPARTAMENTOS DE ESTADÍSTICA Y GEOGRAFÍA DEL INEGI**, un trabajo propio, innovador, relevante e inédito y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia doy mi consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que me permito emitir el **VOTO APROBATORIO**, para que él pueda proceder a imprimirlo así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

A T E N T A M E N T E
"Se Lumen Proferre"
Aguascalientes, Ags., a 13 de junio de 2022

MITC Jorge Eduardo Macías Luévano
Tutor de trabajo práctico
El *nombre completo que aparece en el Voto Aprobatorio debe*
coincidir con el que aparece en el documento empastado. No se puede abreviar, ni omitir nombres

c.c.p.- Interesado
c.c.p.- Secretaría Técnica del Programa de Posgrado

M. EN C. JORGE MARTÍN ALFÉREZ CHÁVEZ
DECANO DEL CENTRO DE CIENCIAS BÁSICAS

P R E S E N T E

Por medio del presente como **Miembros del Comité Tutoral** designado del estudiante **ANGEL RICARDO CASTRO ESTRADA** con ID **180657** quien realizó el trabajo práctico titulado: **DEFINICIÓN DE UNA API PARA MEJORAR LA INTEROPERABILIDAD ENTRE LAS PLATAFORMAS DE SISTEMAS INFORMÁTICOS DE LOS DEPARTAMENTOS DE ESTADÍSTICA Y GEOGRAFÍA DEL INEGI**, un trabajo propio, innovador, relevante e inédito y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia damos nuestro consentimiento de que la versión final del documento ha sido revisada y las correcciones se han incorporado apropiadamente, por lo que nos permitimos emitir el **VOTO APROBATORIO**, para que él pueda proceder a imprimirlo así como continuar con el procedimiento administrativo para la obtención del grado.

Ponemos lo anterior a su digna consideración y sin otro particular por el momento, le enviamos un cordial saludo.

A T E N T A M E N T E
"Se Lumen Proferre"
Aguascalientes, Ags., a 13 de junio de 2022

MITC Jorge Eduardo Macías Luévano
Tutor de trabajo práctico

Dr. Juan Muñoz López
Co-Tutor de trabajo práctico

Dra. Laura Arminda Garza González
Asesor de trabajo práctico

c.c.p.- Interesado
c.c.p.- Secretaría Técnica del Programa de Posgrado

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

DICTAMEN DE LIBERACION ACADEMICA PARA INICIAR LOS TRAMITES DEL EXAMEN DE GRADO

Fecha de dictaminación dd/mm/aaaa: 13/06/2022

NOMBRE: Angel Ricardo Castro Estrada **ID** 180657

PROGRAMA: MAESTRIA EN INFORMATICA Y TECNOLOGIAS COMPUTACIONALES **LGAC (del posgrado):** Gestión de sistemas y tecnologías de información para mejorar competitividad, innovación y cambio organizacional.

TIPO DE TRABAJO: ( ) Tesis (X) Trabajo Práctico

TITULO: Definición de una API para mejorar la interoperabilidad entre las plataformas de sistemas informáticos de los departamentos de estadística y geografía del INEGI

IMPACTO SOCIAL (señalar el impacto logrado): Se aportó un modelo de API para la gestión en los procesos y sistemas informáticos del INEGI, el cual permite una mejor comunicación entre los diferentes sistemas de información de diferentes áreas del INEGI

INDICAR SI NO N.A. (NO APLICA) SEGÚN CORRESPONDA:

	Elementos para la revisión académica del trabajo de tesis o trabajo práctico:
SI	El trabajo es congruente con las LGAC del programa de posgrado
SI	La problemática fue abordada desde un enfoque multidisciplinario
SI	Existe coherencia, continuidad y orden lógico del tema central con cada apartado
SI	Los resultados del trabajo dan respuesta a las preguntas de investigación o a la problemática que aborda
SI	Los resultados presentados en el trabajo son de gran relevancia científica, tecnologíca o profesional según el área
SI	El trabajo demuestra más de una aportación original al conocimiento de su área
SI	Las aportaciones responden a los problemas prioritarios del país
SI	Generó transferecia del conocimiento o tecnológica
SI	Cumple con la ética para la investigación (reporte de la herramienta antiplagio)
	El egresado cumple con lo siguiente:
SI	Cumple con lo señalado por el Reglamento General de Docencia
SI	Cumple con los requisitos señalados en el plan de estudios (créditos curriculares, optativos, actividades complementarias,estancia, predoctoral, etc)
SI	Cuenta con los votos aprobatorios del comité tutoral, en caso de los posgrados profesionales si tiene solo tutorpodrá liberar solo el tutor
SI	Cuenta con la carta de satisfacción del Usuario
SI	Coincide con el título y objetivo registrado
SI	Tiene congruencia con cuerpos académicos
SI	Tiene el CVU del Conacyt actualizado
NO	Tiene el artículo aceptado o publicado y cumple con los requisitos institucionales (en caso que proceda)
	En caso de Tesis por artículos científicos publicados
N.A.	Aceptación o Publicación de los articulos según el nivel del programa
N.A.	El estudiante es el primer autor
N.A.	El autor de correspondencia es el Tutor del Núcleo Académico Básico
N.A.	En los artículos se ven reflejados los objetivos de la tesis, ya que son producto de este trabajo de investigación.
N.A.	Los artículos integran los capítulos de la tesis y se presentan en el idioma en que fueron publicados
N.A.	La aceptación o publicación de los artículos en revistas indexadas de alto impacto

Con base a estos criterios, se autoriza se continúen con los trámites de titulación y programación del examen de grado:

Sí X

No

F I R M A S

Elaboró:

* NOMBRE Y FIRMA DEL CONSEJERO SEGÚN LA LGAC DE ADSCRIPCION: Dr. Cesar Eduardo Velázquez Amador

NOMBRE Y FIRMA DEL SECRETARIO TÉCNICO: MITC Jorge Eduardo Macías Luévano

*** En caso de conflicto de intereses, firmará un revisor miembro del NAB de la LGAC correspondiente distinto al tutor o miembro del comité tutoral, asignado por el Decano**

Revisó:

NOMBRE Y FIRMA DEL SECRETARIO DE INVESTIGACIÓN Y POSGRADO: Dra. Haydee Martínez Rúvalcaba

Autorizó:

NOMBRE Y FIRMA DEL DECANO: M. en C. Jorge Martín Alférez Chávez

Nota: procede el trámite para el Depto. de Apoyo al Posgrado

En cumplimiento con el Art. 105C del Reglamento General de Docencia que a la letra señala entre las funciones del Consejo Académico: .... Cuidar la eficiencia terminal del programa de posgrado y el Art. 105F las funciones del Secretario Técnico, llevar el seguimiento de los alumnos.

AGRADECIMIENTOS

Al MITC Jorge Eduardo Macías Luévano por su orientación y apoyo como mi tutor en el desarrollo de la tesis.

Al Dr. Juan Muñoz López por su ayuda y asesoramiento como cotutor para la realización de este trabajo.

A la Dra. Laura Arminda Garza por su tiempo y asesoramiento en este trabajo.

Al CONACYT por brindarme el sustento para realizar este estudio de postgrado.

A cada uno de mis profesores y compañeros, de los cuáles aprendí variedad de temas y fueron un gran complemento.

A mis amigos Isaac, Shendel, Karla y Sergio que siempre me apoyaron en lo que necesitaba.

DEDICATORIA

Este trabajo está dedicado a mis padres, Ángel y Angélica que siempre me han apoyado de manera incondicional.

A mi novia Mariana que siempre estuvo motivándome en seguir adelante y fue inspiración.

A mis hermanos Bryan, Axel y mi hermana Brenda por confiar siempre en mí.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

RESUMEN

El objetivo del presente trabajo es modelar una API para mejorar la interoperabilidad en la transferencia de información que se produce en el INEGI entre los departamentos de estadística y geografía; para facilitar el manejo de información mediante los diferentes sistemas y así producir nuevos servicios de valor para la institución.

Se define una API REST debido a la escalabilidad, independencia del tipo de plataforma o lenguaje, la sencillez de construcción y la adaptabilidad.

Además, la API comprende una arquitectura jerárquica entre los componentes, donde cada una de estas efectúa una acción distinta para lograr el funcionamiento adecuado con la seguridad necesaria, en autenticación, autorización, servicios y resultados. Asimismo, se toma en cuenta el estándar OpenAPI que permite describir la API de manera que sea sencillo descubrir, comprender e interactuar con los servicios que se definen.

ABSTRACT

The objective of this work is to model an API to improve interoperability in the transfer of information that occurs in the INEGI between the departments of statistics and geography; to facilitate the management of information through the different systems and thus produce new services of value for the institution.

A REST API is defined due to its scalability, platform or language independence, simplicity of construction and adaptability.

In addition, it is a hierarchical architecture between components, where each of these performs a different action to achieve proper operation with the necessary security, authentication, authorization, services and results. Likewise, taking into account the OpenAPI standard that allows describing the API in such a way that it is easy to discover, understand and interact with the services that are defined.

INTRODUCCIÓN

La sociedad, así como las empresas, ya sean micro, pequeñas, medianas o grandes dependen de la tecnología hoy en día, es algo que deben usar para ser competentes en el mercado. Por lo que al usar la tecnología también es importante manejar información, por ejemplo, para la toma de decisiones en la empresa, para tener un historial ya sea de ventas, de productos, etc. Al fin que se va almacenando información hace que una empresa deba tener bien controlado y asegurada esa información.

Es por eso que el uso de una API permite un desarrollo más estructurado en los procesos, actividades y automatización en estos en donde se establecen estándares para la entrada y salida de datos, lo que hace que la implementación de una API con otro sistema sea cómoda para el control de respuestas y para la obtención de errores, con esto se logra que un sistema sea capaz de manejar las situaciones en caso de presentar un fallo y tomar acciones necesarias para la corrección.

En dónde una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. (Red Hat, 2017)

Al desarrollar una API se define como se proporcionará el servicio y como se consumirá, se definen términos de servicio, acuerdos a nivel de servicio como disponibilidad, acuerdos de licencia, precios, soporte, etc. También se definen los formatos de entrada y salida de los datos. (De, 2017, p. 1-2)

La API establecerá métodos y funciones para intercambiar diferentes tipos de información entre departamentos, manejando distintas estructuras de base de datos y archivos. Las diferentes bases de datos proporcionarán la información requerida para compartir la misma entre los departamentos, esta API dispondrá de varios aspectos para que su funcionamiento sea el indicado, con la seguridad requerida y la transferencia de datos exitosa. Como se muestra en la Ilustración 1.

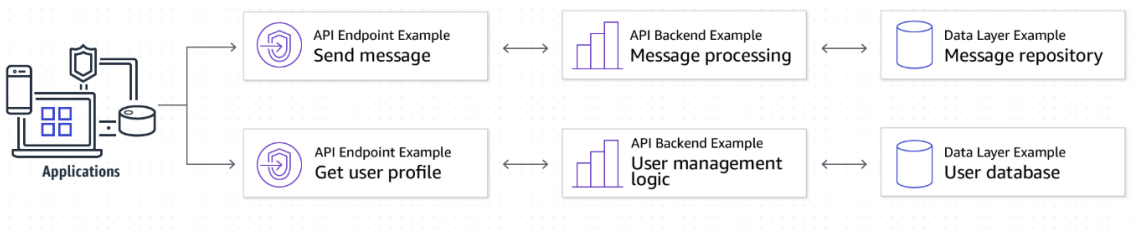


Ilustración 1. Casos de usos de una API (Amazon Web Services, 2019)

El trabajar con una gran cantidad de información permite tener beneficios al realizar un análisis de datos que permitan implementar marketing digital, o implementar un sistema de soporte a la toma de decisiones; en la actualidad las organizaciones manejan grandes

volúmenes de información y/o datos; pero lo importante es que hacen estas organizaciones con estos datos y/o información para obtener el mayor beneficio.

De tal forma que una API posibilita trabajar con extensa variedad de información, desde un prototipo sencillo de una aplicación hasta la administración de instancias de big data.

Como comenta (Abiteboul & Stoyanovich, 2015) la tecnología informática tiene un tremendo poder, y con ese poder viene una inmensa responsabilidad. En ninguna parte es más evidente la necesidad de controlar el poder y utilizar juiciosamente la tecnología que en el análisis de datos masivos, conocido como big data.

Existen una gran variedad de APIs en todo el entorno tecnológico, ya sea para realizar pagos en línea (PayPal, Mercado Pago, Bancos, etc.), obtener información sobres mapas, datos sobre las redes sociales (Twitter, Facebook), timbrado de facturas, para realizar una conexión entre diferentes aplicaciones, por ejemplo, un enlace de un *e-commerce* con una empresa de transporte de paqueterías (DHL, FedEx, Estafeta, etc.). Por lo que a medida que van evolucionando las APIs, estás van optando con un diseño más simple y una sencilla implementación.

Alternativa de desarrollo

Una alternativa para el desarrollo de una API es GraphQL, el cual es un lenguaje de consulta para una API, que maneja una ejecución de consultas mediante el uso de un sistema de tipos de datos.

GraphQL [es] un lenguaje de consulta creado por Facebook en 2012 para describir las capacidades y requisitos de los modelos de datos para aplicaciones cliente-servidor. (Frisendal, 2018). Al tener una significativa cantidad de años en el mercado tecnológico da una idea de que en cualquier momento puede ser que varias empresas opten por esta tecnología o no se use de forma importante.

GraphQL no está vinculado a ninguna base de datos o motor de almacenamiento específico, sino que está respaldado por su código y datos existentes. (*Introduction to GraphQL | GraphQL*, n.d.)

Una ventaja que tiene GraphQL es la optimización, existe una mejor velocidad, ofrece la posibilidad de brindar un servicio escalable y de alta disponibilidad. Aunque cuenta con algunas desventajas, la cuál es que GraphQL no cuenta con soporte para caché, ocupa mayor procesamiento en el servidor, se debe desarrollar un control de errores más detallado ya que los servicios siempre retornan un código de respuesta 200 (OK), GraphQL es una tecnología que se está empezando a usar en las empresas por lo tanto para un óptimo mantenimiento, fiabilidad y escalabilidad en los servicios es preferente realizar una API.

Seguridad

Una API puede ser muy sencilla, como la obtención del tipo de cambio de una divisa, pero a la vez puede ser tan compleja donde existe una seguridad (SSL, en inglés *Secure Socket*

Layer) en todos lados, una autenticación, procesos y algoritmos profundos, códigos de respuesta, control de versiones, manejo de excepciones y/o errores, y más características que la hacen ser completa en funcionamiento.

Las APIs están en todas partes, se usan todos los días y ni siquiera nos damos cuenta. El uso de las APIs a gran escala generan un alto riesgo de un ataque relacionado a estas. Pero del mismo modo que otras amenazas cibernéticas, los efectos de una infracción dependen de un escenario específico y de los datos que se comparten a través de la API. (Macy, 2018)

Se aborda el tema de seguridad debido a que es una característica obligatoria con la que se debe trabajar en una API. La seguridad se debe considerar a partir de la fase de diseño hasta la implementación, ya que las APIs se ven amenazadas a distintos ataques que puedan llegar a afectarla de manera significativa. Como comenta (Vijayakumar, 2018a), la seguridad no se trata solo de autenticación y autorización, sino también sobre qué datos están expuestos en qué contrato de servicio y cómo los consumidores consumen los servicios.

La definición de una API debe considerar características que la hagan segura, ya que la información sensible es valiosa para toda organización, es por eso qué al tener un intercambio de datos entre diferentes métodos definidos en la API es necesario contar con algún tipo de cifrado o encriptación. La forma en que se aborde la seguridad de las API dependerá del tipo de datos que se transfiera. (Red Hat, 2019b) Por ejemplo, como se muestra en la Ilustración 2.

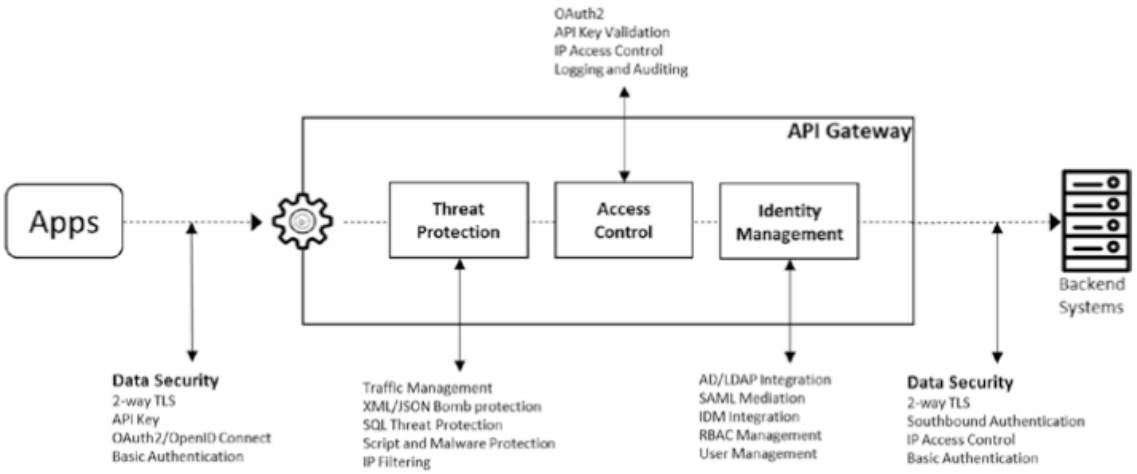


Ilustración 2. Enfoque de seguridad en una API. (De, 2017, p. 142)

Manejo de versiones

Una API nunca será completamente estable, los cambios en ella son inevitables, lo primordial en ellos es como gestionar estos cambios, el anunciar fechas de modificaciones o cambios y la planificación correctamente documentada es una buena práctica.

El control de versiones es una de las consideraciones más importantes al diseñar su API web. Nunca lance una API sin una versión y haga que la versión sea obligatoria. (Mulloy, 2012)

El manejo de versiones en las API ayuda a seguir ofreciendo el servicio desde puntos finales (en inglés endpoints). Al tener versiones, la API va tomando actualizaciones, por lo que en ciertos casos es necesario analizar si es óptimo llevar a cabo un control de versiones debido a que habrá cambios, ya sea en seguridad, en desarrollo de nuevos servicios, correcciones o simplemente detalles en los procesos. Como se comenta en (Yii Framework, 2020), los cambios y nuevas características son implementadas en las nuevas versiones del API, en vez de estar continuamente modificando solo una versión.

Diseño

Se definen las funcionalidades que contendrá la API, así como los tipos de datos que se utilizarán para consumir los servicios. Este diseño puede ser simple sin embargo puede resultar útil para un trabajo en específico pero muy complejo para otro, por lo que resulta útil pensar en la simplicidad en distintos niveles, por ejemplo:

- Formato de datos
- Autenticación
- Autorización
- Políticas de uso

Las API son herramientas para los departamentos de TI, son elementos estratégicos que se convierten en la innovación para muchas empresas. Es por eso que la estrategia API first impulsa primero a diseñar, documentar y generar especificaciones de la API, la cual da la posibilidad de contrastar esta información con las partes involucradas.

El diseño previo de la API requiere el uso de herramientas que cuenten con el modelamiento de esta estrategia, API Blueprint, RAML y Swagger son herramientas para diseñar APIs en donde es posible interactuar con la documentación antes de escribir código, lo que conduce a la especificación OpenAPI. Esto permite testear y solucionar incongruencias en el diseño lo que lleva a obtener un producto final con menor porcentaje de error y mayor escalabilidad.

Desarrollo

Tomando en cuenta el diseño prosigue el desarrollo, en donde es poner en marcha lo establecido en ese diseño y con el uso de la especificación OpenAPI y la estrategia API first comienza el ciclo de desarrollo, creando la estructura a partir del modelado en la herramienta de diseño (Swagger).

El desarrollo de una API al inicio es fácil, pero a medida que va creciendo, ya sean en servicios, en versionamientos, seguridad, etc. se vuelve un poco más complejo (Ilustración 3).

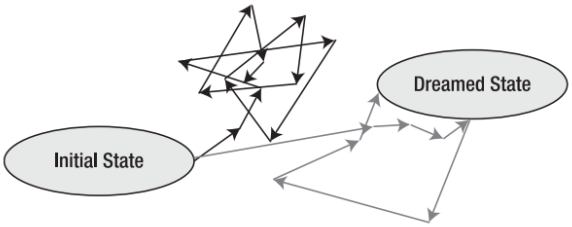


Ilustración 3. Desarrollo de una API. (Tulach, 2008, p. 189)

Documentación

Para llevar a cabo una implementación eficaz de API debe existir una buena documentación, en donde el funcionamiento de la API esté claramente escrito para evitar confusiones; la mayoría de los desarrolladores examinan la documentación antes de realizar la integración. En esta documentación se debe especificar cómo serán las peticiones y las respuestas que puede recibir de esta, lo que origina un entendible conocimiento en los servicios que ofrece la API.

En el documento se debe mostrar ejemplos de ciclos completos de petición y respuesta, esto para guiar adecuadamente al personal que la implementará.

Una buena documentación se podría decir que es aquella en donde exista una categorización de los servicios, la forma de autenticación, una tabla con los códigos de respuesta para el control de errores, ejemplos de peticiones y respuestas, el detallar los tipos de datos para cada servicio, cuales campos son requeridos y que es lo que puede devolver en la respuesta. El tener detallado correctamente cuál es la función del servicio, ser claros y concisos en la documentación para que las personas que trabajarán con la API no tengan algún problema. Como señala (Mihaly, 2011), debido a que los desarrolladores revisan la documentación, cada sección debe enfocarse en un solo tema y resaltar cuál es el tema.

En los servicios de la API es conveniente mostrar ejemplos de la petición y respuesta, los cuáles facilitan a comprender mejor el uso del servicio, así pues, los ejemplos de código son un recurso importante tanto para el aprendizaje inicial como para trabajar hacia la solución de un problema. (Meng et al., 2019)

Normalmente se usan los códigos de estatus 2xx y 3xx como códigos de respuesta correctos y los 4xx y 5xx como códigos de error (Ilustración 4).

HTTP	RPC	Descripción
200	OK	No hay errores.
400	INVALID_ ARGUMENT	El cliente especificó un argumento no válido. Verifica el mensaje de error y los detalles de errores para obtener más información.
400	FAILED_ PRECONDITION	La solicitud no se puede ejecutar en el estado actual del sistema, como borrar un directorio que no esté vacío.
400	OUT_OF_RANGE	El cliente especificó un rango no válido.
401	UNAUTHENTICATED	La solicitud no se autenticó debido a que el token de OAuth no es válido, falta o se venció.
403	PERMISSION_ DENIED	El cliente no cuenta con los permisos necesarios. Esto puede suceder porque el token de OAuth no tiene los alcances correctos, el cliente no tiene permiso o la API no se habilitó para el proyecto del cliente.
404	NOT_FOUND	No se encuentra un recurso específico o la solicitud se rechazó por razones no reveladas, como la lista blanca.
409	ABORTED	Conflicto de simultaneidad, como conflicto del proceso de lectura, modificación y escritura.
409	ALREADY_EXISTS	El recurso que el cliente intentó crear ya existe.
429	RESOURCE_ EXHAUSTED	Sin cuota de recursos o a punto de alcanzar el límite de frecuencia. Para obtener más información, el cliente debe buscar el detalle de error google.rpc.QuotaFailure.
499	CANCELLED	El cliente canceló la solicitud.
500	DATA_LOSS	Daño o pérdida de datos no recuperable. El cliente debe informar el error al usuario.
500	UNKNOWN	Error de servidor desconocido. Por lo general, un error de servidor.
500	INTERNAL	Error del servidor interno Por lo general, un error de servidor.
501	NOT_IMPLEMENTED	El servidor no implementó el método de API.
503	UNAVAILABLE	Servicio no disponible. Por lo general, el servidor no está en funcionamiento.
504	DEADLINE_ EXCEEDED	Se excedió el plazo de la solicitud. Esto ocurrirá solo si el emisor establece una fecha límite más corta que la fecha límite predeterminada del método (es decir, la fecha límite solicitada no es suficiente para que el servidor procese la solicitud) y la solicitud no finalizó dentro de la fecha límite.

