

Buenas Prácticas en el Desarrollo de API's

Best Practices on API's Development

Roberto Iván Zúñiga Moncivais^{1*}, José Gabriel Navarro Favela¹, Jaime Iván López Veyna¹

^{1*} Tecnológico Nacional de México/Instituto Tecnológico de Zacatecas

*m20450779@itz.edu.mx

Resumen

En este documento se presentan distintos proyectos de desarrollo web que integran API's, de los cuales se identifican las técnicas y herramientas que se utilizaron para el desarrollo de API's, esta tecnología ha tenido un amplio crecimiento tanto para intercomunicar sistemas de información como para ser el enlace entre aplicación y datos (Microservicios). En base a estos proyectos y a la identificación de técnicas empleadas en dichos proyectos se proveerá una lista de buenas prácticas de complejidad baja para el desarrollo de API's con la finalidad de que desarrolladores con poca experiencia puedan crear API's seguras, eficientes y fáciles de usar.

Palabras clave: API, Buenas prácticas, Desarrollo

Abstract

This document presents different projects on web development those integrate APIs, of which techniques and tools that were applied for development are identified, this technology has had a strong growth both to intercommunicate information systems and to be a link between application and data (Microservices). Based on these projects and identification of the techniques applied on the projects will prove a best practices list with low complexity for API Development with the objective that inexperienced developers could to create API secure, efficient and easy-use.

Keywords: API, Best Practices, Web Development, Microservices

1. Introducción

Una API (Application Programming Interface - Interfaz de Programación de Aplicaciones) son un conjunto de funciones que se usa para diseñar e integrar el software de las aplicaciones, ya sean Web, Móvil o de escritorio (RedHat, 2017).

Las API's son herramientas muy importantes por que permiten conectar y compartir datos y de esta forma facilitar servicios y/o ampliarlos para que los usuarios puedan acceder a ellos sin necesidad de ir a cierta ubicación o en cierto horario. A continuación, se muestran algunos trabajos a manera de antecedente en donde se utilizan aplicaciones distribuidas con el fin de demostrar la importancia del uso de API's.

Luis Antonio Gama Moreno, José Ángel Torres Rangel y Carlos Martínez Hernández (Gama Moreno, Torres Rangel, & Martínez Hernández, 2014) señalan la importancia del uso de API's ya que estas reducen los tiempos y costos al realizar trámites, esto se demostró a través de la realización de una aplicación móvil para expedir documentos académicos electrónicos de la universidad, en la cual el alumno ingresa sus datos personales, selecciona el documento que necesita, realiza el pago por medio de la herramienta PayPal y al confirmarse el pago se le descarga el documento solicitado por el alumno. En conclusión, el alumno por medio de la aplicación puede hacer la solicitud de un documento a cualquier hora en cualquier día de la semana sin la necesidad de una oficina abierta para realizar el trámite.

Pedro Gomesa, Mário Antunesa (Gomesa & Antunesa, 2012) comentan la facilidad que brinda el uso de tecnologías de la información,

ya que estas otorgan comodidad a la hora de consultar información que antes solamente podía consultarse desde una oficina