Ilustración 4. Ejemplo de códigos de estatus (Google Cloud, 2021b)

La documentación de la API debe proporcionar conocimientos básicos para facilitar la entrada en una API para desarrolladores sin experiencia previa en el dominio cubierto por la API. (Meng et al., 2019)

Como señala (Henning, n.d.) las APIs deben documentarse antes de implementarse. Un gran problema con la documentación de la API es que generalmente se escribe después de la implementación de la API.

Información

El procesamiento de la información dentro de una API es de gran interés para el creador, pero aún más importante para el cliente. La velocidad del proceso es algo que las organizaciones buscan, en que los tiempos de trabajo sean los óptimos para obtener lo deseado.

Los procesos que se manejan dentro de una API deben ser lo más eficientes posibles y sin error alguno, para que los consumidores/clientes obtengan el resultado esperado, esto es primordial para las API en las cuales hay conexión entre empresas.

El desarrollo de APIs ha ido en aumento para las funcionalidades del negocio. Las APIs también son la base para construir canales de comunicación en el internet de las cosas. (Siriwardena, 2014)

La interfaz de programación de aplicación (API, en inglés *Application Programming Interface*), es la base de la revolución de la nube, los dispositivos móviles y el internet de las cosas (IoT, en inglés *Internet of Things*). (Ashby & Jensen, 2018)

Cualquier cliente debe poder llamar a la API, con independencia de cómo esté implementada internamente. Para ello, es necesario usar protocolos estándar y contar con un mecanismo por medio del cual el cliente y el servicio web puedan acordar el formato de los datos que se intercambian. (Masashi Narumoto, 2018)

Así pues, las API son importantes para toda empresa hoy en día, ya sea para tener centralizados los procesos de un servicio, o en este caso el estandarizar el envío de documentos entre departamentos, facilitando la distribución y descarga de estos.

Cada empresa cuenta con objetivos a cumplir, con la ayuda de la tecnología, el contar con una API para optimizar los procesos es sustancial, cada una de estas empresas define la arquitectura de acuerdo a sus requerimientos, ya sea, con arquitectura SOAP, REST, Extensible Markup Language - Remote Procedure Call (XML-RPC) o JavaScript Object Notation – Remote Procedure Call, y con un formato XML o JSON. Por lo tanto, el diseño de la API es una de las partes más importantes ya que es la base donde se centra el desarrollo y el funcionamiento.

Áreas de oportunidad

En empresas y organizaciones sobre todo de tamaño grandes, la distribución de la información puede llegar a ser tardada, incluso en un escenario grave, se puede perder y no contar más con ella.

Con una API de intercambio de documentos se pretende simplificar el trabajo de los usuarios, además de que también facilita el trabajo a los desarrolladores. Este tipo de tecnología complementado con la transferencia de documentos es de gran valor para empresas locales, empresas nacionales, e incluso empresas internacionales.

Dado que la información en una empresa es de suma relevancia, esta API mejoraría las operaciones entre departamentos, optimizando la distribución de documentos, además de permitir una reducción de tiempo y costes de mantenimiento en cuánto al software, generar una estructuración en los procesos y datos del negocio. Asimismo, la seguridad de la información para cualquier empresa es un tema valioso, por lo que, las tecnologías informáticas deben contar con este requisito fundamental.

Una API puede aportar nuevas oportunidades a la empresa u organización, en este caso de la distribución de información entre departamentos, esta puede expandirse y lograr que alguna información pública de la empresa pueda consumirse por terceros, creando una API pública y así distribuirla de una manera más práctica.

PLANTEAMIENTO DEL PROBLEMA

El uso de la información en las empresas es fundamental, es un recurso vital y el manejo correcto de esta hace que una empresa obtenga éxito en sus proyectos. La definición de una API para transferir información contribuye a un control deseable de información en una empresa en las distintas áreas de las que esta dispone.

Objetivo general

Modelar una API para mejorar la interoperabilidad en la transferencia de información que se produce en el INEGI entre las áreas de estadística y geografía; para facilitar el manejo de información mediante los diferentes sistemas y así producir nuevos servicios de valor para la institución.

Objetivos específicos

1. Determinar una arquitectura de API escalable que permita la evolución para generar nuevos servicios que requiera la institución.
2. Establecer una solución para facilitar el manejo de la información en la institución entre sus áreas.
3. Determinar si la API propuesta para la institución es funcional a través de sus distintos sistemas de TI.

Preguntas de investigación

1. ¿Cómo será la funcionalidad de los métodos de alto nivel que contendrá la API para la distribución de la información?
2. ¿Mejorará la accesibilidad, los tiempos de entrega, la seguridad y la fiabilidad en la distribución de la información entre los distintos departamentos de la institución?
3. ¿Cómo influirá la API dentro de los sistemas de TI con los que cuenta la institución?

Justificación

La definición y creación de una API es un proceso esencial en las empresas y organizaciones actualmente, este tipo de software proporciona un conjunto de funciones o métodos donde estos pueden ser utilizados por otro tipo de sistemas y programas.

Una API permite definir servicios que logran una comunicación sin necesidad de utilizar el mismo lenguaje de programación, con lo que las APIs disponen de una escalabilidad para la implementación de nuevos servicios, siendo capaz de tolerar picos grandes de carga. Además, el contar con una API ayuda a reducir la carga de trabajo para el desarrollo en el área de TI, además de lograr una automatización y estandarización de procesos en una empresa.

La API para transferencia de información es una tecnología no muy usual pero muy importante para las empresas y para los usuarios de estas, al agregar una API de este tipo,

mejorará la relación entre distintas áreas o departamentos para poder obtener información de manera más ágil, siendo que la API permite conectarse con diferentes sistemas.

FUNDAMENTACIÓN TEÓRICA

Las aplicaciones tecnológicas han evolucionado de manera rápida con la sociedad y las organizaciones. La creación de sistemas informáticos para las empresas es una necesidad importante para manejar la información, por lo tanto, estas aplicaciones de software se conectan con las áreas funcionales de la organización llevando a cabo sus actividades con los diferentes servicios. Dentro de las organizaciones el contar con un control eficiente y eficaz de la información es valioso, así pues, los sistemas informáticos contribuyen a disponer de los datos necesarios para gestionar y comunicar las tareas a realizar en cada área.

Las sistemas informáticos cuentan con distintas funcionalidades y están relacionados a la organización que lo emplea, de modo que los desarrollos de estos en cualquier organización manejan una ingeniería de software, ya sea de poca o a gran escala, lo que conlleva a una comprensión de estandarizar procesos y/o servicios de una manera centralizada que sirva para la comunicación de la(s) base(s) de datos, protocolos de comunicación y ejecución de procedimientos para cumplir un determinado servicio, lo que suscita a un desarrollo de una API.

API

Una API es un conjunto de protocolos y servicios que se procesan en un sistema centralizado que realizan ciertos procedimientos definidos para la organización, con el cual un software secundario o terciario logra comunicarse con esta para solicitar un servicio propio de la API con una respectiva estructura en la petición. Los servicios que ofrecen las API dependen del modelo de negocio de la organización y va de la mano con el objetivo para la definición de esta.

Las API ayudan a mejorar la funcionalidad de los sistemas informáticos, mantienen una estructura definida que cumplen protocolos de autenticación, comunicación y gestión a través de un middleware.

Una API ayuda a exponer un servicio comercial o un activo empresarial a los desarrolladores que crean una aplicación. (De, 2017)

Una API está diseñada para que los servicios con los que cuenta sean consumidos a través de la red por distintos softwares (Ilustración 5), no hay relevancia en qué tipo de lenguaje de programación se encuentre ya que las API generan los procesos internos, los consumidores o los clientes solo realizan la petición al servicio requerido, la API procesa las peticiones a través de su estructura y al finalizar devuelve la respuesta a través de un formato prestablecido como JSON.

El uso de API en cualquier organización, empresa y negocio proporciona múltiples beneficios cuando se trata de mejorar los procesos internos de la empresa. Existen distintos motivos por lo que es provechoso integrar una API a la empresa:

- Brindan apoyo a nuevos proyectos
- Reestructuran y conducen procesos internos
- Se crean nuevas oportunidades de crecimiento al crear nuevos servicios
- Escalabilidad de los procesos que integran la API
- Facilidad de integración a los consumidores
- Ofrece una personalización por servicio
- Automatización en procesos internos de la empresa

Una API ofrece una forma sencilla de conectarse, integrarse y ampliar sistemas de software. (Biehl, 2015)

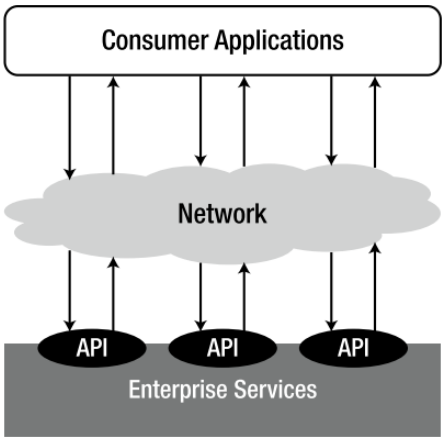


Ilustración 5. Una API provee una interfaz para que las aplicaciones de los consumidores interactúen con los servicios de las organizaciones (De, 2017)

Tipos de API

Las API pueden encontrarse de diferentes formas, es decir, con distintos protocolos y funcionalidades, el tipo de API se encuentra sujeta a las necesidades de las organizaciones. Estas se mencionan a continuación:

- API abierta
  Este tipo de API se encuentran disponibles para cualquier usuario, esto es que es pública y no cuenta con restricciones.

- API interna
  Son API privadas de las organizaciones, están diseñadas para un uso exclusivo dentro de las organizaciones. Cuentan con un control de seguridad y acceso que permite a diferentes sistemas y equipos consumir los servicios.

- API de socios
  Este tipo de API es similar a las API abiertas, solo que cuentan con un acceso restringido y para el consumo de esta se requiere un derecho o licencia, dicho de otro modo, para el acceso a están es un servicio pagado por lo tanto son para usuarios específicos.

- API compuesta
  Este tipo de API soporta distintos sistemas de integración, por lo que son ideales para el manejo de microservicios, los cuales necesitan varios servicios para realizar una operación.

Como la información, estandarización de procesos y contar con una seguridad adecuado es algo primordial para toda organización, una API interna es idónea.

Este tipo de API interna conectará el sistema interno o sistemas de la organización con los procesos internos que proporciona la API con el fin de reducir el trabajo de desarrollo y aumentar la colaboración entre los distintos departamento o áreas.

Arquitectura

Para consumir los servicios de una API existen un conjunto de protocolos los cuales determinan el formato de datos que manejan para las peticiones y respuestas, los cuales son:

- SOAP
  El protocolo simple de acceso a objetos que utiliza XML como formato para la transferencia de datos que envía y recibe un consumidor de la API. Se encuentra en conjunto con el lenguaje de descripción de servicios web (WSDL, en inglés *Web Services Definition Language*) para la conexión entre el cliente y la API. Cuando se trata de interfaces de programación de aplicaciones (API), una API SOAP se desarrolla de una manera más estructurada y formalizada. (Kin Lane, 2020)

- XML-RPC
  Este protocolo de llamada de procedimiento remoto (RPC) utiliza *Extensible Market Language* (XML), un XML específico para transferir datos, además usa un ancho de banda bajo y es más simple que SOAP.

- JSON-RPC
  Protocolo de llamada de procedimiento remoto (RPC) con *JavaScript Object Notation* (JSON), es similar a XML-RPC, pero en lugar de usar formato XML para transferir datos usa JSON.

- REST
  Transferencia de estado representacional, es una arquitectura web no un protocolo como los otros. El servicio REST cuenta con ciertas características, como: arquitectura cliente-servidor, caché, sistema en capas y sin estado (sesión).

Una API SOAP ha sido muy elegida en las organizaciones, pero al paso del tiempo la información también va evolucionando y resulta complejo y poco flexible seguir usando este tipo de API, las API REST son la mejor opción, dado que se centran en mejorar la escalabilidad y el rendimiento.

Así mismo, el formato de texto JSON es sumamente simple, la velocidad de procesamiento es alta a comparación con el formato XML que es muy estructurado, cuenta con una estructura jerárquica; lo que da pauta a usar un formato JSON ya que la respuesta rápida y flexibilidad en los datos permite un crecimiento dentro de la API.

Al consumir los servicios de una API REST, el lenguaje de programación es agnóstico, solo depende del formato que estarán las peticiones del cliente, en este caso se trabajará con un formato tipo JSON por lo que es muy fácil la integración para cualquier otro sistema y no requiere mucho esfuerzo.

Una arquitectura API típica incluye seis aspectos clave. (Vijayakumar, 2018b). (Ilustración 6).

Ilustración 6. Aspectos de la arquitectura en una API. (Vijayakumar, 2018b)

Cada uno de los aspectos de la arquitectura en una API que se presentan en la ilustración 6 son relevantes, su desarrollo a profundidad depende del entorno del negocio. A continuación, se describe cada uno de ellos:

- Rendimiento: La API debe permitir un sistema de almacenamiento en caché, así con esto se logra un aumento en la cantidad de respuestas atendidas por unidad de tiempo. También se espera que una API tenga una gran disponibilidad en todos sus servicios.
- Seguridad: Este aspecto es esencial, ya que radica en la prevención de incidencias, puesto que la información debe ser protegida de forma adecuada y al mismo tiempo se considera mantener la disponibilidad de la API llevando a cabo un proceso para la seguridad de la información y servicios de la API.
- Experiencia del desarrollador: En la implementación, el desarrollador es la persona el cual trabajará con la API, por lo tanto, es primordial que existan herramientas de apoyo como una buena documentación, ejemplos, foros, etc. para lograr una exitoso desarrollo.
- Integración: Se refiere a que una API permite la conexión entre aplicaciones, datos y dispositivos.
- Manejo de errores: Establece como la API responderá a diferentes tipos de casos y/o incidentes. En la documentación se debe especificar las distintas respuestas posibles de los sistemas para determinar los comportamientos y acciones a realizar.

- Uso y telemetría: Una API es diferente para cada organización, o las organizaciones pueden tener varias API y cada una tendrá un uso diferente, sus funciones podrán utilizar telemetría, la telemetría es el proceso de recopilación automática de mediciones periódicas de dispositivos remotos. (Google Cloud, 2021a). El monitoreo de una API es valioso puesto que ayudaría a un mejoramiento a futuro de esta misma.

Métodos HTTP

Las API REST realizan principalmente operaciones CRUD por sus siglas en inglés, lo que corresponde a *Create* (Crear), *Read* (Leer), *Update* (Actualizar) y *Delete* (Eliminar). El protocolo de transferencia de hipertexto (HTTP, en inglés *Hypertext Transfer Protocol*) especifica distintos métodos de petición para realizar una acción, los cuáles se muestran en la Tabla 1.

Método HTTP	*Acción*
GET	Obtener o leer un recurso específico
PUT	Actualización de un recurso destino
DELETE	Borra un recurso especificado
POST	Enviar una entidad a un recurso específico

Tabla 1. Métodos HTTP en API REST

Al realizar las peticiones de los servicios que contiene la API, a través del Localizador de Recursos Uniforme (URL, en inglés *Uniform Resource Locator*), deberán llevar la estructura de la petición definida, esto para contar con la autenticación y protocolos definidos por los servicios de la API.

Encabezados

Los encabezados o *headers* son los parámetros que contienen la información esencial sobre la transmisión de los recursos de cliente a servidor, por esta razón los encabezados HTTP son una parte importante en la solicitud y respuesta de la API.

Los HTTP *headers* son la parte central de las peticiones y respuestas HTTP, son esquemas de llave:valor que contienen metadatos que transmiten información acerca del servidor del cliente y del navegador.

Estos *headers* van de la mano con los métodos HTTP, ya que muestran información específica como la información de los datos enviados, donde se muestra la longitud; el tipo de método HTTP; el código de respuesta; el tipo de contenido; tipo de compresión, etc.

Middleware

Es un software que gestiona los datos, la comunicación y la autenticación de la API. El middleware ayuda en la autenticación, verifica que la petición a los servicios cuente con un usuario validado. Es decir, manejan las peticiones y respuestas, sobre un flujo de control y seguridad.

Al obtener una petición al servicio de la API REST, el middleware es el primero en entrar en acción, realiza la validación del usuario, la validación del token de seguridad dentro de la petición, además de aprobar los permisos del usuario sobre los servicios con los que cuenta la API, también se encarga de revisar que los tipos de datos sobre la petición sean los definidos para evitar errores internos en la API. Aprobando el middleware sigue la petición del servicio y procesa el requerimiento del consumidor, devolviendo la respuesta en el formato prestablecido (JSON).

El uso de un middleware está relacionado con la seguridad, apoya en la protección de ataques, este servicio restringe el acceso a las rutas a los usuarios no identificados, es decir, el middleware es un diseño para manejar los cuidados transversales de la API REST, como el manejo de autenticación y autorización. Incluyendo la funcionalidad del middleware, los procesos de la API se centrarán solo en la lógica del negocio, haciendo un control de los servicios más eficaces y fáciles de mantener.

Aunque el middleware no garantiza un rendimiento ideal para cualquier lógica empresarial, es particularmente importante definir los procesos que se manejan y el objetivo de la API para conocer si es conveniente anexar este servicio o refactorizar la lógica del middleware, sin embargo, el middleware aumenta la seguridad interna obteniendo un óptimo control sobre las validaciones antes de procesar los servicios de la API.

Autenticación

La autenticación dentro de la API es el proceso en el cual se comprueba la autenticidad del cliente para consumir los servicios.

Para hacer uso de una API REST como cliente, primeramente, se realiza una estructura de petición que sea admitida por la API REST, el tipo de estructura y formato es proporcionada en la documentación de esta. En esta estructura puede llevar un apartado para la autenticación o no, esto depende del tipo de API REST con la que se trabaje, por lo regular las APIs REST públicas no cuentan con un requerimiento de datos para la autenticación, esto es debido a como el nombre del tipo de API lo indica, son públicas y en consecuencia cualquier persona que solicite el servicio de esta lo podrá consumir.

Para las demás tipos de APIs REST, las privadas, de socios e internas, estás sí hacen uso de una estructura con los datos para la autenticación, la estructura puede contener distintos tipos de datos, como un usuario y contraseña, y también puede hacer uso de un token para mantener una mayor seguridad en el uso de los servicios que contiene la API REST, en la Ilustración 7 se muestra un ejemplo de los datos para la autenticación con formato JSON.

Ilustración 7. Ejemplo de petición a API REST con datos de autenticación y token en formato JSON

La autenticación a utilizar dependerá de la importancia de la información para la empresa u organización, si la información que se utilizará en los servicios de la API son datos sensibles o críticos siempre es fundamental contar con un método de autenticación para cualquier servicio, ya que como se mencionaba antes, la información es lo más importante y no es posible dejar el acceso a los datos abierto a cualquier persona, solo estos datos pueden ser compartidos con las personas y/o sistemas autorizados.

Token

Un token es una firma cifrada que permite a la API identificar al usuario, el token sirve como un tipo de autenticación, debido a que las API REST no cuentan con sesiones, es decir no manejan estado, por lo tanto, el token es de gran ayuda para certificar que el usuario se encuentre autenticado.

Existen distintos tipos de tokens y cada tipo cuenta con una funcionalidad específica, a continuación, se detallan:

- Token de datos
  Realiza el intercambio de datos entre el cliente y el servidor, contiene los datos de la petición al servicio; para este tipo de token existe el JSON Web Token (JWT), el cual es un token estandarizado en el RFC 7519 que permite el intercambio seguro de datos entre el cliente y servidor; este token JWT es seguro para cifrar mensajes cortos, de tal manera que los servicios dentro de la API verifica si cuenta con los derechos de acceso requeridos.
- Token de identificación
  Gestiona la identidad del cliente, permite a la aplicación obtener los datos del cliente sin tener que gestionar las credenciales de autenticación. Este token sirve como el método de autenticación a la API.
- Token de acceso (*bearer token*)
  Este token es emitido por un servidor de autorización, es obtenido a través de un servicio de la API y este token es usado para acceder a los recursos de los servicios,

sirve como un método de autenticación y autorización por parte de la API. Así mismo, este tipo de token puede configurarse con un tiempo de vida, es decir que caduque después de un tiempo definido y así mantener una seguridad en caso de que el token pueda ser interceptado por alguna persona o sistema no autorizado.

La elección de alguno de estos tipos de token recae en cómo será la funcionalidad de autenticación dentro de la API REST, el token debe contar con un algoritmo criptográfico fuerte, y en este caso de trabajo el token debe ser apto para cualquier tipo de servicio que cuente la API. La autenticación por token es hecha por parte del servidor, así pues, la generación del token depende de las tecnologías que se usan dentro del sistema (backend).

De tal manera para la funcionalidad de este trabajo, el token de acceso es el ideal en vista de que este tipo de token da una mayor seguridad para controlar los accesos a toda la API o a ciertos servicios dentro de esta, también de que podemos controlar la vigencia de este token. En la Ilustración 8, se muestra el flujo para la obtención del token.

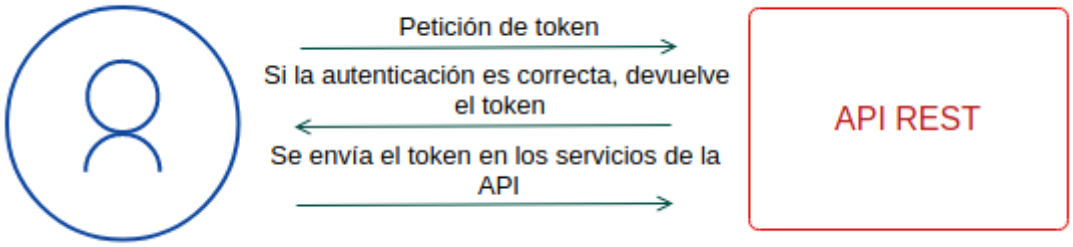


Ilustración 8. Flujo de petición de token a API REST

Este tipo de autenticación basada en token de acceso es conveniente dado que las APIs REST no manejan estado, el token mantiene un tipo de estado de sesión y habilita el consumo de los servicios, este token es enviado a los demás servicios de la API para mantener una autenticación y autorización a estos. Si no es enviado el token o se encuentra fuera de vigencia la API REST devolverá como respuesta un mensaje de que el token de seguridad no se encuentra actualizado y no es posible acceder al servicio requerido.

Por ello es crucial mantener un token vigente para que los servicios consumidos devuelvan lo que se pretende y con eso el cliente pueda continuar con su flujo de procesos sin algún retraso.

Encriptación

También llamado cifrado, es el proceso de transformar datos en texto sin formato y hacerlos ilegibles para todos, excepto para aquellos que deben leer los datos, con el objetivo de mantener la confidencialidad. (Lakshmiraghavan, 2013)

La encriptación permite que la información se mantenga segura y privada, este proceso se realiza a través de un algoritmo el cual cifra los datos de forma que preserve una confianza

para utilizar una encriptación. Considerar una encriptación cuando se habla de seguridad no es algo para dudar, debido a que en caso de que alguna información que se transmita sea interceptada, los datos se encontrarán encriptados, lo que quiere decir que no es información legible y solo con la llave o el método de desencriptar se podrán recuperar los datos a como estaban originalmente. Por consiguiente, el descifrado da pauta a regresar a su forma normal los datos, es decir, restaurar el texto y así el receptor pueda manejar los datos y/o información recibidos.

El beneficio de cifrar los datos es proteger la privacidad de cualquier usuario o empresa, además que evitar una gran vulnerabilidad como el robo de información.

Permisos

Como especifica la RAE, es una licencia o consentimiento para hacer o decir algo (Real Academia Española, 2020)

Una API puede establecer permisos para acceder a sus servicios, estos permisos apoyan con un control sobre las acciones que el cliente puede realizar con la API.
De modo que, para poseer una administración eficiente sobre los permisos, es ideal generar roles de usuarios, donde los usuarios o clientes se agrupan en roles en función de su característica.
Cada rol tiene permisos predeterminados sobre qué es lo que puede realizar, los roles definen la habilidad de lo que el cliente tiene permitido llevar a cabo. Existirán una variedad de roles distintos, estos se generan de acuerdo a los servicios que se tengan y como se desea administrar.

API Gateway

También llamado puerta de enlace de API, es un término que maneja las solicitudes de los usuarios, realiza una función de enrutamiento hacia los servicios lo cual permite un control exhaustivo que brinda una seguridad, puesto que administra el control de acceso, la gestión de llamadas a la API, análisis y supervisión del flujo de los procesos, políticas y alertas de seguridad.

Como se comenta en el sitio web de Red Hat, una puerta de enlace de API es una herramienta de gestión de API que se encuentra entre el cliente y un conjunto de servicios de backend. (Red Hat, 2019a)

Esta herramienta proporciona una amplia utilidad en la administración de las solicitudes a las API, engloba distintas funcionalidades en las cuales el uso del middleware se encuentre implícito y se hace de este un uso significativo para la autenticación y acceso de los usuarios, también el uso del token es considerado, el filtrado de IP (*Internet Protocol*), un límite de solicitudes y es posible realizar una integración con un firewall de aplicaciones web (WAF, en

inglés *Web Application Firewall*), todo con la finalidad de contar con un acceso confiable y seguro en la API.

Las puertas de enlace API están ubicadas de manera que todo el tráfico entrante y saliente pueda ser inspeccionado por ellas. Aplicar reglas de seguridad, implementar requisitos de registro, habilitar revisores, todo esto es compatible. (Preibisch, 2018)

Seguridad en la API

La seguridad implica proteger la información, evitando exponer los datos vulnerables o confidenciales; protección de los servicios, previniendo una garantía de funcionalidad con eficacia, es decir, con los datos esperados y completos;

En la API REST es utilizado el modelo de HTTP de modo que concede el cifrado de seguridad de la capa de transporte (TLS, en inglés *Transport Layer Security*). Este tipo de cifrado verifica que los datos enviados se encuentren cifrados y no alterados. El protocolo TLS se divide en dos partes:

1. El protocolo de enlace, en el cual el cliente y el servidor se familiarizan con las capacidades criptográficas del uno al otro, además estos establece una clave criptográfica que protege los datos en la transferencia.
2. La transferencia de datos, esta ocurre al final del protocolo de enlace, en donde al tener verificada la sesión cliente-servidor, ambas partes pueden enviar sus datos de forma segura.

TLS juega un papel importante en la protección de los datos transferidos a través de enlaces de comunicación (Siriwardena, 2020), ya que garantizan la seguridad y la privacidad de la información y se puede utilizar de forma universal, independientemente del sistema operativo y aplicación.

La seguridad en la API no es algo ordinario para tomar a la ligera, en cada organización, empresa, persona, comunidad, sistema informático, etc. existe una significancia en la información, por lo tanto, no todos los datos se protegen de la misma forma. El tipo de seguridad que se aborde en cada dato depende de su tipo.

La seguridad de la API es un campo amplio y es uno de los temas de seguridad que están ganando cada vez más atención y demanda (Vijayakumar, 2018a); es por eso, que es necesario considerar los siguientes aspectos de seguridad en la API:

- Autenticación
- Autorización
- Encriptación SSL/TLS
- Limitar la velocidad y el tráfico a través del API gateway
- Bitácora de operaciones y errores

Existen varias formas para proteger los datos y las API, por ejemplo, el uso de tokens es un modo común de hacerlo, estos influyen en la autenticación del cliente y permite los accesos a servicios y recursos de la API. La autenticación es primordial tomarla en cuenta ya que con esto logramos una seguridad con la verificación de la identidad de los clientes, en consecuencia, al contar con una autenticación. La autorización es una función que otorga permisos a los clientes para el uso de los servicios, con esto se logra una gestión de los servicios sobre los clientes, permitiendo solo el uso limitado y autorizado a estos mismos.

Versionamiento

El versionado es una práctica que permite que una API vaya evolucionando con el negocio y a su vez con los cambios tecnológicos al paso del tiempo. Ya que el manejo de actualizaciones es uno de los principales desafíos en los servicios de las APIs, por lo que una estrategia de control de versiones es crucial.

Existen diversos tipos de versionamiento, como:

- A través del Identificador de Recursos Uniforme (URI, en inglés *Uniform Resource Identifier*)
  Es el enfoque más sencillo y utilizado, ofrece formas significativamente más flexibles a través del manejo de identificadores de versión, aunque tiene la garantía de una posible ruptura de integración del cliente cuando se actualiza una versión

  https://example.com/v3

- A través de parámetros de consulta
  Esto es incluyendo el número de versión como parámetro de consulta. Es una forma sencilla de versionar una API desde el punto de vista de la implementación

  https://example.com/api/resource?version=1.2

- A través del *header* con solicitudes personalizadas
  Esta forma no satura la URI con información de versiones, sino que en el *header* se proporciona el número de versión como atributo.

  x-api-version: 1
  https://example.com/api/resource

- Mediante negociación de contenido, usando el *header Accept*
  Este enfoque permite versionar una representación de recurso único en lugar de versionar toda la API. Tiende a ser un poco más complejo, ya que los clientes deben conocer los encabezados antes de realizar una petición a un recurso.

  Accept: application/vnd.myapp+json; version=1 https://example.com/api/resource

No es posible cambiar las cosas sin tener en cuenta sus posibles impactos en los clientes existentes. La única forma de controlar este proceso es aprovechar el control de versiones. (Varga, 2016)

Especificación OpenAPI

OpenAPI (OAS), es el estándar para la definición de una API. Es agnóstico en cuánto al lenguaje y establece un contexto en común para el desarrollo, diseño y pruebas para las APIs.

La especificación OpenAPI define una descripción de interfaz estándar, independiente del lenguaje de programación para las API HTTP, que permite que tanto los humanos como las computadoras descubran y comprendan las capacidades de un servicio sin requerir acceso al código fuente, documentación adicional o inspección del tráfico de red. (Miller et al., 2021)

Esta facilita el ciclo de desarrollo iniciando por la definición, donde ofrece distintas ventajas, como:

- Permite centrarse en las necesidades del consumidor
- Creación de documentación completa
- Estandarización para el formato, códigos de estado HTTP, versionamiento, tipos de datos.

OpenAPI define como describir una interfaz API REST, es decir, una definición que describe lo que una API o un servicio realiza a través de un archivo YAML o JSON.

Beneficios
Es un formato estandarizado para describir la API REST y es legible por humanos y/o máquinas.

- Describe recursos de la API REST (propiedades o tipos de datos, puntos finales (endpoints), operaciones, parámetros y la autenticación o autorización de la API)
- La dirección u orientación, hace una fácil comprensión de lo que un servicio o la API REST hace.
- Permite extender la API con herramientas, en la cual estas pueden tomar la definición de OpenAPI como entrada y realizar diferentes acciones con ello. como validador de formato, generador de documentación, generador de SDK para consumir la API.

La definición de la API a través de OpenAPI permite comprender rápidamente que hace exactamente la API REST.

Modelo del Proceso Estadístico y Geográfico

Con la intención de que el INEGI estandarice sus procesos, se llevó a cabo la implementación del Modelo Genérico del Proceso Estadístico (GSBPM por sus siglas en inglés, *Generic Statistical Business Process Model*). Este modelo proporciona un marco estándar para ayudar

a las organizaciones estadísticas a modernizar los procesos de producción estadística, para la compartición de métodos y componentes.

GSBPM está destinado a aplicarse a todas las actividades realizadas para la producción de estadísticas oficiales, tanto a nivel nacional como internacional. (UNECE et al., 2009)

El Modelo del Proceso Estadístico y Geográfico (MPEG) utiliza un enfoque a procesos que consta de 8 fases sobre las cuales se deben situar las actividades de cada programa.

1. Documentación de necesidades

   Documentar las necesidades de información que se realizarán en el programa de información.

2. Diseño

   Se planifican las actividades que se realizarán de acuerdo con los componentes obtenidos en la fase anterior.

3. Construcción

   Elaboración y pruebas de la infraestructura informática, aplicaciones y servicios de software que será utilizado para ejecutar la producción de información.

4. Captación

   Se captan los datos y/o las imágenes, incluyendo metadatos, para la generación de productos de información estadística y geográfica.

5. Procesamiento

   Prepara los datos captados para el análisis, el cual realiza clasificaciones y metodologías de acuerdo a lo planificado en la fase de diseño dando como resultado un proceso de transformación en los datos.

6. Análisis de la producción

   Se verifica que la información sea apta para su propósito, que se encuentre lista para su uso y difusión.

7. Difusión

   Proporciona la información generada al usuario a través de los medios y estrategias establecidos.

8. Evaluación del proceso

Se examina si el ciclo de producción de información requiere un cambio o se debe seguir llevando a cabo utilizando las mismas especificaciones de necesidades, diseño y construcción.

Como se muestra en la Ilustración 9, se ve el ciclo del programa MPEG, se le llama ciclo al completar todas las fases, de inicio a fin. Las tres primeras fases denotan la planeación y las otras 5 fases son de la producción.

Ilustración 9. Ciclo del MPEG (INEGI, 2019)

En cada fase del modelo existe un Rol Responsable de Fase y Rol Responsable del Proceso, donde en cada fase estos roles cuentan una actividad a realizar con la cual cada una de estas fases logre su objetivo y actividades correspondientes.

El MPEG está soportado por un Sistema de Registro de Evidencias llamado PTracking que permite a las distintas áreas cargar la información producida en cada una de las etapas del proceso, permitiendo conservar un respaldo histórico de estas.
El MPEG permite trabajar sobre un proceso en común, para distintos tipos de proyectos, así como estructurar y reutilizar prácticas de proyectos ya definidos.

El PTracking es la plataforma transversal que a través de APIs ofrece diferentes servicios a las áreas a través de un cliente web que los sistemas informáticos del Instituto pueden utilizar.

DISEÑO DE LA INTERVENCIÓN

Tomando como base lo expuesto en el marco teórico acerca del PTracking y del MPEG, se considera que en la construcción de aplicaciones de software se procurará una estandarización de los sistemas de gestión, en este caso la API servirá como un medio para que este modelo apoye en la estandarización.

A través de las diferentes aplicaciones con las que se produce información estadística y geográfica se busca que con la API se logre homogeneidad en la automatización de intercambio de evidencias.

Actualmente la Coordinación de Informática del INEGI cuenta con una implementación en funcionamiento del sistema PTracking. El cual como se muestra en la Ilustración 10, se comparten distintos sistemas de almacenamiento, como bases de datos relacionales, un almacenamiento conectado en red (NAS, en inglés *Network Attached Storage*) y base de datos NOSQL. Cada sistema de almacenamiento tiene un fin para el instituto, en cada uno se almacena información distinta que es de suma importancia.

Los sistemas de almacenamiento que contienen resultados del procesamiento de actividades se encuentran comunicados con distintas APIs del sistema PTracking. Cada API mantiene una función objetivo, y se encuentran dividas por su uso específico para mantener una eficiencia y carga de trabajo mínima, que, a su vez cada API dispone un canal de comunicación específico, como un sistema web (sistema PTracking) y un sistema informático característico del área.

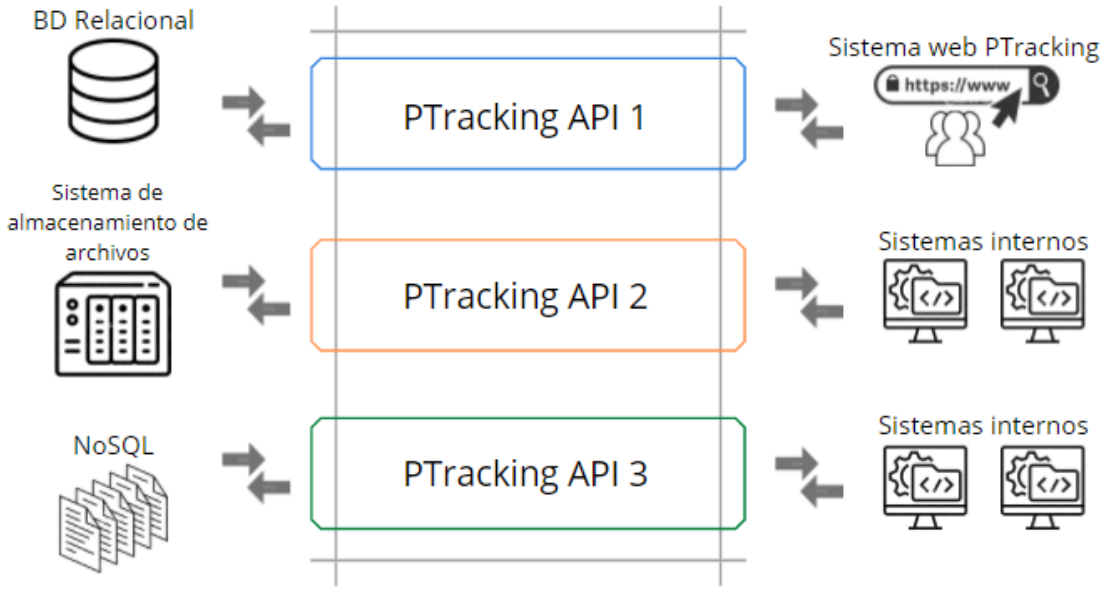


Ilustración 10. Arquitectura de Interoperabilidad

En el almacenamiento de base de datos se guardan los registros de la evidencia o resultados de cada proceso de las áreas que producen información estadística y geografía en el INEGI.

En el almacenamiento NAS, se guardan archivos, documentos e imágenes, en donde de igual manera que los registros de las bases de datos son resultados de cada etapa del proceso de producción estadística y sirven como la evidencia de la realización de alguna actividad.

Y por último se tiene la base de datos no relacional, que tiene una función en particular con el uso de las APIs que contiene el sistema de PTracking.

El sistema PTracking permite a los usuarios del Instituto cargar la información referente a sus actividades, subiendo el resultado del procesamiento de esta. A través de este sistema interno que maneja el INEGI, es posible unificar los resultados en un sistema, el cual a su vez tiene servicios de descubrimiento de procesos.

Existen diversos sistemas informáticos heterogéneos, por lo que la API lleva a cabo un proceso de homogeneización para el intercambio de información.

Definición de la API

Al comprender el funcionamiento interno de la carga de evidencias y los diversos sistemas informáticos en el Instituto, la definición de la API comprende distintos requerimientos en cuanto a seguridad, tamaño de información, permisos de acceso y constituye un canal de solicitud y respuesta web.

Arquitectura REST

La arquitectura de tipo REST del PTracking permite a los sistemas informáticos que consumen sus servicios, hacer uso de funcionalidades que son útiles para la integración de una plataforma de sistemas.

Al ser una API que se apoya en el estándar HTTP, permite que diversos sistemas se integren de manera rápida y homogénea. Este es un estilo de arquitectura cliente-servidor, formado por clientes, servidores, recursos y todo sobre una gestión de solicitudes HTTP.

Para que una API sea considerada de tipo REST debe cumplir con distintos criterios, como:

- Arquitectura cliente-servidor, se encuentran tenuemente acopladas solo por la interfaz de comunicación, lo que significa que no hay preocupación por cómo se trabaja de un lado o de otro.
- Sistema sin estado, no se almacena la solicitud y estas se encuentran independientes.
- Un sistema jerárquico en capas (seguridad, permisos, balanceo de carga), cada una de estas capas lleva a cabo una operatividad dentro del sistema. Ayudar a mejorar el rendimiento, escalabilidad y seguridad.
- Manipulación con URI, facilita el acceso a la información.
- Hipermedia como el motor del estado de la aplicación (HATEOAS, en inglés, *Hypermedia as the engine of application state*), permite que el cliente pueda moverse por la aplicación siguiendo los identificadores únicos URI en formato hipermedia

Así pues, la API REST es independiente del tipo de plataforma o lenguaje de programación, ofrece una libertad para la implementación y desarrollo. En la Ilustración 11, se muestra la arquitectura base de una API REST, la cual consiste en una petición por parte del cliente con un formato JSON, esta petición a su vez pasa por un método HTTP y el servidor de la API lo obtiene y realiza su función.

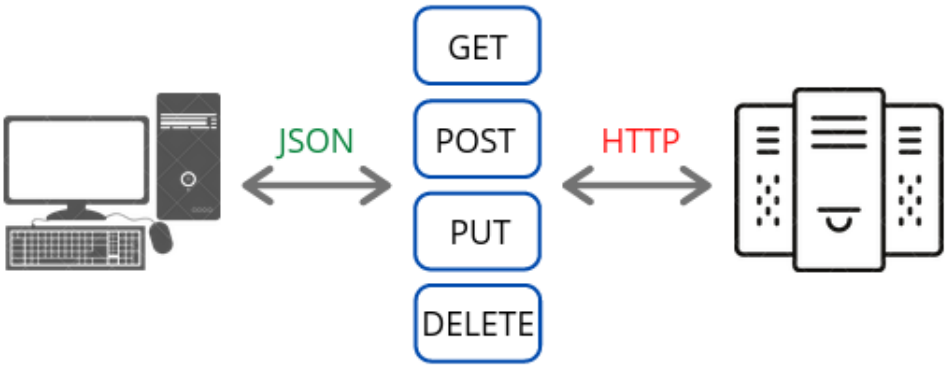


Ilustración 11. Arquitectura base API REST

Esta arquitectura de cliente-servidor es una interfaz uniforme, que provoca que ambas partes sean completamente independientes, lo que permite que cualquier parte pueda realizar cambios internos sin verse afectada la comunicación de peticiones, mientras la interfaz no cambie.

El diseño de una API REST está destinado a promover la longevidad del software y la evolución independiente, esto es, que muchas limitaciones se oponen directamente a la eficiencia a corto plazo, es decir, un buen diseño a corto plazo proveerá la funcionalidad deseada en el momento pero se debe pensar en el diseño a largo plazo para mantener una escalabilidad fiable.

Versiones del API

El control de versiones es una parte crucial del diseño de la API, ya que brinda a los desarrolladores la posibilidad de realizar cambios en la API, ya sea de correcciones o mejoras, sin fracturar algún servicio al cliente mientras se implementan las actualizaciones.
La API debe evolucionar de acuerdo con el modelo de negocio y su alineación, además de evolucionar tecnológicamente.

Se cuenta con distintos tipos de versionado como se comentó en la fundamentación teórica. Basándose en la fundamentación y en la forma de trabajo del PTracking, es ideal integrar el versionamiento a través de la URL, como se muestra en la Tabla 2, debido a que es simple, escalable y práctico.

https://www.apirest-ejemplo.com.mx/v1.0/users
https://www.apirest-ejemplo.com.mx/v2.0/users
https://www.apirest-ejemplo.com.mx/v2.1/products

Tabla 2. Versionamiento de API por URI

Mantener la compatibilidad de los servicios con versiones anteriores a menudo es un costo prohibitivo y/o muy difícil de mantener. Para estos casos, al ejecutar un cambio de versión en la API, se deberá proveer un aviso anticipado a los clientes que la consumen para permitir que se realicen las adecuaciones necesarias por cada cliente de tal forma que este enfoque sea más pragmático en los usuarios.

Los cambios de versionamiento de la API son necesarios cuando se constituye a un cambio radical en un punto final de la API, es decir cualquier cambio que obligue al consumidor a realizar un cambio.

Por lo tanto, los desarrolladores esperan que el funcionamiento de la API no se vea alterado por nuevas implementaciones o innovaciones, y el versionado es una clave para lograr que la API obtenga éxito a largo plazo.

Se podrá estructurar las versiones en 2 apartados, como se muestra en la Tabla 3.

Mayor	Puede llegar a afectar el funcionamiento de los consumidores de la API
Menor	Cambios mínimos que no alterar el formato de entrada o respuesta de las peticiones

Tabla 3. Estructura de versionamiento

De tal manera, que la versión estaría compuesta por Mayor.Menor, por ejemplo, versión *1.5*, que quiere decir que es la versión 1 de la API y se le han hecho 5 cambios mínimos, como actualizaciones o corrección de errores.

El proceso de versionado de servicios se vería representado de la siguiente manera:

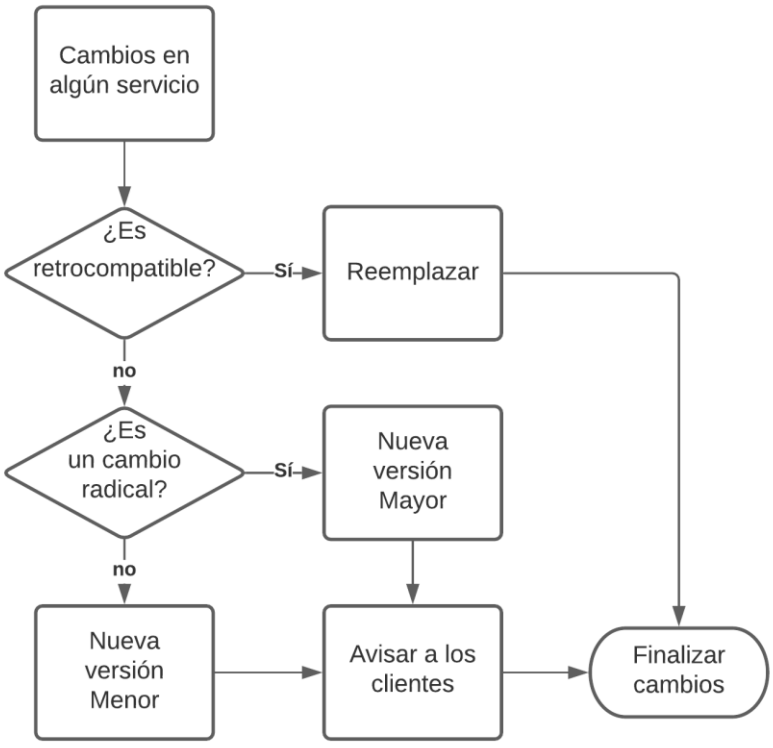


Ilustración 12. Proceso de versionado

Para mantener un control sobre el versionado, se recomienda utilizar un sistema de control de versiones (VCS, en inglés *Version Control System*) para mantener de forma eficaz un proyecto de software de forma distribuida y ordenada a lo largo del tiempo.

Ofrece varias ventajas, como:

- Trabajo colaborativo
- Permite generar flujos de trabajo flexibles
- Sistema distribuido
- Integridad de la información asegurada
- Reduce los tiempos de despliegue
- Resolución de conflictos

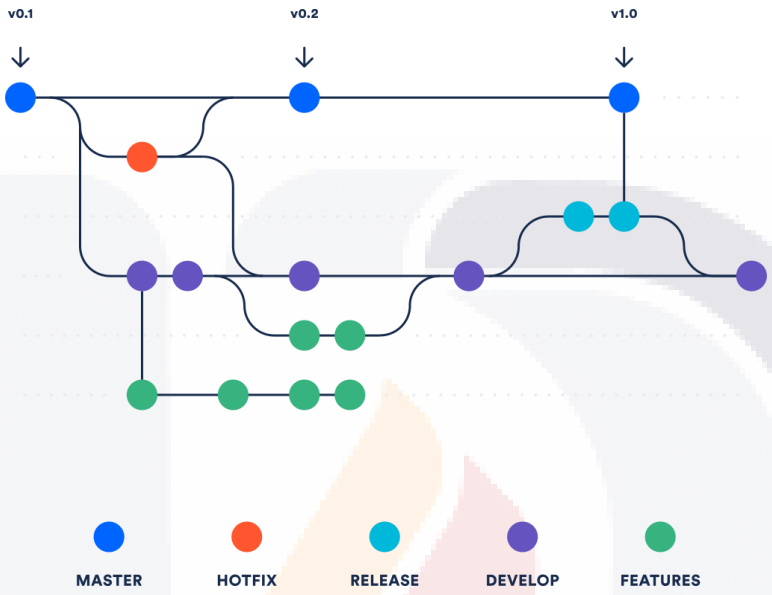


Ilustración 13. Ejemplo de trabajo con VCS (Atlassian, 2019)

El sistema de control de versiones no solo es para código fuente, sino para todo lo que conforma el proyecto. Hay muchas opciones de software que ayudan a llevar el control de versiones, como: Git, Mercurial, SVN y Preforce.

Además, para la creación de la API con el uso de un VCS otorga un sobresaliente flujo de trabajo sobre el desarrollo del software y asimismo el proyecto podría ser almacenado en un servicio de almacenamiento o repositorio con sistema de control de versiones.
Todos los repositorios contienen un historial del proyecto y las funciones se pueden incorporar desde el repositorio de cualquier desarrollador al repositorio de cualquier otra persona. (Haaranen & Lehtinen, 2015)

Proceso de gestión, autenticación y validación

Este es un proceso muy sustancial en la API, donde se verifica que existe una autenticación, procesa la solicitud de petición y la delega a su recurso para devolver un resultado, registra todas las solicitudes entrantes y valida el formato de entrada para obtener una capa de seguridad dentro de esta. Todas estas actividades están sobre un conjunto de servicios y funciones de software llamado middleware, que este a vez se localiza implícito en un servicio llamado API Gateway. Estas capas pueden variar de una API a otra y es debido a los requerimientos y la forma de trabajo que se tendrá con la API.

Con el uso del PTracking para la interoperabilidad de los sistemas informáticos, es beneficioso implementar esta capa encapsulada con la variedad de servicios. Por medio del API Gateway se orquesta el enrutamiento de los servicios solicitados, de tal manera que sería posible administrar la carga de la API, ya sea realizando un balanceo de carga a través del servicio o proteger el servicio de esta misma en caso de encontrar alguna amenaza o acceso no autorizado.

Esencialmente el middleware realiza funciones específicas en la solicitud antes de realizar el servicio de la API. Es un patrón de diseño para agregar de manera elocuente el proceso transversal como el manejo de autenticación sin tener que realizar grandes cambios con otros puntos o servicios en el código de programación.

Como se mostró en la Ilustración 10, se cuentan con distintos tipos de almacenamiento, en donde uno de estos es una base de datos no relacional (NoSQL), este almacenamiento servirá para colocar los log o registros de las peticiones a la API. Estos logs registran cuando entra una solicitud, independientemente de cuál sea la respuesta de la API. Esta bitácora permite analizar y detectar ya sean errores relativos al sistema, red, problemas operacionales o actividades irregulares.

Cada componente del middleware se instancia como un paso en una canalización, por lo que el orden puede ser importante para la corrección funcional. En la Ilustración 14, se observa el flujo que llevará la solicitud hasta llevarlo al servicio requerido.

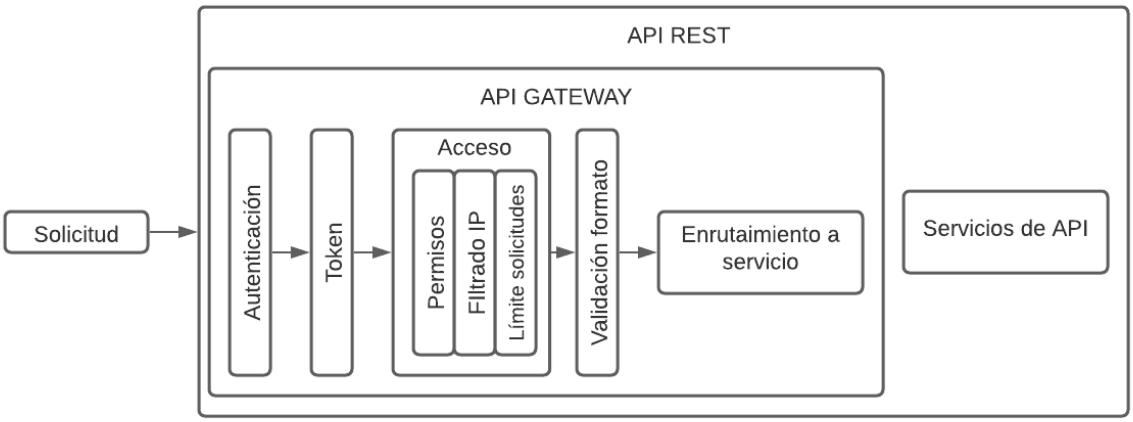


Ilustración 14. Proceso API Gateway - Middleware

En este flujo primero arriba la solicitud a la API entra la solicitud a la API, inmediatamente entra en acción el API Gateway que contiene el funcionamiento del Middleware. Dentro del Middleware entra en funcionamiento la Autenticación del usuario, un token de seguridad, así como el Acceso o Autorización que comprende permisos de servicios, filtrado de IP hasta un límite de solicitudes, al obtener la validación de acceso, entra la validación del formato de la solicitud. Al obtener una garantía del Middleware pasa al enrutamiento de servicio, donde se orquesta la solicitud al servicio requerido de la solicitud del cliente.

En cada solicitud dentro de la API es necesario almacenar un log o un registro, desde que entra la solicitud del cliente hasta que termina el proceso de salida u ocurra un problema.
Estos logs sirven para diversas cosas, ya sea para analizar el tiempo de respuesta y procesamiento, mantener la seguridad de la API por actividades irregulares o problemas operacionales.
Por lo tanto, para poder conocer a detalle y trabajar con los logs, es preciso que contengan ciertos campos como:

- Usuario o cliente
- IP
- Fecha
- Hora
- URI
- Datos de la solicitud
- Respuesta de la solicitud
- Tiempo de procesamiento

Así también, es requisito acumular estos logs dentro de un base de datos NoSQL, la cual ofrece esquemas flexibles, escalabilidad y un alto rendimiento, de manera que por cada solicitud a un servicio de la API se deberá almacenar un log y una NoSQL es eficiente para esta tarea.

Al conocer que la API REST no tiene estado, la obtención del token es conveniente para recordar el usuario y contraseña del cliente entre las llamadas a los servicios de la API. Que es una cadena de texto encriptada con clave.

Nombre de los recursos

Los objetos o servicios de la API REST son consumidos a través de un Identificador de Recursos Uniforme (URI) o *endpoint*, de manera que estos no pueden ser duplicados.
Estos recursos deben cumplir con unas características para brindar una mejor comprensión de este:

- Coherente
- Intuitivo
- Sencillo

Las definiciones de las URI deben ser similares a estructuras de directorios. Es decir, de tipo jerárquico, este tipo de nivel de usabilidad en el nombre no es solo un nombre delimitado por barras inclinadas (*slash*), sino un árbol con ramas con distintos niveles de grado para cada servicio.

Al crear la estructura del URI, es aconsejable atender estas directrices:

- Texto en minúsculas
- Utilizar guiones en vez de espacios
- Estáticas, la relación entre el nombre del URI y la implementación del proceso requiere ser independiente
- Implicar una acción

- Ser única
- Ser lo más corta posibles
- Forma plural para lograr uniformidad
- Semántica para el cliente

La estructura básica de una URI sería como se muestra en la siguiente imagen.

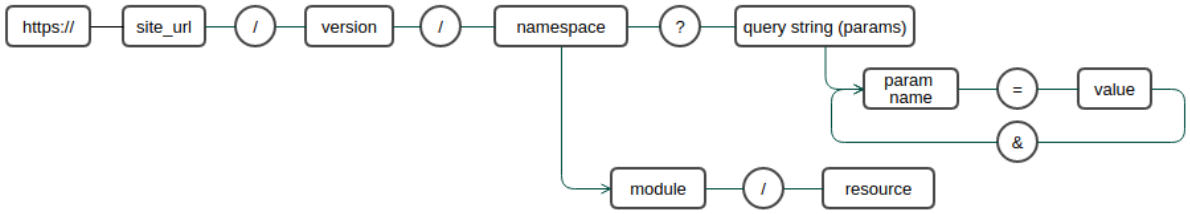


Ilustración 15. Estructura URI

Por ejemplo:

https://apirest.com.mx/v1.0/clients/get

Códigos de estado

Cuando se ejecuta una llamada a un recurso es de suma importancia conocer si el proceso se realiza de manera satisfactoria o por otra parte si ha resultado en algún error. Para esto, se tienen los códigos de estado de respuesta HTTP.

Las tablas siguientes muestran los códigos de estado según la respuesta del recurso.

Código de estado	Descripción
200 Correcto	La solicitud se ha completado satisfactoriamente
201 Creado	La solicitud se ha completado satisfactoriamente; se ha creado un recurso nuevo
202 Aceptado	La solicitud se ha recibido, pero aún no se ha completado. (acciones que tardan mucho en completarse)

Tabla 4. Códigos de estado con respuesta satisfactoria

Código de estado	Descripción
400 Solicitud incorrecta	La solicitud contiene parámetros que no son válidos o que están ausentes
401 No autorizado	El cliente no está autorizado a realizar la solicitud
403 Prohibido	El cliente no está autorizado a completar la solicitud; no posee los permisos necesarios
404 No encontrado	El recurso solicitado no existe
406 No aceptable	Se ha solicitado un tipo de contenido o de codificación de contenido no soportado

415 Tipo de soporte no soportado	La solicitud contiene un tipo de contenido o una codificación de contenido desconocidos

Tabla 5. Códigos de estado con respuesta de errores esperados

Código de estado	Descripción
500 Error interno del servidor	Se ha producido un problema
501 No implementado	La solicitud no tiene soporte en la API REST
503 Servicio no disponible	El servidor no está listo para manejar la petición

Tabla 6. Códigos de estado con respuesta de errores inesperados

Con el código de estado es posible agregar en otro campo un mensaje de respuesta (si es necesario).

Formato de entrada y salida

La información o representación es posible en distintos formatos, como JSON, XML. XLT o texto sin formato. JSON es el lenguaje de programación más popular, ya que es simple de comprender tanto las máquinas como las personas.

La API REST que utilizan el formato JSON constituyen un estándar de facto que muchas tecnologías del lado del servidor pueden decodificar sin mucha carga de trabajo.
Aunque también existe el XML que le secunda, este no es fácilmente manipulable en el lado del cliente y necesita una conversión del lado del servidor que ocupa un poco más de procesamiento.

Este estándar basado en texto plano que es para el intercambio de información entre distintas tecnologías y es homogéneo en su uso; lo cual para los servicios de la API el uso de este formato permite un soporte de estructuras, como objetos y arreglos, lo cual proporciona una gran sencillez en las estructuras.

Los datos en el formato JSON contienen una colección de pares llave-valor.

- La llave o key es como su nombre lo indica, el nombre del dato para identificarlo de forma única, este es una secuencia de caracteres entre comillas.
- El valor, son los tipos de datos válidos, ya sea un arreglo, un objeto, cadena, booleano, número o nulo.

Cabe mencionar que un objecto JSON comienza y termina con llaves {}. Al utilizar distintas llaves y valor, estas deben ir separadas por una coma, y cada llave es seguida por dos puntos para hacer diferencia con su valor.

En la Ilustración 16 se muestra un ejemplo de un formato JSON con distintos tipos de valores.

```json
{
    "key1": true,
    "key2": 123,
    "key3": "text",
    "key4": [
        {
            "a1": "abc"
        },
        {
            "a1": "abc",
            "a2": "def"
        }
    ],
    "key5": {
        "o1": "object",
        "o2": false,
        "o3": 98765
    }
}
```

Ilustración 16. Ejemplo formato JSON con distintos tipos de datos

Tanto en el formato de entrada como de salida se empleará JSON, lo cual permite una mejor organización e implementación de la API. Al seguir el formato con las reglas que especifica puede aumentar la productividad, aprovechar las herramientas y enfocar en la lógica de los servicios del negocio.

Al realizar la solicitud a la API REST se deberá especificar el encabezado *Content-Type* con el valor application/json, esto para asegurarnos que responda con formato JSON y sea simple el manejo de las respuestas.

Toda solicitud de un servicio de la API debe regresar una salida, esto para detectar y analizar el comportamiento de esta misma. Si la petición contiene todos los parámetros correctos y no exista ningún inconveniente, la respuesta en la API mostrará un código de respuesta 200, el cual indica que la solicitud se ha procesado correctamente. Esta respuesta contiene una llave principal llamada response, donde este contiene toda la información del resultado de la solicitud, que a su vez comprende distintas llaves con su valor, las cuales son:

- result: Incluye un valor de tipo booleano, donde se indica si la solicitud completa se procesó correctamente.
- response_code: Muestra el código de respuesta HTTP en un tipo numérico.
- message: Mensaje breve relacionado al código de respuesta.
- data: Arreglo que contiene la información resultante al servicio solicitado.
- result: Valor de tipo booleano, donde se indica si la solicitud del servicio se procesó correctamente.
- response_code: Muestra el código de respuesta de la solicitud del servicio.
- message: Mensaje breve relacionado al código de respuesta.
- data: Arreglo con los datos devueltos del proceso del servicio solicitado.

A continuación, en la Ilustración 17 se muestra un ejemplo de una respuesta del procesamiento de un servicio de la API.


```json
{
    "response": {
        "result": true,
        "response_code": 200,
        "message": "Request successful",
        "data": {
            "result": true,
            "response_code": 200,
            "message": "Process finished successfully.",
            "data": {
                "id": 112233,
                "status": "active"
            }
        }
    }
}
```


Ilustración 17. Respuesta JSON de solicitud procesada correctamente

Para suprimir la confusión de los clientes de la API al ocurrir un problema, estos se manejan con distinción y devuelve los códigos de respuesta HTTP en el formato de salida donde indican el tipo de dificultad en la petición. Por este motivo, los consumidores y administradores de la API consiguen suficiente información para comprender el problema ocurrido.

Una respuesta de un problema regresa el código de error, así como un mensaje corto y conciso de lo ocurrido. En la Ilustración 18 se observa un ejemplo de la respuesta en el formato JSON de una solicitud a un servicio con un parámetro incorrecto.


```json
{
    "response": {
        "result": false,
        "response_code": 400,
        "message": "Malformed request. Invalid data types of params in 'request'",
        "data": {
            "invalidParams": [
                "wrongParam"
            ]
        }
    }
}
```


Ilustración 18. Respuesta JSON de solicitud con parámetro incorrecto

Modelo Espiral y Metodología PSP

Para la definición de la API se utilizó el modelo en espiral junto con la metodología Proceso Personal de Software (PSP, en inglés *Personal Software Process*), debido a que el modelo espiral es un tipo de gestión de proyectos con la técnica iterativa con una estructura fija de fases. Además de que este es genérico y puede combinarse con demás métodos de desarrollo clásicos y ágiles. Y el PSP que al ser un proceso personal y estar basado en principios de mejora, permite establecer las metas, identificar los métodos, además de que permite analizar los resultados y así ajustar para obtener los resultados esperados de calidad.

Una ventaja de usar este modelo espiral es que minimiza los riesgos ya que los conflictos entre los requisitos y el diseño se evitan mediante un enfoque cíclico, y los resultados se van comprobando constantemente, y si es necesario, modificarse.

La principal ventaja del modelo en espiral es que su gama de opciones se adapta a las buenas características de los modelos de procesos de software existentes, mientras que su enfoque basado en el riesgo evita muchas de sus dificultades. (Boehm, 1988)

Como se muestra en la Ilustración 19, el modelo es iterativo con una mejora continua minimizando riesgos siguiendo las fases en orden obteniendo retroalimentación de una iteración a otra.

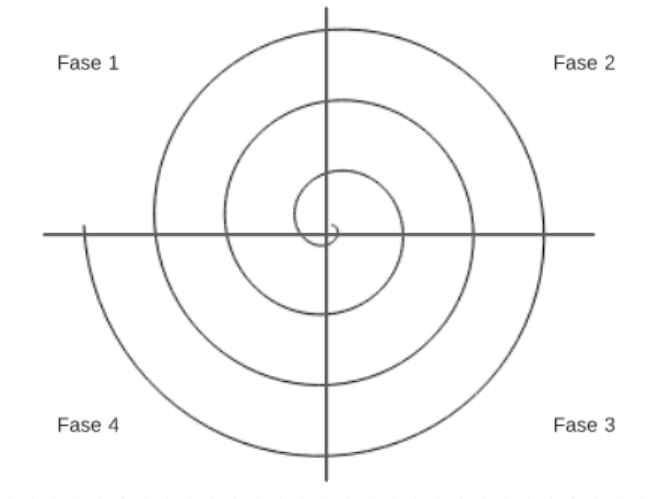


Ilustración 19. Modelo espiral

El desarrollo de la definición de la API se llevó a cabo con las 4 fases principales del modelo.

- Fase 1: Planificación: Se definen los recursos, y demás información relacionada con el proyecto.
- Fase 2: Análisis y evaluación de riesgos: Se identifican y evalúan los riesgos potenciales.
- Fase 3: Desarrollo y prueba: Se realizan las características del proyecto y se van creando prototipos donde se añaden funcionalidades.
- Fase 4: Planificación del siguiente ciclo: De las fases anteriores se toman los resultados de las evaluaciones y pruebas para volver a planificar las mejoras.

Como complemento, la metodología PSP proporciona un marco de trabajo estructurado para planificar el trabajo, y obtener productos de la mejor calidad, los procesos del modelo PSP se muestran en la Ilustración 20. Ya que se recopilan datos y el tiempo que lleva desarrollar un producto está determinado por el tamaño de este. Además el PSP puede ser aplicado en distintos aspectos de los trabajos de software, como la especificación de requerimientos, definición de procesos, pruebas y eliminación de defectos.

El uso del PSP para la fase de desarrollo se verá únicamente en la creación de un prototipo de la API con las especificaciones definidas en este trabajo y será medible a través de días el cual es correlacional al tiempo de desarrollo de la API.

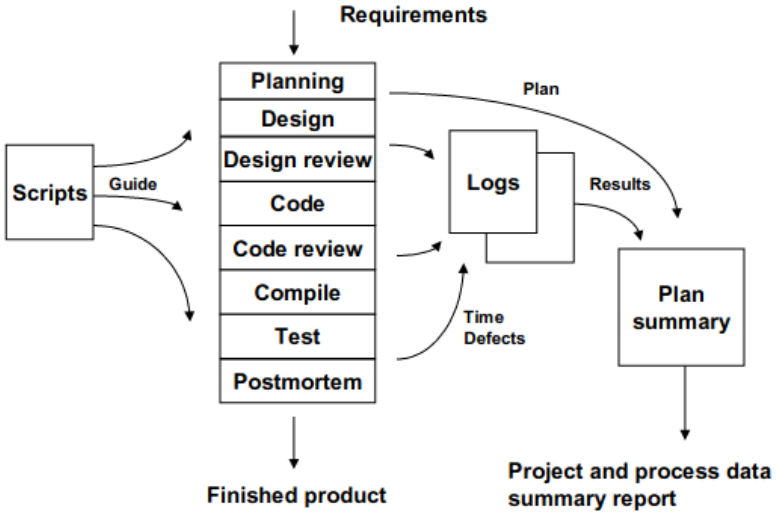


Ilustración 20. Flujo de proceso PSP (Humphrey, 2000)

El modelo fue usado con base *expertise* y requerimientos del proyecto de MPEG y PTracking, el cual busca una transferencia segura de la información, además de que la API continuará creciendo en distintas características.

RESULTADOS DE LA INTERVENCIÓN

En este capítulo se presentan los resultados del análisis del diseño de la API. Estos resultados mostrarán las especificaciones, como la arquitectura, el flujo de la API, el funcionamiento del middleware, los servicios e integración de la API con distintos sistemas..

Arquitectura

La arquitectura REST contempla distintos conceptos en los cuáles, algunas personas no entienden lo que realmente es REST y deducen como una definición de API. La investigación de Roy Fielding en la cual da a conocer REST a la sociedad no especifica la forma de crear una API tal cual, sino sobre HTTP en sí mismo, es por eso que REST está pensado como una solución para atender cuestiones de escalabilidad y consistencia.

El flujo de autenticación por token es que el cliente envía el usuario y contraseña de acceso al servicio de la API específico para la obtención del token. Por lo que la API puede responder con 2 respuestas en general, como se muestra en la Ilustración 21:

1. Si es correcto, se autentica al cliente y genera el token del lado del servidor que asimismo se le regresa la cadena del token al cliente.
2. Si no es correcto, se retorna el código de error al cliente.

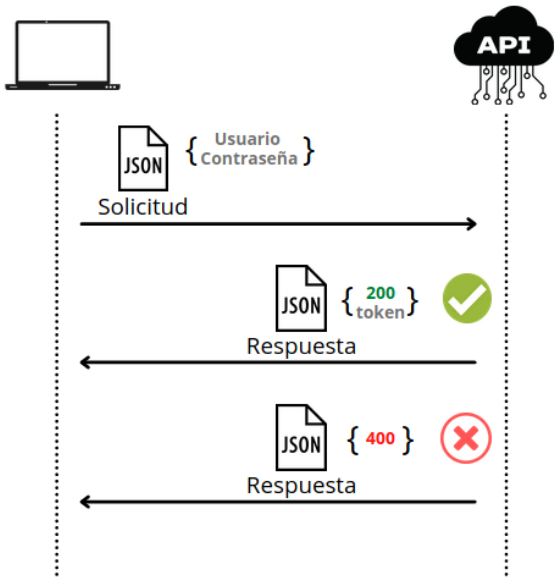


Ilustración 21. Flujo de token

El token de acceso será usado para futuras llamadas a los servicios requeridos. Es decir, cada vez que el cliente desee consumir un servicio tendrá que enviar el token al servidor junto al cuerpo de la petición para notificar que se encuentra correctamente autenticado.
Al recibir la petición de la llamada, la API verifica sí es válido y concede el acceso para realizar el servicio.

Cabe mencionar que el token contiene un tiempo de validez que se establece dentro de la API al generar el token. Durante el tiempo válido de vida, la API autoriza el acceso a los recursos solicitados y autorizados al cliente. Es posible crear un servicio de renovación de token, donde éste serviría para aumentar el tiempo de validez del token sin necesidad de enviar nuevamente el usuario y contraseña, aunque es opcional y por seguridad es preferible generar un nuevo token al expirar.

Flujo

El flujo de una solicitud simple a través de la API REST, como se muestra en la Ilustración 22, se compone de la siguiente manera:

- El cliente realiza una solicitud al servicio en formato JSON con los parámetros de autenticación especificados, ya sea el token de autorización o su usuario y contraseña; los parámetros del servicio requeridos; y con el método de HTTP requerido por el servicio, GET, POST, PUT o DELETE.
- La API recibe la solicitud y entra en acción el middleware, donde se verifica la estructura general de ésta, en caso de que sea incorrecta la estructura de la solicitud retorna el código de error 400.
- Si existe el token como parámetro de autenticación, se valida. En caso de que el token no sea correcto o se encuentre expirado retorna el código de error 401.
- Si no tiene token, se valida el usuario y contraseña, en caso de ser incorrecta la autenticación retorna el código de error 403.
- Al tener la autenticación se obtiene la URL del servicio solicitado y se valida que sea correcta en formato y en el nombre del servicio, si no es validada retorna el código de error 404.
- Con la URL del servicio obtenido se verifica si el recurso solicitado se encuentra en caché, si sí se encuentra retorna el código de respuesta 200 y los datos en la respuesta del servicio requerido. Si no se tiene en caché, la API procesa el recurso solicitado y retorna el código de respuesta 200 con la respuesta procesada al cliente.

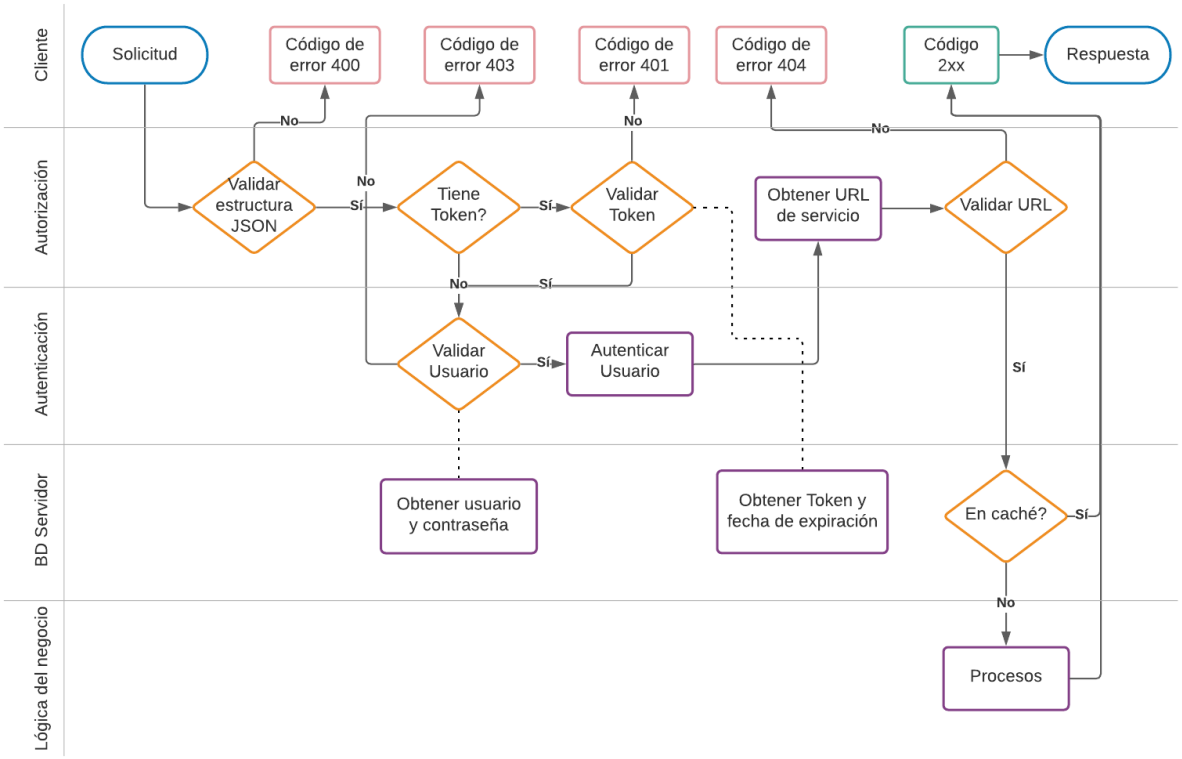


Ilustración 22. Flujo de solicitud a un servicio a la API

Middleware

Es un código intermedio que se encuentra entre la petición del cliente y el código de la API en donde protege de peticiones indebidas.

El siguiente código muestra el funcionamiento del middleware para la autenticación del usuario y validación de la petición.

```
1.  class AuthApi {
2.      public static function handle($request, Closure $next) {
3.          $requestArray = json_decode($request->getContent(), true);
4.          // Validate structure and login / token
5.          $auth = ApiConfig::authenticate($requestArray);
6.          if($auth !== true) {
7.              return ApiConfig::json($auth);
8.          }
9.          // Get path - service
10.         $path = ApiConfig::setArrayPath(explode('/', $request->path()));
11.         $serviceURL = ApiConfig::validateServiceURL($path);
12.         if($serviceURL !== true) {
13.             return ApiConfig::json($serviceURL);
14.         }
15.         // Validate data from request
16.         $requestData = ApiConfig::validateRequestData($requestArray, $path);
17.         if($requestData !== true) {
18.             return ApiConfig::json($requestData);
19.         }
20.         return $next($request);
21.     }
```

```
22. }
```

Ilustración 23. Función general de middleware

Lo primero que realiza es obtener los datos de la petición en formato JSON, para así realizar el proceso de autenticación donde se revisan los *headers* de la petición y obtiene el valor del token, en caso de no contar con el parámetro token entra en acción la función de validar la autenticación de la petición, donde se comprueba que los parámetros estén con la correcta estructura.

```
1.  public static function authenticate($data) {
2.      $validate = ApiConfig::validateRequestAuth($data);
3.      if($validate === true) {
4.          ApiConfig::$token = Arr::get($data, 'headers.auth.token');
5.
6.          if(isset(ApiConfig::$token)) {
7.              return ApiConfig::validateToken($data);
8.          }
9.
10.         ApiConfig::$user = Arr::get($data, 'headers.auth.user');
11.         ApiConfig::$password = Arr::get($data, 'headers.auth.password');
12.         ApiConfig::$request = Arr::get($data, 'request');
13.         return ApiConfig::authenticateUser();
14.     }
15.     return $validate;
16. }
```

Ilustración 24. Función de autenticación

La validación de la petición confirma que los parámetros de autenticación de usuario y contraseña estén en los *headers* de la petición para así obtenerlos y realizar la comparación. En caso de que carezca de algún parámetro en los *headers* para la autenticación, esta parte del middleware retornará la respuesta con el código de error y mensaje establecido. Así también valida que contenga en la estructura un parámetro de *request* que contiene los datos de la petición hacia el servicio requerido; en caso de no contar o algún parámetro de la estructura se retorna un error en la respuesta.
Si atraviesa todas las validaciones correctamente seguirá con el proceso de autenticar el usuario y contraseña.

```
1.  public static function validateRequestAuth($data) {
2.      foreach($data as $key => $items) {
3.          if($key === 'headers') {
4.              foreach($items as $item_key => $item) {
5.                  if($item_key === 'auth') {
6.                      foreach($item as $item2_key => $item2) {
7.                          if($item2_key === 'user') {
8.                              continue;
9.                          } elseif($item2_key === 'password') {
10.                             continue;
11.                         } else {
12.                             http_response_code(self::API_WRONG_PARAMS);
13.                             return [
14.                                 'response' => [
15.                                     'result' => false,
```

```
                                    'response_code' => self::API_WRONG_PARAMS,
                                    'message' => 'The key \''.$item2_key.'\' is not
    allowed',
                                    'data' => $data
                                ]
                            ];
                        }
                    }
                } else {
                    http_response_code(self::API_WRONG_PARAMS);
                    return [
                        'response' => [
                            'result' => false,
                            'response_code' => self::API_WRONG_PARAMS,
                            'message' => 'Malformed request. The key \''.$item_key.'\' is
    not allowed',
                            'data' => $data
                        ]
                    ];
                }
            }
        } elseif($key === 'request') {
            continue;
        } else {
            http_response_code(self::API_WRONG_PARAMS);
            return [
                'response' => [
                    'result' => false,
                    'response_code' => self::API_WRONG_PARAMS,
                    'message' => 'Malformed request. The key \''.$key.'\' is not
    allowed',
                    'data' => $data
                ]
            ];
        }
    }
    return true;
}
```

Ilustración 25. Función de validación para formato de la petición

Los códigos de respuesta que retornan las funciones del middleware se muestran en lo siguiente.

```
const API_SUCCESS =                200;
const API_CREATED =                201;
const API_ACCEPTED =               202;

const API_WRONG_PARAMS =           400;
const API_NOT_AUTHORIZED =         401;
const API_FORBIDDEN =              403;
const API_NOT_FOUND =              404;
const API_NOT_ACCEPTED =           406;
const API_NOT_SUPPORTED =          415;

const API_SERVER_ERROR =           500;
const API_NOT_IMPLEMENTED =        501;
const API_NOT_AVAILABLE =          503;
```

Ilustración 26. Códigos de respuesta de la API REST

La parte de autenticación del usuario, valida que el usuario se encuentra asignado para hacer uso de la API, este realiza una consulta a la tabla *user* y revisa si existe el usuario que realiza la solicitud, si existe realiza la validación de la contraseña, cabe mencionar que la contraseña se encuentra encriptada por seguridad, si fue correcta retorna que aprobó la validación, en caso de que no exista el usuario en la tabla retorna una respuesta de que 'la autenticación falló, que no fue encontrado el usuario', así mismo con la contraseña, si fue correcta retorna el mensaje de que 'la autenticación falló, la contraseña es incorrecta'.

```
1.  public static function authenticateUser() {
2.     $row = ApiConfig::where('user', ApiConfig::$user)->first();
3.     if($row) {
4.         if(password_verify(ApiConfig::$password, $row['password']) === true) {
5.             return true;
6.         } else {
7.             http_response_code(self::API_FORBIDDEN);
8.             return [
9.                 'response' => [
10.                    'result' => false,
11.                    'response_code' => self::API_FORBIDDEN,
12.                    'message' => 'Authentication failed. Incorrect password',
13.                ]
14.            ];
15.        }
16.    } else {
17.        http_response_code(self::API_FORBIDDEN);
18.        return [
19.            'response' => [
20.                'result' => false,
21.                'response_code' => self::API_FORBIDDEN,
22.                'message' => 'Authentication failed. User key not found',
23.            ]
24.        ];
25.    }
26. }
```

Ilustración 27. Función de autenticación de usuario

Al ser aprobado por la autenticación y la validación de la estructura, el middleware prosigue con la obtención de la ruta del servicio requerido, dónde esta función sólo regresa la URN de la petición en formato de arreglo. Esto para realizar la validación de la estructura de cada servicio.

```
1.  public static function setArrayPath($path) {
2.     if(sizeof($path) == 2) {
3.         return [
4.             $path[0] => $path[1]
5.         ];
6.     }
7.     return null;
8.  }
```

Ilustración 28. Función para obtener ruta (servicio)

Con la ruta del servicio recaba por la función anterior se valida que sea una URL correcta, esta corrobra que el servicio solicita por la URL exista, en caso de que no, retorna el código de respuesta 404 de recurso solicita no existente.

```
1.  public static function validateServiceURL($path) {
2.     $keyPath = key($path);
3.     if(isset(ApiConfig::REQUESTS[$keyPath][$path[$keyPath]])) return true;
4.     else {
5.         http_response_code(self::API_NOT_FOUND);
6.         return [
7.             'response' => [
8.                 'result' => false,
9.                 'response_code' => self::API_NOT_FOUND,
10.                'message' => 'Service URL not found',
11.                'data' => []
12.            ]
13.        ];
14.    }
15. }
```

Ilustración 29. Función de validación de servicio

Teniendo el arreglo que contiene la ruta de servicio se procede a validar los datos del parámetro *request* de la solicitud. Esta función del middleware itera sobre cada parámetro del *request* para así inspeccionar que el tipo de dato recibido es el ideal para esa solicitud, es decir, es decir, si se tiene un servicio de obtener información de un reporte específico donde en la solicitud se requiere un Id, este valor deberá ser numérico o como se haya especificado en los parámetros requeridos del servicio. Esta va almacenando todos los parámetros que están incorrectos y retorna los resultados con el mensaje de los datos incorrectos, ya sea por el tipo de dato, de que algún parámetro no se encuentre especificado en el servicio o que haga falta algún parámetro en la solicitud.

```
1.  public static function validateRequestData($data, $path) {
2.     if(array_key_exists('request', $data)) {
3.         $keyPath = key($path);
4.         $paramsRequest = ApiConfig::REQUESTS[$keyPath][$path[$keyPath]];
5.         $wrongDataTypes = $wrongKey = $duplicateKeys = $foundKeys = array();
6.         foreach($data['request'] as $itemKey => $itemValue) {
7.             if(!array_key_exists($itemKey, $paramsRequest)) {
8.                 // if the params is invalid
9.                 $wrongKey[] = $itemKey;
10.            } else {
11.                $dataType = $paramsRequest[$itemKey];
12.                switch($dataType) {
13.                    case ConfigRequest::TYPE_REQUEST_STRING:
14.                        if(!is_string($itemValue)) {
15.                            $wrongDataTypes[] = 'Invalid data type of param
    \''.$itemKey.'\'. Must be \''.ConfigRequest::TYPE_REQUEST_STRING.'\'';
16.                        }
17.                        break;
18.                    case ConfigRequest::TYPE_REQUEST_INT:
19.                        if(!is_numeric($itemValue) || is_float($itemValue) ||
    is_string($itemValue)) {
20.                            $wrongDataTypes[] = 'Invalid data type of param
    \''.$itemKey.'\'. Must be \''.ConfigRequest::TYPE_REQUEST_INT.'\'';
21.                        }
22.                        break;
23.                    case ConfigRequest::TYPE_REQUEST_FLOAT:
```

```
24.                         if(!is_float($itemValue) && (!is_numeric($itemValue) ||
    is_string($itemValue))) {
25.                             $wrongDataTypes[] = 'Invalid data type of param
    \''.$itemKey.'\'. Must be \''.ConfigRequest::TYPE_REQUEST_FLOAT.'\'';
26.                         }
27.                         break;
28.                     case ConfigRequest::TYPE_REQUEST_ARRAY:
29.                         if(!is_array($itemValue)) {
30.                             $wrongDataTypes[] = 'Invalid data type of param
    \''.$itemKey.'\'. Must be \''.ConfigRequest::TYPE_REQUEST_ARRAY.'\'';
31.                         }
32.                         break;
33.                     default:
34.                         $wrongDataTypes[] = 'Invalid data type of param \''.$itemKey.'\'.
    Data type not allowed';
35.                         break;
36.                 }
37.                 $foundKeys[] = $itemKey;
38.             }
39.         }
40.         if(sizeof($wrongKey) > 0) {
41.             http_response_code(self::API_WRONG_PARAMS);
42.             return [
43.                 'response' => [
44.                     'result' => false,
45.                     'response_code' => self::API_WRONG_PARAMS,
46.                     'message' => 'Malformed request. Invalid data types of params in
    \'request\'',
47.                     'data' => ['invalidParams' => $wrongKey]
48.                 ]
49.             ];
50.         } elseif(sizeof($wrongDataTypes) > 0) {
51.             http_response_code(self::API_WRONG_PARAMS);
52.             return [
53.                 'response' => [
54.                     'result' => false,
55.                     'response_code' => self::API_WRONG_PARAMS,
56.                     'message' => 'Malformed request. Invalid params in \'request\'',
57.                     'data' => ['invalidDataTypesParams' => $wrongDataTypes]
58.                 ]
59.             ];
60.         } elseif(sizeof($foundKeys) < sizeof($paramsRequest)) {
61.             http_response_code(self::API_WRONG_PARAMS);
62.             return [
63.                 'response' => [
64.                     'result' => false,
65.                     'response_code' => self::API_WRONG_PARAMS,
66.                     'message' => 'Malformed request. Missing params in \'request\'',
67.                     'data' => $data
68.                 ]
69.             ];
70.         }
71.     } else {
72.         http_response_code(self::API_WRONG_PARAMS);
73.         return [
74.             'response' => [
75.                 'result' => false,
76.                 'response_code' => self::API_WRONG_PARAMS,
77.                 'message' => 'Malformed request. The key \'request\' is missing',
78.                 'data' => $data
79.             ]
80.         ];
81.     }
82.     return true;
83. }
```

Ilustración 30. Función de validación de parámetros del servicio

En este caso se especificaron 4 tipos de datos: entero (*int*), flotante (*float*), arreglo (*array*) y texto (*string*). Estos tipos de datos serán admitidos en la configuración de los parámetros de los servicios.

```
1.  const TYPE_REQUEST_INT =    'int';
2.  const TYPE_REQUEST_ARRAY =  'array';
3.  const TYPE_REQUEST_STRING = 'string';
4.  const TYPE_REQUEST_FLOAT =  'float'
```

Ilustración 31. Tipos de datos aceptados en los parámetros de la API REST

Al ser aprobada la solicitud en cuanto a estructura, autenticación y validación de datos, se procede a continuar con la lógica del servicio solicitado. En esta sección terminaría el proceso del middleware. Aunque es posible agregar más validaciones de acuerdo con lo requerido.

```
1.  const TOKEN_CREATE = [
2.     'user'                 => self::TYPE_REQUEST_INT,
3.     'password'             => self::TYPE_REQUEST_STRING
4.  ];
5.
6.  const REPORTS_POPULATION = [
7.     'date_year'            => self::TYPE_REQUEST_INT,
8.     'states'               => self::TYPE_REQUEST_ARRAY,
9.     'group_by'             => self::TYPE_REQUEST_STRING,
10. ];
```

Ilustración 32. Parámetros y tipo de datos aceptado por servicio de la API REST

Servicios

Los servicios o recursos son los procesos lógicos del negocio, estos recursos son manipulados mediante solicitudes HTTP, donde cada método tiene una función en específico para contar mejores prácticas.

Todos los servicios debe cumplir con las siguientes convenciones mientras se diseña la API REST:

- Aceptar y responder usando el formato JSON
- Usar sustantivos en el nombre del recurso en lugar de verbos.
- El método HTTP se encuentra en relación con las operaciones de un CRUD (*Create, Read, Update, Delete*).
- Devolver códigos de error en cada solicitud
- Utilizar SSL / TLS por motivos de seguridad, es imprescindible
- Almacenar datos en caché

Cada servicio creado es creado con base a las necesidades de los departamentos, ya sea de forma genérica o específica.

La Ilustración 21 muestra unos casos de uso que podrá contener la API para el consumo a través de los distintos sistemas informáticos.

En el diagrama se identifica al actor: Aplicación de cliente, asimismo, se muestran como casos de uso algunas funcionalidades que tendría la API, como: Generar token, Obtener archivo, Modificar archivo, Subir archivo, Obtener reporte, Revisar credenciales de autenticación, Revisar token, etc.

De acuerdo con el diagrama se pueden inferir lo siguiente:

- La aplicación de cliente puede Generar token, Obtener archivo, Modificar archivo, Subir archivo, Eliminar archivo, Obtener datos de resultados, Modificar datos de resultados, Subir datos de resultados, Eliminar datos de resultados y Obtener reportes.
- El caso de uso ‘Generar token’ incluye otro caso de uso que es ‘Revisar credenciales de autenticación’, en donde se valida el usuario y contraseña del cliente para autorizar el acceso.
- Y los demás casos de uso incluyen el caso de uso ‘Revisar token’. Lo que quiere decir que al procesar alguno de estos casos de uso se realizará la función de validación de token para procesar la solicitud.

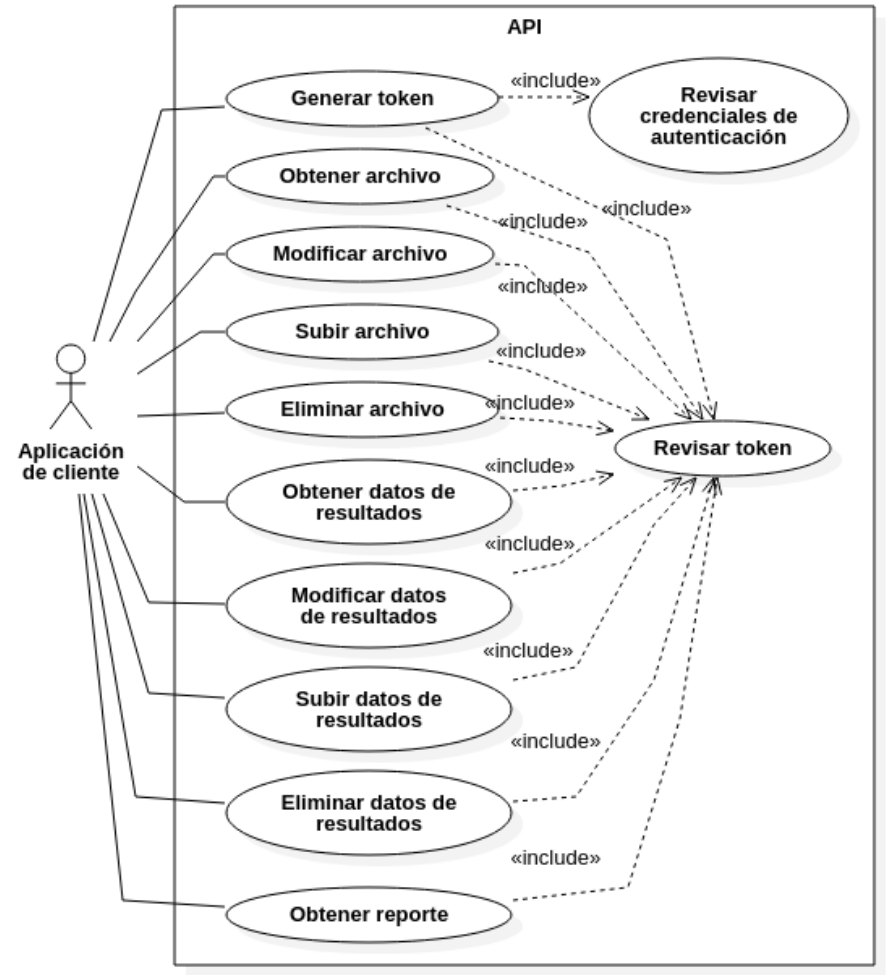


Ilustración 33. Diagrama casos de uso

Integración con otros sistemas informáticos

Cada API REST cuenta con su propio modelo de autenticación y estructura de petición y respuesta, lo que implica crear token, obtener claves de acceso, o lo que sea necesario; por lo tanto, para una integración exitosa es preciso seguir la documentación de la API.

La integración de un sistema informático a la API REST es agnóstico al lenguaje en el que están estos sistemas, siempre y cuando la solicitud y la respuesta se envié a través del protocolo HTTP.

La integración es la conexión entre una o más aplicaciones a una API donde estos sistemas intercambian datos. Las integraciones a la API favorecen los procesos y organización de los datos, donde ayuda a comunicarse entre sí sin interrupciones humanas. Además, permite automatizar sistemas mejorando la productividad.

La construcción de una integración de API desde el inicio requiere una comprensión profunda de cada sistema que requiere la conexión. Esto debido a que cada sistema maneja la respuesta del servicio de la API a su modo, realizando los procesos necesarios, sin excluir el manejo de errores, realizando pruebas de integración, así como la supervisión del rendimiento de esta.

Esta integración es agnóstica al lenguaje, sólo sería requerido conocer a detalle qué utilidad se obtendrá con la respuesta de los servicios de la API en los sistemas que la consuman para obtener el mayor beneficio de la integración. De manera que la integración a la API es posible realizar con cualquier lenguaje de programación, ya sea Python, Ruby, PHP, C#, Java, NodeJS, etc.

A continuación, se muestran algunos ejemplos para realizar una petición a la API REST.

PHP

```php
$json_data = json_encode([
  'headers' => [
            'auth' => [
                      'user_key' => 9999999,
                      'password' => 'password'
            ]
  ],
  'request' => [
            // data
  ]
]);

$curl = curl_init();
curl_setopt_array($curl, [
  CURLOPT_URL => "https://apirest.com.mx/services/{service}",
  CURLOPT_RETURNTRANSFER => true,
  CURLOPT_ENCODING => "",
  CURLOPT_MAXREDIRS => 10,
  CURLOPT_TIMEOUT => 30,
  CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
  CURLOPT_CUSTOMREQUEST => "POST",
  CURLOPT_POSTFIELDS => $json_data,
  CURLOPT_HTTPHEADER => [
            "Content-Type: application/json",
            "cache-control: no-cache"
  ]
]);

$response = curl_exec($curl);
$err = curl_error($curl);
curl_close($curl);

if($err) {
  echo "cURL Error #:" . $err;
} else {
  echo $response;
}
```

Ilustración 34. Código en PHP para realizar petición a API REST

NodeJS

```
let request = require("request");
let options = {
  method: 'POST',
  url: 'https://apirest.com.mx/services/{service}',
  headers: {
            'cache-control': 'no-cache',
            'Content-Type': 'application/json'
  },
  body: {
            headers: {
                      auth: {
                                user_key: 9999999,
                                password: 'password'
                      }
            },
            request: {
                      // data
            },
  },
  json: true
};

request(options, function (error, response, body) {
  if (error) throw new Error(response);
            console.log(body);
});
```

Ilustración 35. Código en NodeJS para realizar petición a API REST

Ruby

```
require 'uri'
require 'net/http'
body = {
  "headers" => {
            "auth" => {
                      "user_key" => 116101,
                      "password" => "password"
            }
  },
  "request" => {
            # data
  }
}

url = URI("https://apirest.com.mx/services/{service}")
http = Net::HTTP.new(url.host, url.port)
request = Net::HTTP::Post.new(url)
request["Content-Type"] = 'application/json'
request["cache-control"] = 'no-cache'
request.body = body

response = http.request(request)
puts response.read_body
```

Ilustración 36. Código en Ruby para realizar petición a API REST

Python

```
import requests
import json
url = "https://apirest.com.mx/services/{service}"

```

```
payload = {
  "headers": {
            "auth": {
                      "user_key": 999999,
                      "password": "password"
            }
  },
  "request":  {
            # data
  }
}

json_data = json.dumps(payload)

headers = {
  'Content-Type': "application/json",
  'cache-control': "no-cache",
}

response = requests.request("POST", url, data=json_data, headers =headers)
print(response.text)
```

Ilustración 37. Código en Python para realizar petición a API REST

Java

```
OkHttpClient client = new OkHttpClient();

MediaType mediaType = MediaType.parse("application/json");
RequestBody body = RequestBody.create(mediaType, json_data);
Request = new Request.Builder()
  .url("https://apirest.com.mx/services/{service}")
  .post(body)
  .addHeader("Content-Type", "application/json")
  .addHeader("cache-control", "no-cache")
  .build();

Response = client.newCall(request).execute();
```

Ilustración 38. Código en Java para realizar petición a API REST

Junto con el desarrollo de la API va de la mano la documentación ordenada y comprensible de los servicios con la que cuenta; el uso de la especificación OpenAPI permite una coordinación entre los desarrollos de *backend* y *frontend*, lo que concede una coherencia en la parte pruebas y desarrollo.
El uso de este estándar implica aplicar el enfoque *API first*, diseñar la API antes del desarrollo de código o la realización de integraciones.

En la Ilustración 39 se muestra un ejemplo del uso del estándar OpenAPI con cuatros servicios, cada uno con un método HTTP distinto y siguiendo la estructura JSON que se definió.

```yaml
openapi: 3.0.1

servers:
  # Added by API Auto Mocking Plugin
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/onericky/MITC/1.0.0

info:
  version: "1.0.0"
  title: API REST MITC
  description: Documentación para API REST

tags:
- name: token
  description: endpoint para generar token

paths:
  /token:
    post:
      tags:
      - "token"
      summary: Genera token de autenticación
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/BodyToken'
        required: true
      responses:
        200:
          description: token generado correctamente
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SuccessToken'
        400:
          $ref: '#/components/responses/BadRequest'
        401:
          $ref: '#/components/responses/Unauthorized'
        404:
          $ref: '#/components/responses/NotFound'
        500:
          $ref: '#/components/responses/ServerError'

  /file/{idFile}:
    get:
      tags:
        - "file"
      summary: regresa archivo en base64
      parameters:
        - $ref: '#/components/parameters/token'
        - name: idFile
          in: path
          description: nombre del recurso (archivo)
          required: true
          schema:
            type: string
      responses:
        200:
          description: (ok) archivo codificado en base64
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SuccessFile'
        400:
          $ref: '#/components/responses/BadRequest'
        401:
          $ref: '#/components/responses/Unauthorized'
        404:
```

```yaml
          $ref: '#/components/responses/NotFound'
        500:
          $ref: '#/components/responses/ServerError'

    delete:
      tags:
        - "file"
      summary: borrar archivo en base64
      parameters:
        - $ref: '#/components/parameters/token'
        - name: idFile
          in: path
          description: nombre del recurso (archivo)
          required: true
          schema:
            type: string
      responses:
        200:
          description: (ok) archivo eliminado
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SuccessFile'
        400:
          $ref: '#/components/responses/BadRequest'
        401:
          $ref: '#/components/responses/Unauthorized'
        404:
          $ref: '#/components/responses/NotFound'
        500:
            $ref: '#/components/responses/ServerError'

    put:
      tags:
        - "file"
      summary: actualiza archivo
      parameters:
        - $ref: '#/components/parameters/token'
        - name: idFile
          in: path
          description: nombre del recurso (archivo)
          required: true
          schema:
            type: string
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/BodyFile'
        required: true
      responses:
        200:
          description: (ok) archivo eliminado
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SuccessFile'
        400:
          $ref: '#/components/responses/BadRequest'
        401:
          $ref: '#/components/responses/Unauthorized'
        404:
          $ref: '#/components/responses/NotFound'
        500:
          $ref: '#/components/responses/ServerError'

components:
  responses:
    Unauthorized:
      description: acceso denegado
```

```yaml
    NotFound:
      description: service not found
    BadRequest:
      description: something went wrong
    ServerError:
      description: server error


parameters:
  token:
      in: header
      name: token
      description: token de autenticación
      required: true
      schema:
        type: string


schemas:
  BodyToken:
    type: object
    properties:
      headers:
        type: object
        properties:
          auth:
            type: object
            properties:
              usuario:
                type: string
                description: usuario en body
              password:
                type: string
                description: contraseña en body

  BodyFile:
    type: object
    properties:
      headers:
        type: object
        properties:
          auth:
            type: object
            properties:
              usuario:
                type: string
                description: usuario en body
              password:
                type: string
                description: contraseña en body
      request:
        type: object
        properties:
          dataBase64:
            type: string
            description: archivo encriptado en base64

  SuccessToken:
    type: object
    properties:
      response:
        type: object
        properties:
          result:
            type: boolean
            description: status
          response_code:
            type: integer
            description: HTTP response code
          message:
```

```
210.                type: string
211.                description: brief message about general API request
212.              data:
213.                type: object
214.                properties:
215.                  result:
216.                    type: boolean
217.                    description: status
218.                  response_code:
219.                    type: integer
220.                    description: HTTP response code from service
221.                  message:
222.                    type: string
223.                    description: brief message about service response
224.                  data:
225.                    type: object
226.                    properties:
227.                      token:
228.                        type: string
229.                        description: cadena de token
230.
231.
232.      SuccessFile:
233.        type: object
234.        properties:
235.          token:
236.            type: string
237.            description: token de autenticación
```

Ilustración 39. Descripción de servicios de API REST con el estándar OpenAPI

EVALUACIÓN DE LA INTERVENCIÓN

REST en un modelo de arquitectura de comunicación basada en web. Las aplicaciones REST usan el protocolo estándar de comunicación HTTP para comunicar datos desde los sistemas gestores de base de datos a aplicaciones de terceros. El desarrollo de aplicaciones de servicios basados en REST representa varias virtudes:

- El cambio de servicios REST de aprovisionamiento no requiere algún cambio en la implementación de código del lado del cliente.
- Es liviano, ya que está diseñado para consumir el servicios a través de la URI con valores en formato JSON.
- Agnóstico al lenguaje y plataforma.
- Es mucho más sencillo desarrollar nuevos servicios que SOAP.
- Está diseñado para usar el protocolo HTTP y HTTPS lo que permite integrar los mecanismos de seguridad disponibles de estos. Y utiliza los verbos HTTP.
- Sistema en capas en la arquitectura, donde trabajan juntas para construir una jerarquía que permite a crear una aplicación modular y escalable.
- El cliente y el servidor están separados el uno del otro y permite evolucionar individualmente.
- La API REST no tiene estado, por lo que las peticiones contienen todos los datos para realizarla correctamente y de forma independiente.

REST hace las aplicaciones más escalables, debido a que no tienen estado, este es uno de los principios básicos de REST. Por lo que será independiente el proceso de cada solicitud a los servicios.

Sin embargo, al no contar con estado, generalmente las solicitudes GET devuelven la misma respuesta. Lo que hace que el almacenamiento en caché sea un factor crítico para la escalabilidad y el rendimiento.

Al separar las preocupaciones de la interfaz de usuario de las preocupaciones de almacenamiento de datos, mejoramos la portabilidad de la interfaz de usuario a través de múltiples plataformas y mejoramos la escalabilidad al simplificar los componentes del servidor. (Fielding, 2000)

Además, la API REST utiliza JSON como formato de intercambio de datos. JSON es mucho más compacto y de menor tamaño en comparación con otro formato como XML. Incluso la API puede responder con distintos formatos utilizando el encabezado *Accept*.

REST instruye en que cada recurso debe ser accesible y direccionable de forma única a través de la URL, o mejor conocidos endpoints. Las URL claras permiten un control transparente de las acciones.

Así también, teniendo en cuenta el principio HATEOAS de que una API REST permite navegar por todos los servicios finales siguiendo los enlaces. HATEOAS es fundamental en la API para la concepción original de Fielding de REST.

De acuerdo con el estudio realizado por (Postman, 2021), en lo que respecta a los estilos arquitectónicos para las API, una gran mayoría de los encuestados (94%) usa REST, casi la mitad dijo que no solo usa REST, sino que “lo usa y lo ama”.

El tiempo es un obstáculo principal para la realización de una API, así como la complejidad, por lo tanto el diseño y definición de una API es primordial donde se orientará para el desarrollo de esta misma.

Sin embargo una API puede ser de gran ayuda en algún proyecto o no es necesaria como tal una sino solo un WebService o función general del sistema.

De acuerdo a la definición de la API, se realizó un cuestionario estandarizado formal en el que la recolección de la información constituye una respuesta fundamental para la evaluación del trabajo, así como para añadir un valor de acuerdo al análisis de este.

Para la aplicación del cuestionario se utilizó el software “Formularios Google” en donde se generó el instrumento y se recopilaron las respuestas en línea. Se realizó una petición a la participación del cuestionario a través de un correo con el link al instrumento. Este se encuentra en el Anexo A.

Resultados Cuestionario

Estos resultados están basados en una encuesta a 35 trabajadores del INEGI del área de TI. Cabe mencionar que la 2da pregunta es primordial para seguir con las demás preguntas del cuestionario. El análisis de las preguntas se muestran a continuación.

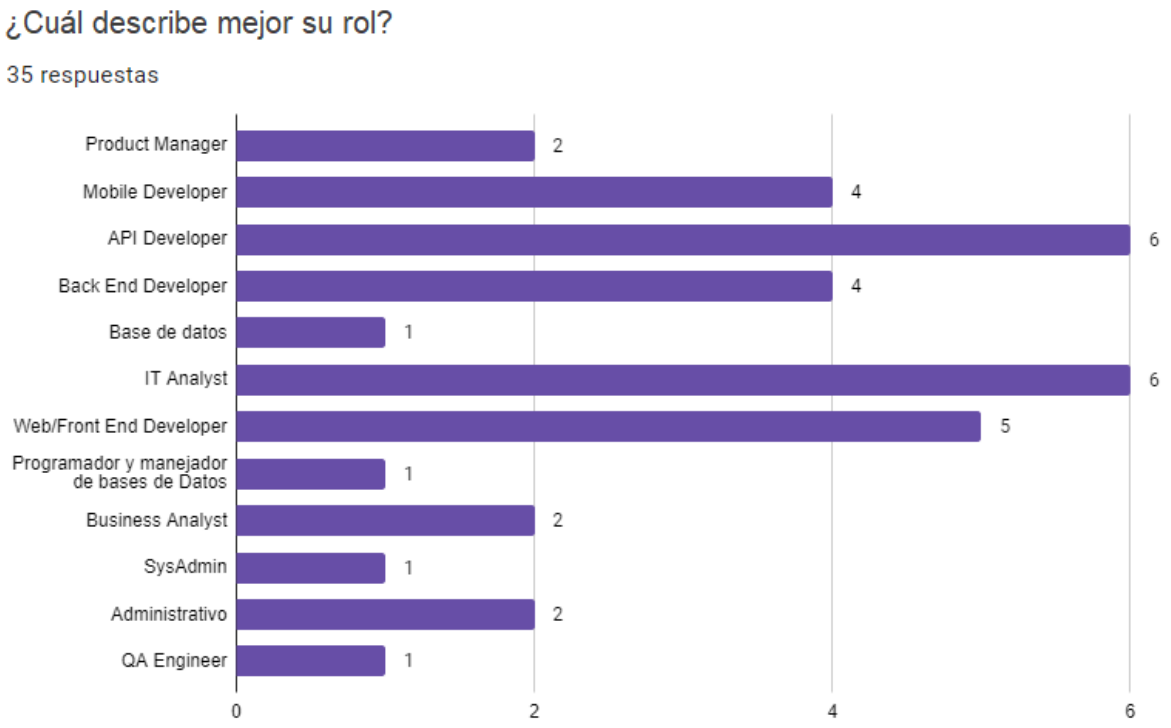


Ilustración 40. Resultados pregunta 1

Cómo se muestra en la Ilustración 40, al ser las respuestas del personal de TI, predomina el desarrollo (programador), siendo la mayor parte “API Developer”, ”Web / Front End Developer”, aunque también se encuentra “IT Analyst” como mayoría en las respuestas.

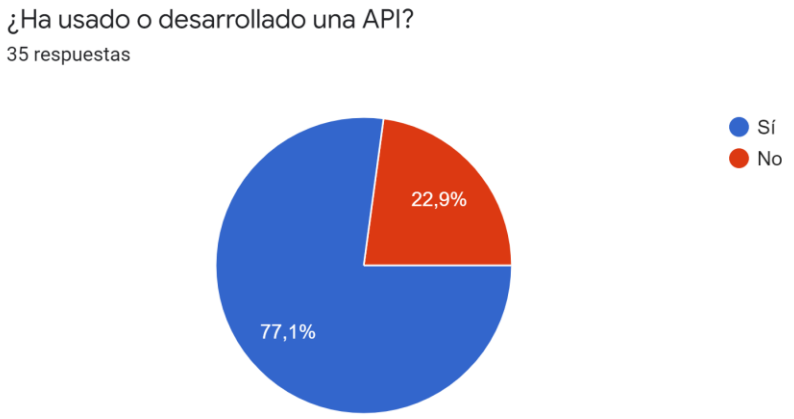


Ilustración 41. Resultados pregunta 2

Con los resultados de esta pregunta se pretendía conocer si la persona contaba con conocimiento sobre las API para así continuar con el cuestionario. El 77% de las personas contestó que sí usó o desarrolló una API por lo tanto las siguientes preguntas solo tienen 27 respuestas.

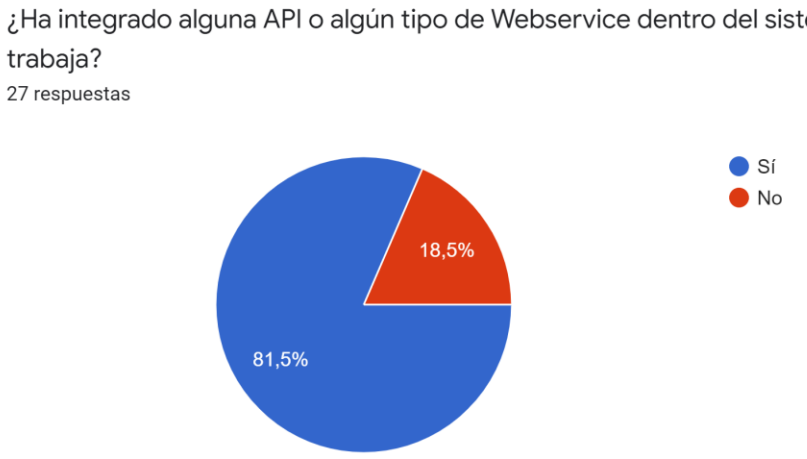


Ilustración 42. Resultados pregunta 3

De acuerdo a esta pregunta, la mayoría ha integrado una API o WebService por lo tanto, sin embargo en 18.5% restante de las personas no integraron pero sí han usado una API, ya sea

de manera implícita en un sistema o explícita a través de una herramienta.

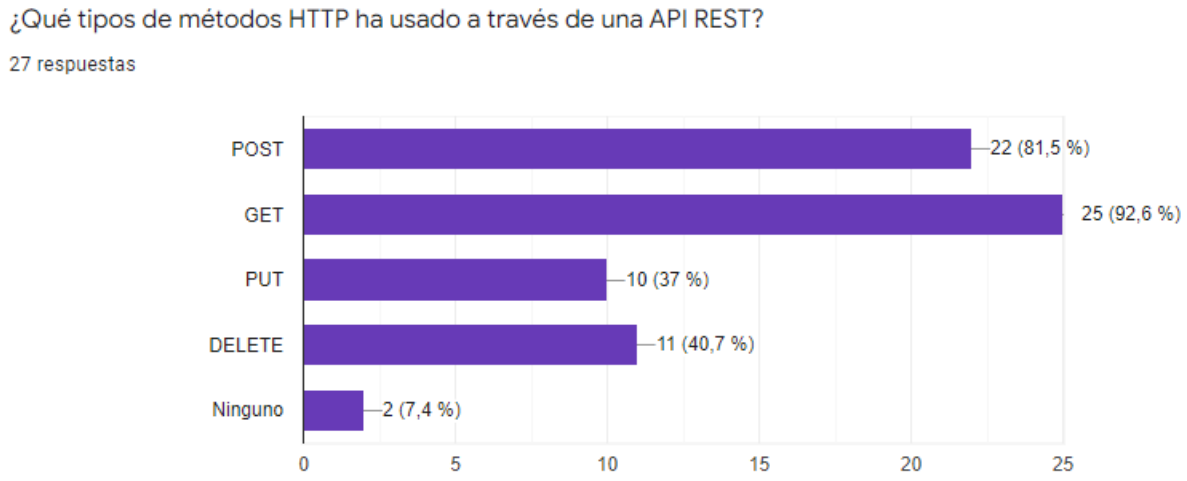


Ilustración 43. Resultados pregunta 4

Los resultados de la Ilustración 43, muestran que la mayoría de las personas ha usado el método POST y GET en donde se podría decir que son los más comunes en desarrollo, sin embargo, también el uso del PUT y DELETE es importante en una API para el tipo de servicio que se requiera.

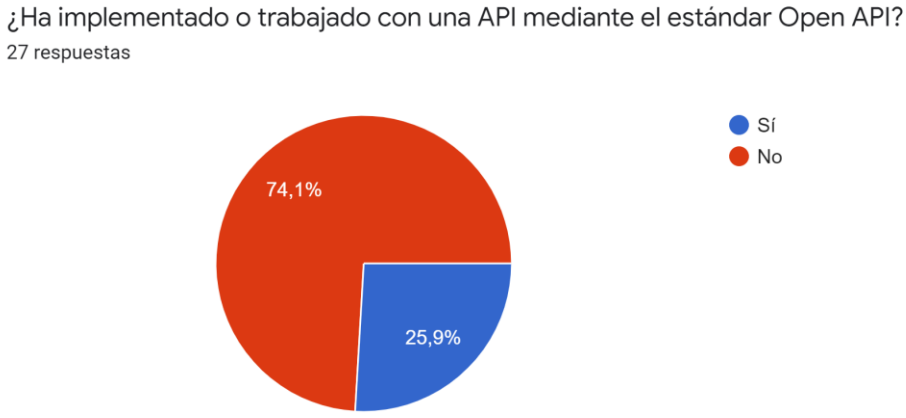


Ilustración 44. Resultados pregunta 5

Acorde a la Ilustración 44, el 74% de las personas contestaron que “No han implementado o trabajado con el estándar Open API”, por lo que está observación es significante en el estudio.

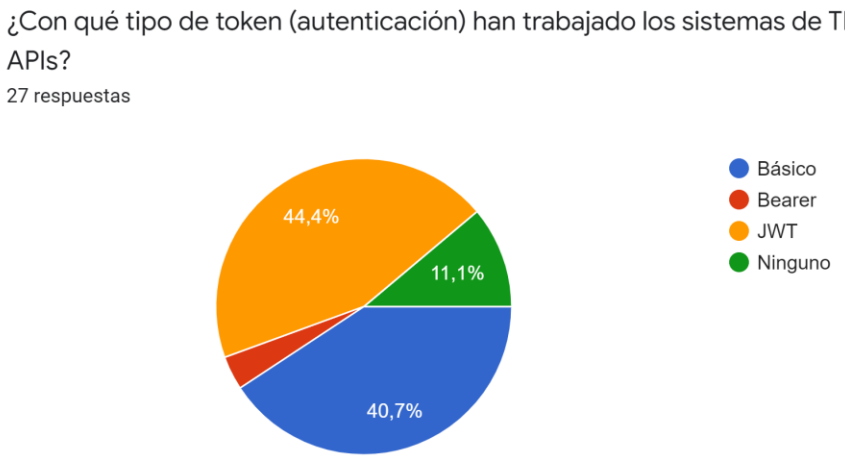


Ilustración 45. Resultados pregunta 6

El token que utilizan más las personas es el Básico y JWT, lo que indica que sí utilizan token en los sistemas de TI.

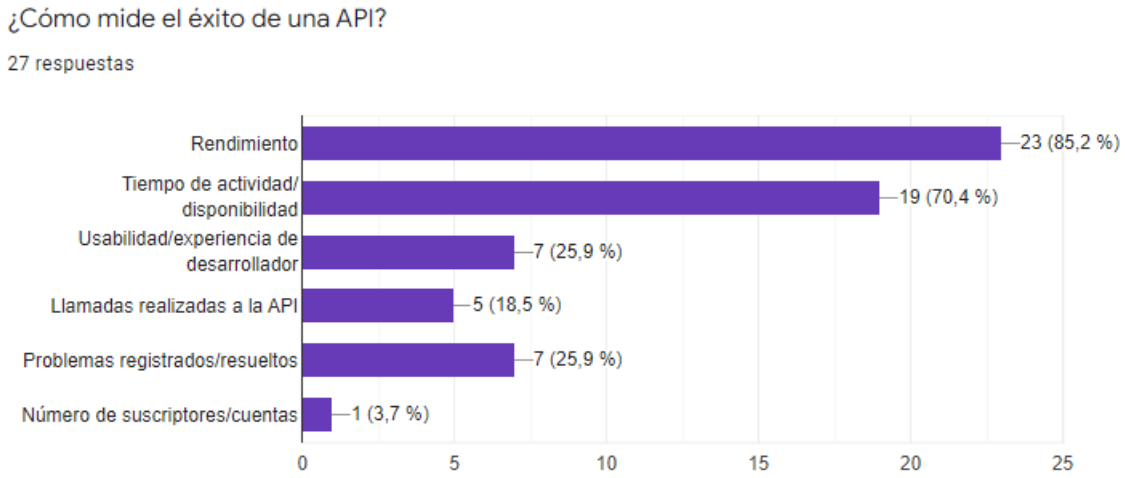


Ilustración 46. Resultados pregunta 7

Con la Ilustración 46, se muestra que el “Rendimiento” es el principal factor para medir el éxito de una API, siguiendo el factor de “Tiempo de actividad/disponibilidad” y dejando por última el “Número de suscriptores/cuentas”.

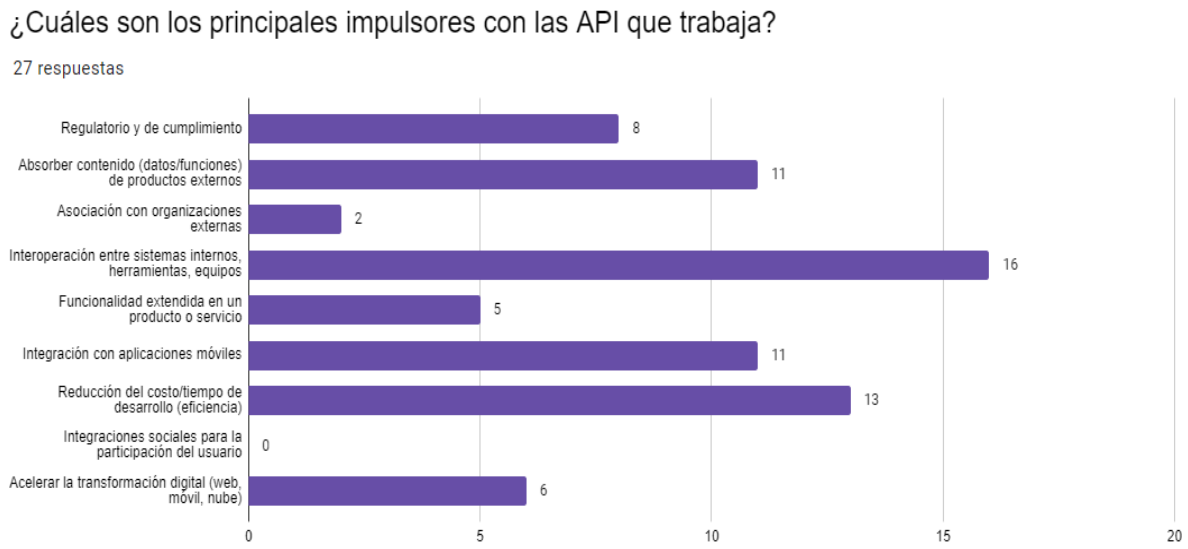


Ilustración 47. Resultados pregunta 8

Las respuestas de esta pregunta, muestra que la “Interoperación entre sistemas internos, herramientas, equipo” fue la mayor elegida con un poco más del 50% de selecciones, siguiendo la “Reducción del costo/tiempo de desarrollo (eficiencia)”.

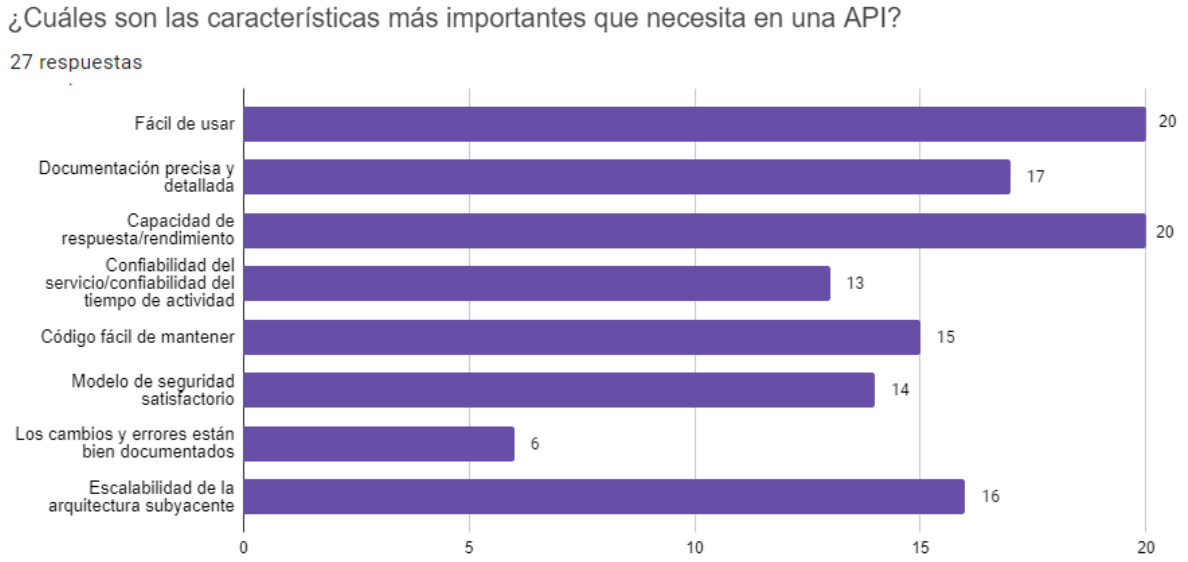


Ilustración 48. Resultados pregunta 9

Con base a la Ilustración 48, según los encuestados las características más importantes que necesita una API es “Fácil de usar” y “Capacidad de respuesta/rendimiento”, y siendo la menos importante “Los cambios y errores están bien documentados”.

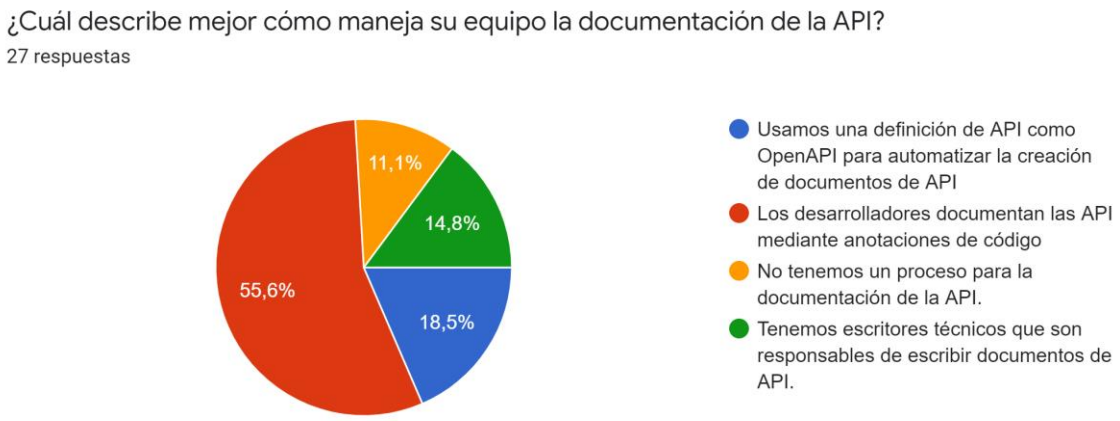


Ilustración 49. Resultados pregunta 10

Un poco más de la mitad de las personas respondieron que “Los desarrolladores documentan las API mediante las anotaciones de código”, siguiendo con un 18.5% “Usamos una definición de API como OpenAPI para automatizar la creación de documentos de API.

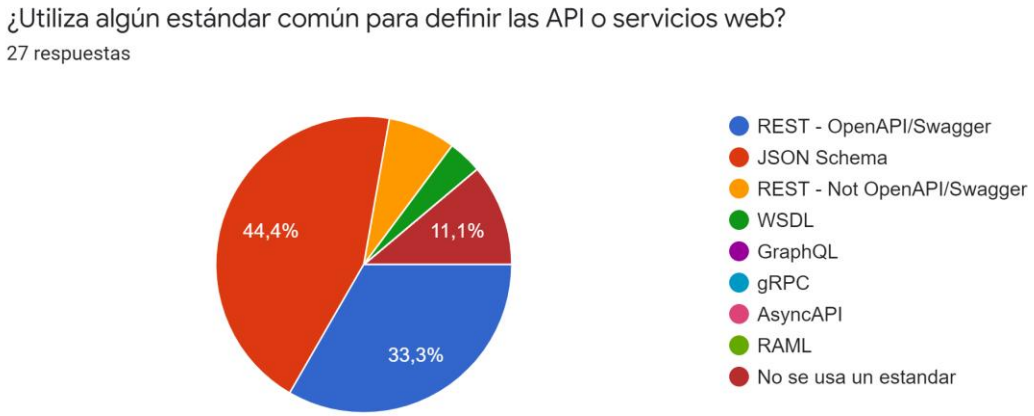


Ilustración 50. Resultados pregunta 11

En la Ilustración 50 se muestra que “REST - OpenAPI/Swagger” y “JSON Schema” dominan en la respuesta, siendo un poco mayor está última.

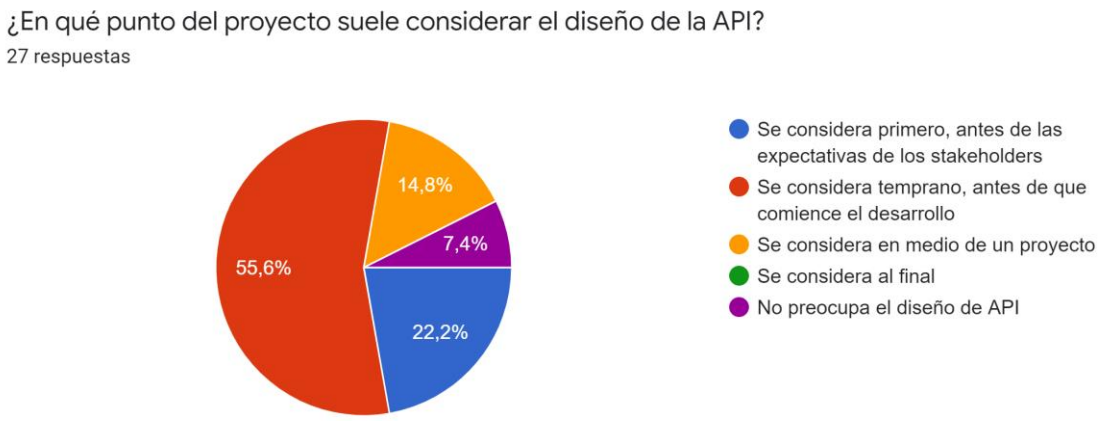


Ilustración 51. Resultados pregunta 12

Con base a las respuestas en la Ilustración 51, el 55.6% seleccionó “Se considera temprano, antes de que comience el desarrollo”, por lo que es una opción muy aceptable y sin embargo nunca se considera al final y se ve en las respuestas que esa opción no fue elegida.

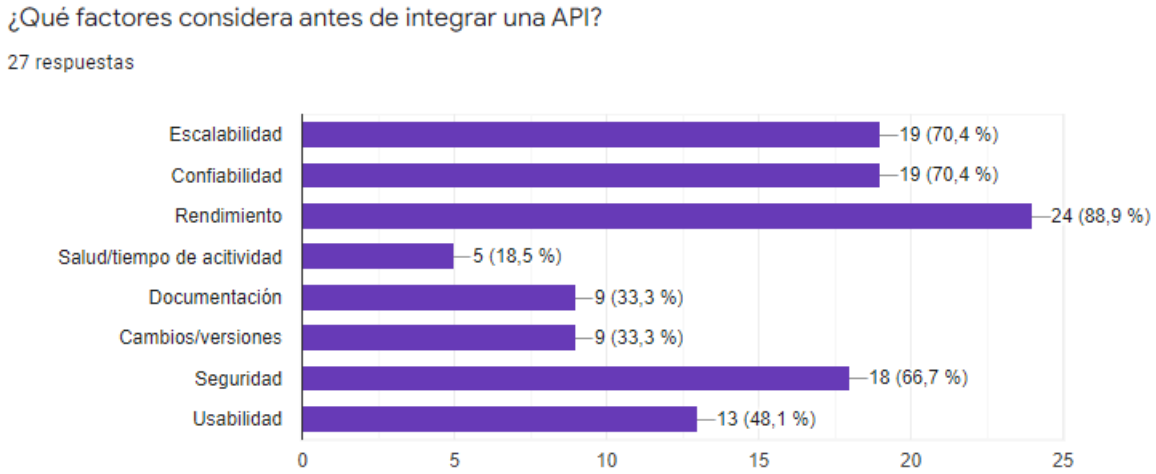


Ilustración 52. Resultados pregunta 13

Con base a la Ilustración 52, el factor más importante para los encuestados fue el “Rendimiento”, siguiendo la “Escalabilidad”, “Confiabilidad” con un 70% y “Seguridad” con un poco más del 65% de elección.

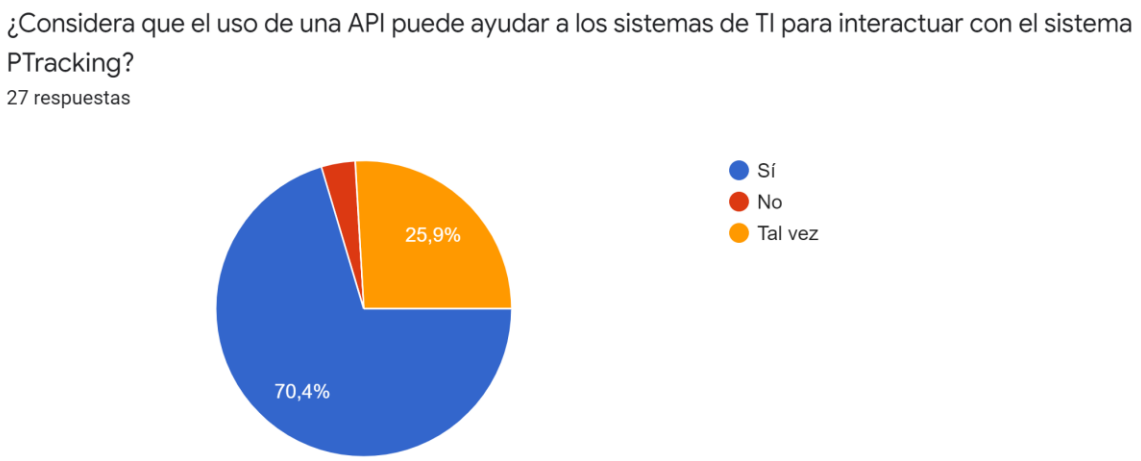


Ilustración 53. Resultados pregunta 14

Con un 70% de respuestas afirmativas y 25% de decisión media, los encuestados consideran que el uso de una API podría ayudar a los sistemas de TI con el PTracking, mientras que sólo un 3.7% piensa que no.

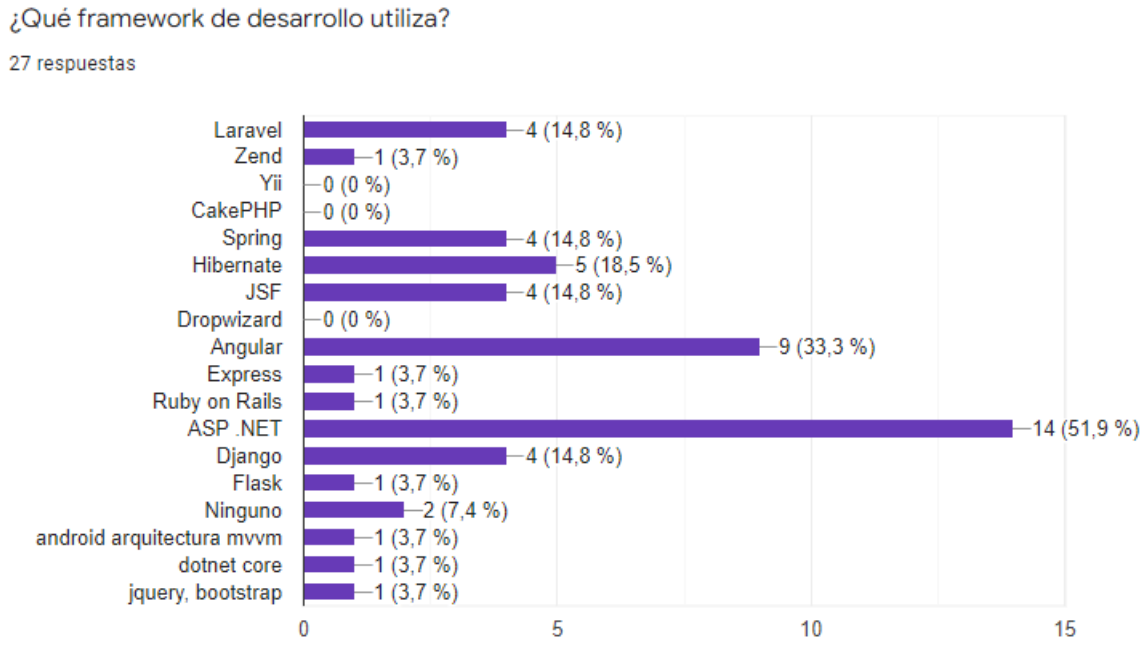


Ilustración 54. Resultados pregunta 15

Observando los resultados en la Ilustración 54 se usan distintos frameworks aunque predomina “ASP .NET” con un 51.9% y “Angular” con un 33.33%.

Discusión de resultados obtenidos respecto a objetivos específicos

De acuerdo con los tres objetivos específicos y las preguntas de investigación que se plantean al inicio del trabajo práctico, se muestran los resultados obtenidos.

Objetivo específico 1. Determinar una arquitectura de API escalable que permita la evolución para generar nuevos servicios que requiera la institución.

Resolución: Con base a la investigación y desarrollo del trabajo práctico se define una API REST la cual permite varios beneficios como la escalabilidad gracias a que se tiene separado el cliente y servidor, además de la independencia de los servicios. Por lo tanto se cumple con el objetivo planteado.

Pregunta de investigación 1. ¿Cómo será la funcionalidad de los métodos de alto nivel que contendrá la API para la distribución de la información?

Resolución: Estos métodos o servicios son manipulados mediante solicitudes HTTP, ya que la arquitectura REST se apoya totalmente en el estándar HTTP. Así pues los métodos HTTP están relacionados con las operaciones CRUD, como se muestra en la Tabla 1.

Objetivo específico 2. Establecer una solución para facilitar el manejo de la información en la institución entre sus áreas.

Resolución: A través de los servicios definidos en la API, el manejo y transferencia de información sería más eficiente puesto que con la API conectada al sistema PTracking el intercambio de información será estandarizada y los sistemas de TI tendrán una mejor conectividad a este sistema centralizado por lo que el objetivo planteado se cumple.

Pregunta de investigación 2. ¿Mejorará la accesibilidad, los tiempos de entrega, la seguridad y la fiabilidad en la distribución de la información entre los distintos departamentos de la institución?

Resolución: Sí, la API permite la comunicación de manera rápida y sencilla desde cualquier sistema, además, los procesos internos de la institución se desarrollan de manera estandarizada y centralizada dentro de la API, lo cual asegura que la información se mantendrá íntegra y disponible en los servicios de esta, asimismo, se establecen distintos procesos como: autenticación, autorización, limitación de peticiones y monitoreo dentro del middleware y API Gateway lo que la hace segura.

Objetivo específico 3. Determinar si la API propuesta para la institución es funcional a través de sus distintos sistemas de TI.

Resolución: La API REST tiene un estilo cliente-servidor. Los sistemas informáticos de la institución actúan como clientes de la API para acceder al sistema y así la integración de los clientes es agnóstica al lenguaje de programación. De acuerdo a la pregunta de la Ilustración 54, la mayoría de los encuestados usan ASP .NET por lo tanto, como se muestra en la Ilustración 42, la conexión a la API no es una complejidad y será útil mientras se tenga una documentación desarrollada de manera ordenada y precisa para los servicios y autenticación.

Además siguiendo la definición de la API en este trabajo práctico mantendrá una seguridad a través del middleware en el desarrollo, asimismo, en la escalabilidad de la API podrá aumentar en servicios o cambios mayores a través de un versionamiento.

Pregunta de investigación 3. ¿Cómo influirá la API dentro de los sistemas de TI con los que cuenta la institución?

Resolución: La integración de la API con los sistemas de información y el sistema PTracking permiten ahorros en tiempo y costes de desarrollo; además de acceder a los recursos de manera controlada y segura por medio de accesos y autorizaciones establecidos.

El diseño de la API es de una manera accesible debido a que el formato de texto JSON con la que se define para las peticiones y respuestas es sencillo para el intercambio de datos. Y cada servicio que se cree tendrá distintos códigos de estado, como se muestra en la Tabla 4, 5 y 6.

En el desarrollo de la API se define trabajar con el estándar OpenAPI para establecer un buen diseño de la API, documentación completa de servicios, parámetros requeridos y códigos de respuesta por lo tanto la influencia de una API con este estándar será valiosa para la estandarización en los procesos y mejora de los sistemas informáticos.

Recomendaciones

Para cualquier proyecto de TI, ya sea software, gestión de base de datos, integración con la nube, etc., siempre es importante seguir las buenas prácticas, en este caso con la API se muestran algunos puntos.

- Contar con la propiedad HATEOAS
- Proteger la API a través de distintos sistemas de seguridad
- Probar la API apoyándose de alguna de las distintas herramientas disponibles, como Postman, Firecamp, Testfully, Insomnia, Swagger o alguna otra alternativa
- Documentar los recursos de forma correcta y actualizada, esto junto con el estándar OpenAPI. En este punto no es tanto documentar lo que hace tal línea de código sino describir lo que hace cada *endpoint* de la API junto con los parámetros que recibe y lo que retorna.
- Otro punto importante es tener una bitácora de errores y operaciones, es decir, guardar toda la información relevante como la hora, parámetros, *endpoint*, de la manera que se pueda monitorear el proyecto y mantener segura y lo más estable posible.
- Manejar correctamente los códigos de estado, ya que no es para nada correcto siempre devolver un código 200 cuando dentro de la API falló alguna operación.

Con el paso del tiempo surgirán nuevos cambios importantes, nuevas tecnologías, nuevos métodos, nuevas funciones; en donde estas pueden modificar el comportamiento de la API así como un estándar en el formato de las respuestas, para evitar dificultades de adaptación, siempre es mejor versionar esos tipos de cambios considerables en la API. Con esto se podrá mantener la compatibilidad de las distintas versiones con los clientes.

A partir de una API monolítica que cuente con gran variedad de servicios y complejidad es factible transformar en microservicios individuales más pequeños mejor acoplados para hacer que la aplicación sea más flexible, altamente disponible y eficiente en el uso de los recursos.

CONCLUSIONES

Durante el desarrollo de este trabajo práctico se encuentran diversos conceptos que conlleva una API, desde los tipos de API que existen, arquitectura, flujo, formato de datos, etc.

Con base al uso de la API dentro del INEGI en el intercambio de información entre los distintos sistemas informáticos de sus áreas y el sistema PTracking, una API REST representa múltiples beneficios en cuánto a independencia, esto permite que sea consumida por infinidad de clientes sea indiferente la naturaleza de estos.

La API proporciona escalabilidad, fiabilidad y una portabilidad factible, ya que aísla al cliente, y asimismo dicha separación entre el cliente y servidor posibilita migrar a otro servidor, bases de datos o tecnología de manera transparente siempre y cuando la definición de la API con el estándar OpenAPI se siga cumpliendo.

La definición de la API REST con el estándar OpenAPI mejorará la funcionalidad en la integración de los diferentes sistemas de TI de los departamentos de estadística y geografía del instituto en razón de que el uso del estándar permite una comunicación visual y con ello tolera que los equipos de desarrollo trabajen de manera separada. Adicionalmente, el uso de una API con el estándar transforma la manera de afrontar las integraciones y personalización en un modo rígido con altos costes.

Conforme a los resultados que se obtuvieron de la aplicación del cuestionario, la mayoría no estuvo en contacto con el estándar OpenAPI por lo que llevaría un reto en la creación pero mejoraría significativamente el uso de la API en los sistemas, así también aportará un mejor mantenimiento y el desarrollo continuo.

GLOSARIO

API. Conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. API significa interfaz de programación de aplicaciones.

Amenaza cibernética. Toda aquella acción que aprovecha una vulnerabilidad para atacar o invadir un sistema informático.

Autenticación. Capacidad de demostrar que un usuario o una aplicación es realmente quién dicha persona o aplicación asegura ser.

Autorización. Protector de los recursos importantes de un sistema, ya que limita el acceso solamente a los usuarios autorizados y a sus aplicaciones.

Balanceo de carga. Distribución de la actividad de procesamiento y la comunicación de manera uniforme a través de una red informática a fin de que no exista algo sobrecargado.

Base de datos. Recopilación organizada de información o datos estructurados, que normalmente se almacena de forma electrónica en un sistema informático.

Big data. Datos que son tan grandes o complejos que no pueden manejarse con los métodos tradicionales de procesamiento. En general, también se lo conoce por sus tres V: volumen, variedad y velocidad.

Caché. Es una capa de almacenamiento de datos de alta velocidad que almacena un subconjunto de datos, normalmente transitorios, de modo que las solicitudes futuras de dichos datos se atienden con mayor rapidez que si se debe acceder a los datos desde la ubicación de almacenamiento principal.

Código de estatus. Parte de la respuesta devuelta por el servidor cuando un cliente llama a una URL. Con la ayuda de un código de estatus, el servidor indica al cliente si la solicitud se ha procesado correctamente o si se ha producido un error.

E-commerce. Método de compraventa de bienes, productos o servicios valiéndose de internet como medio.

Encriptación. Procedimiento de seguridad que consiste en la alteración, mediante algoritmos, de los datos que componen un archivo.

Git. Sistema de control de versiones distribuido de código abierto, permite a los desarrolladores descargar un software, realizar cambios y subir la versión que han modificado.

Gobernanza. Conjunto de procesos, funciones, políticas, normas y mediciones que garantizan el uso eficaz y eficiente de la información con el fin de ayudar a una empresa a cumplir sus objetivos.

GraphQL. Lenguaje de consulta y un tiempo de ejecución del servidor para las interfaces de programación de aplicaciones (API); su función es brindar a los clientes exactamente los datos que solicitan y nada más.

GSBPM. Estándar internacional que propone una estructura de procesos y subprocesos del modelo de producción de estadísticas, por sus siglas en inglés: Generic Statistical Business Process Model.

HTTP. Es el nombre de un protocolo el cual permite realizar una petición de datos y recursos, de sus siglas en inglés: Hypertext Transfer Protocol.

Interoperabilidad. Capacidad de las plataformas digitales para intercambiar información, ya sean datos, documentos u otros objetos digitales, de manera uniforme y eficiente.

IoT. Describe la red de objetos físicos (cosas) que incorporan sensores, software y otras tecnologías con el fin de conectar e intercambiar datos con otros dispositivos y sistemas a través de Internet, de sus siglas en inglés: Internet of Things.

Java. Tipo de lenguaje de programación y plataforma informática, permite que los desarrolladores de aplicaciones escriban el programa una sola vez y lo ejecuten en cualquier dispositivo.

JSON. Es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones.

JWT. Es un estándar abierto basado en JSON para crear un token que sirva para enviar datos entre aplicaciones o servicios y garantizar que sean válidos y seguros.

Microservicios. Son un enfoque arquitectónico y organizativo para el desarrollo de software donde el software está compuesto por pequeños servicios independientes que se comunican a través de API bien definidas.

Middleware. Se refiere a un sistema de software que ofrece servicios y funciones comunes para las aplicaciones. En general, el middleware se encarga de las tareas de gestión de datos, servicios de aplicaciones, mensajería, autenticación y gestión de API.

MPEG. Marco conceptual de referencia donde se estandariza la producción de información estadística y geográfica en 8 fases, por sus siglas: Modelo del Proceso Estadístico y Geográfico.

NAS. Tecnología de almacenamiento de red que brinda la posibilidad de concentrar y almacenar todo tipo de contenidos en un único dispositivo, para que siempre se encuentren disponibles para quién lo solicite a través de la res, independientemente del estado de las computadoras que la integran.

NoSQL. Se refiere a tipos de bases de datos no relacionales, y estas bases de datos almacenan datos en un formato diferente al de las tablas relacionales.

PHP. Lenguaje de programación destinado a desarrollar aplicaciones para la web y crear páginas web, favoreciendo la conexión entre los servidores y la interfaz de usuario.

Protocolo. Sistema de normas que regulan la comunicación entre dos o más sistemas que se transmiten información a través de diversos medios físicos.

Protocolos de comunicación. Conjunto de normas que están obligadas a cumplir todas las máquinas y programas que intervienen en una comunicación de datos entre ordenadores sin las cuales la comunicación resultaría caótica y por tanto imposible.

PTracking. Sistema de registro de evidencias del INEGI.

Python. Lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. Se trata de un lenguaje interpretado, las aplicaciones escritas en Python se ejecutan directamente por el ordenador utilizando un programa denominado interpretador.

REST. Interfaz que conecta varios sistemas basados en el protocolo HTTP y sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos.

Ruby. Lenguaje de programación, que está principalmente orientado a objetos pero también puede ser programación funcional.

Servidor. Sistema que proporciona recursos, datos, servicios o programas a otros ordenadores, conocidos como clientes, a través de una red.

Sistema de control de versiones. Herramienta de software que ayuda a realizar un seguimiento de los cambios realizados en el código a lo largo del tiempo. Si se comete un error los desarrolladores pueden ir hacia atrás y comparar las versiones anteriores del código.

SOAP. Protocolo de aplicaciones fundacional basado en formato de mensaje XML utilizado en interacciones de servicios web, por sus siglas en inglés: Simple Object Access Protocol.

Software. Programa o conjunto de programas que constituyen la parte lógica de un sistema informático, que permite el desarrollo de procedimientos automáticos y rutinas de procesamiento de datos con el objetivo de realizar aplicaciones informáticas.

SSL. Tecnología estándar para mantener segura una conexión a Internet, así como para proteger cualquier información confidencial que se envía entre dos sistemas.

TI. Por sus siglas, Tecnología de la Información, es el uso de cualquier computadora, almacenamiento, redes, hardware, software, internet, es decir, a todo lo relacionado con la tecnología informática.

TLS. Protocolo que hace uso de certificados digitales para establecer comunicaciones seguras a través de Internet.

Token. Conjunto de caracteres que sirve como firma cifrada que permite a la API identificar la autenticación.

URI. Cadena de caracteres que se utilizan para identificar un recurso o un nombre en internet. Su propósito es permitir la interacción entre diferentes recursos en Internet y otro tipo de red.

WSDL. Notación XML para describir un servicio web. Una definición WSDL indica a un cliente cómo componer una solicitud de servicio web y describe la interfaz que proporciona el proveedor del servicio web.

XML. Archivo de texto sin formato que utiliza una serie de etiquetas personalizadas con la finalidad de describir tanto la estructura como otras características del documento.

BIBLIOGRAFÍA

Abiteboul, S., & Stoyanovich, J. (2015). *Data, Responsibly – ACM SIGMOD Blog*. http://wp.sigmod.org/?p=1900

Amazon Web Services, I. (2019). *Administración de API: Servicios, prácticas recomendadas y herramientas de API*. https://aws.amazon.com/es/api-gateway/api-management/

Ashby, D., & Jensen, C. T. (2018). *APIs for dummies*.

Atlassian. (2019). *Software de control de versiones: descripción general | Bitbucket*. https://bitbucket.org/product/es/version-control-software

Biehl, M. (2015). *API Architecture: The Big Picture for Building APIs*.

Boehm, B. W. (1988). A Spiral Model of Software Development and Enhancement. *Computer*, *21*(5), 61–72. https://doi.org/10.1109/2.59

De, B. (2017). API Management. In *API Management*. Apress. https://doi.org/10.1007/978-1-4842-1305-6

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*.

Frisendal, T. (2018). Visual Design of GraphQL Data. In *Visual Design of GraphQL Data*. Apress. https://doi.org/10.1007/978-1-4842-3904-9

Google Cloud. (2021a). *Compila un sistema de telemetría de ubicación geográfica escalable con la API de Google Maps*. https://cloud.google.com/solutions/scalable-geolocation-telemetry-system-using-maps-api?hl=es-419

Google Cloud. (2021b). *Errores | API de Cloud | Google Cloud*. https://cloud.google.com/apis/design/errors?hl=es-419

Haaranen, L., & Lehtinen, T. (2015). Teaching Git on the Side-Version Control System as a Course Platform. *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. http://dx.doi.org/10.1145/2729094.2742608

Henning, M. (n.d.). *API Design Matters - ACM Queue*. Retrieved March 19, 2020, from https://queue.acm.org/detail.cfm?id=1255422

Humphrey, W. (2000). *The Personal Software Process (PSP)*. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5283

INEGI. (2019). *¿Qué es el MPEG?* https://extranet.inegi.org.mx/calidad/wp-content/uploads/2018/12/Que%20es%20el%20MPEG.pdf

Introduction to GraphQL | GraphQL. (n.d.). Retrieved April 5, 2020, from https://graphql.org/learn/

Kin Lane. (2020). *SOAP API: What Is a SOAP API and How Does It work? [API 101] | Postman Blog*. https://blog.postman.com/soap-api-definition/

Lakshmiraghavan, B. (2013). Encryption and Signing. In *Pro ASP.NET Web API Security* (pp. 103–117). Apress. https://doi.org/10.1007/978-1-4302-5783-7_6

Masashi Narumoto. (2018). *Guía de diseño de una API - Best practices for cloud applications | Microsoft Docs*. https://docs.microsoft.com/es-es/azure/architecture/best-practices/api-design

Meng, M., Steinhardt, S., & Schubert, A. (2019). *How Developers Use API Documentation: An Observation Study - Special Interest Group on Design of Communication*. https://sigdoc.acm.org/cdq/how-developers-use-api-documentation-an-observation-study/

Mihaly, F. (2011). *Writing helpful API documentation | The Amiable API*. https://theamiableapi.com/2011/11/01/api-design-best-practice-write-helpful-documentation/

Miller, D., Whitlock, J., Gardiner, M., Ralphson, M., Ratovsky, R., & Sarid, U. (2021, February 15). *OpenAPI Specification v3.1.0 | Introduction, Definitions, & More*. https://spec.openapis.org/oas/latest.html

Mulloy, B. (2012). *Web API Design: Crafting Interfaces that Developers Love*. Independently published.

Postman, Inc. (2021). *2021 State of the API Report*. 2021 State of the API Report. https://www.postman.com/state-of-api/

Preibisch, S. (2018). API Development. In *API Development*. Apress. https://doi.org/10.1007/978-1-4842-4140-0

Real Academia Española. (2020). *permiso | Definición | Diccionario de la lengua española | RAE - ASALE*. https://dle.rae.es/permiso

Red Hat. (2017). *¿Qué es una API?* https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

Red Hat. (2019a). *¿Cuál es la función de una puerta de enlace de API?* https://www.redhat.com/es/topics/api/what-does-an-api-gateway-do

Red Hat. (2019b). *¿Qué es la seguridad de las API?* https://www.redhat.com/es/topics/security/api-security

Siriwardena, P. (2020). Advanced API Security. In *Advanced API Security*. Apress. https://doi.org/10.1007/978-1-4842-2050-4

Tulach, J. (2008). Practical API design: Confessions of a java framework architect. In *Practical API Design: Confessions of a Java Framework Architect*. Apress Media LLC. https://doi.org/10.1007/978-1-4302-0974-4

UNECE, Eurostat, & OECD. (2009). *Generic Statistical Business Process Model*. http://www.unece.org/stats/documents/2009.03.metis.htm

Varga, E. (2016). Creating Maintainable APIs. In *Creating Maintainable APIs*. Apress. https://doi.org/10.1007/978-1-4842-2196-9

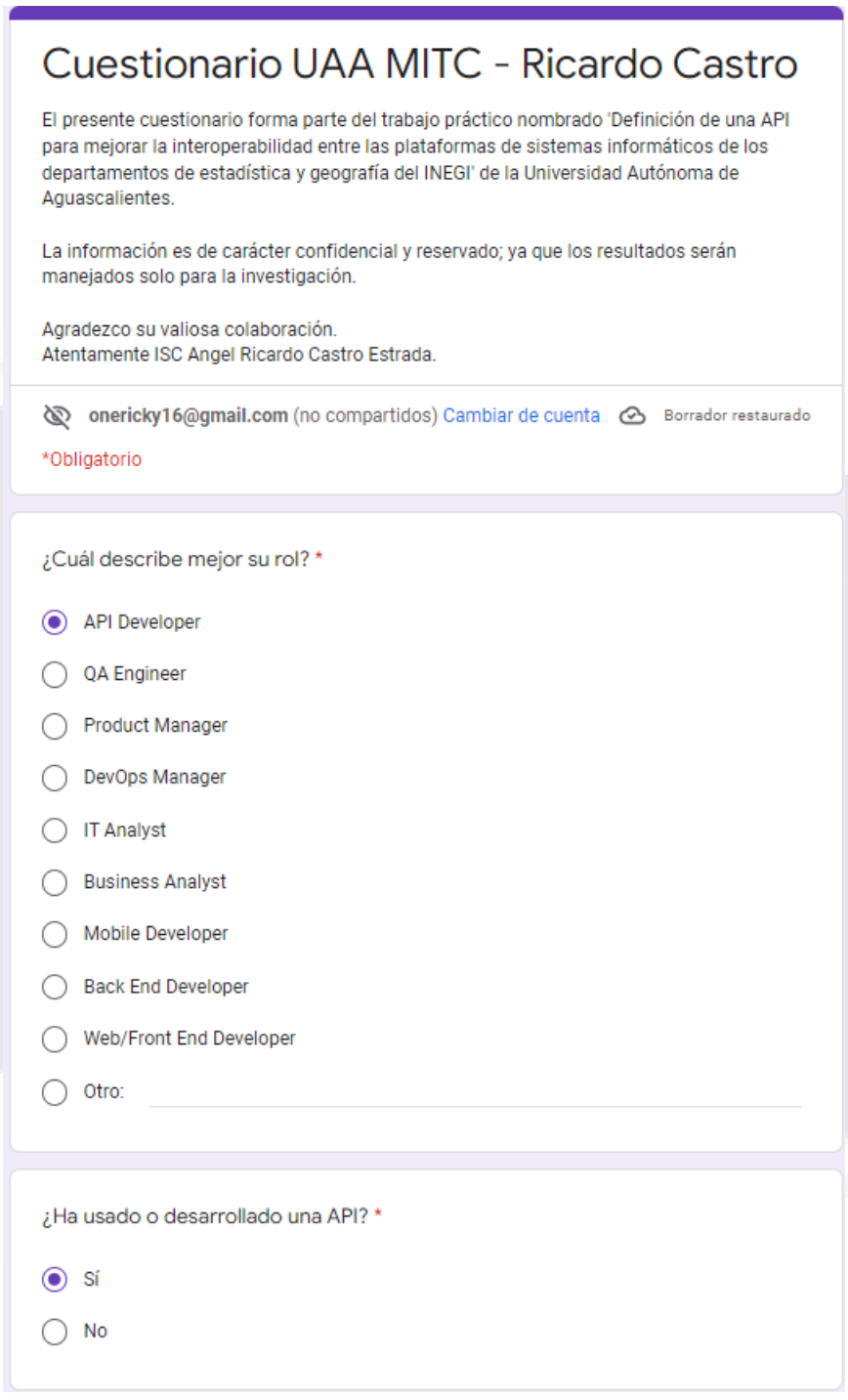
Vijayakumar, T. (2018a). API Security. In *Practical API Architecture and Development with Azure and AWS* (pp. 97–132). Apress. https://doi.org/10.1007/978-1-4842-3555-3_5

Vijayakumar, T. (2018b). Practical API Architecture and Development with Azure and AWS. In *Practical API Architecture and Development with Azure and AWS*. Apress. https://doi.org/10.1007/978-1-4842-3555-3

Yii Framework. (2020). *Servicios Web RESTful: Gestión de versiones | Guía Definitiva de Yii 2.0 | Yii PHP Framework*. https://www.yiiframework.com/doc/guide/2.0/es/rest-versioning

ANEXOS

Anexo A. Cuestionario implementado en línea mediante el software “Formularios Google”



Cuestionario UAA MITC - Ricardo Castro

El presente cuestionario forma parte del trabajo práctico nombrado 'Definición de una API para mejorar la interoperabilidad entre las plataformas de sistemas informáticos de los departamentos de estadística y geografía del INEGI' de la Universidad Autónoma de Aguascalientes.

La información es de carácter confidencial y reservado; ya que los resultados serán manejados solo para la investigación.

Agradezco su valiosa colaboración.
Atentamente ISC Angel Ricardo Castro Estrada.

onericky16@gmail.com (no compartidos) Cambiar de cuenta Borrador restaurado

*Obligatorio

¿Cuál describe mejor su rol? *

- (•) API Developer
- ( ) QA Engineer
- ( ) Product Manager
- ( ) DevOps Manager
- ( ) IT Analyst
- ( ) Business Analyst
- ( ) Mobile Developer
- ( ) Back End Developer
- ( ) Web/Front End Developer
- ( ) Otro: _____

¿Ha usado o desarrollado una API? *

- (•) Sí
- ( ) No

Cuestionario UAA MITC - Ricardo Castro

onericky16@gmail.com (no compartidos) Cambiar de cuenta

*Obligatorio

¿Ha integrado alguna API o algún tipo de Webservice dentro del sistema informático con el que trabaja? *

- ◯ Sí
- ◯ No

¿Qué tipos de métodos HTTP ha usado a través de una API REST? *

- ☐ POST
- ☐ GET
- ☐ PUT
- ☐ DELETE
- ☐ Ninguno

¿Ha implementado o trabajado con una API mediante el estándar Open API? *

- ◯ Sí
- ◯ No

¿Con qué tipo de token (autenticación) han trabajado los sistemas de TI para la integración con APIs? *

- ◯ Básico
- ◯ Bearer
- ◯ JWT
- ◯ Ninguno

¿Cómo mide el éxito de una API? *

- ☐ Rendimiento
- ☐ Tiempo de actividad/disponibilidad
- ☐ Usabilidad/experiencia de desarrollador
- ☐ Llamadas realizadas a la API
- ☐ Problemas registrados/resueltos
- ☐ Número de suscriptores/cuentas
- ☐ Otro: _____

¿Cuáles son los principales impulsores con las API que trabaja? *

- [ ] Regulatorio y de cumplimiento
- [ ] Absorber contenido (datos/funciones) de productos externos
- [ ] Asociación con organizaciones externas
- [ ] Interoperación entre sistemas internos, herramientas, equipos
- [ ] Funcionalidad extendida en un producto o servicio
- [ ] Integración con aplicaciones móviles
- [ ] Reducción del costo/tiempo de desarrollo (eficiencia)
- [ ] Integraciones sociales para la participación del usuario
- [ ] Acelerar la transformación digital (web, móvil, nube)
- [ ] Otro: _____

¿Cuáles son las características más importantes que necesita en una API? *

- [ ] Fácil de usar
- [ ] Documentación precisa y detallada
- [ ] Capacidad de respuesta/rendimiento
- [ ] Confiabilidad del servicio/confiabilidad del tiempo de actividad
- [ ] Código fácil de mantener
- [ ] Modelo de seguridad satisfactorio
- [ ] Los cambios y errores están bien documentados
- [ ] Escalabilidad de la arquitectura subyacente
- [ ] Otro: _____

¿Cuál describe mejor cómo maneja su equipo la documentación de la API? *

- ◯ Usamos una definición de API como OpenAPI para automatizar la creación de documentos de API
- ◯ Los desarrolladores documentan las API mediante anotaciones de código
- ◯ No tenemos un proceso para la documentación de la API.
- ◯ Tenemos escritores técnicos que son responsables de escribir documentos de API.

¿Utiliza algún estándar común para definir las API o servicios web? *

- ◯ REST - OpenAPI/Swagger
- ◯ JSON Schema
- ◯ REST - Not OpenAPI/Swagger
- ◯ WSDL
- ◯ GraphQL
- ◯ gRPC
- ◯ AsyncAPI
- ◯ RAML
- ◯ No se usa un estandar

¿En qué punto del proyecto suele considerar el diseño de la API? *

- ◯ Se considera primero, antes de las expectativas de los stakeholders
- ◯ Se considera temprano, antes de que comience el desarrollo
- ◯ Se considera en medio de un proyecto
- ◯ Se considera al final
- ◯ No preocupa el diseño de API

¿Qué factores considera antes de integrar una API? *

- [ ] Escalabilidad
- [ ] Confiabilidad
- [ ] Rendimiento
- [ ] Salud/tiempo de acitividad
- [ ] Documentación
- [ ] Cambios/versiones
- [ ] Seguridad
- [ ] Usabilidad
- [ ] Otro: _____

¿Considera que el uso de una API puede ayudar a los sistemas de TI para interactuar con el sistema PTracking? *

- ( ) Sí
- ( ) No
- ( ) Tal vez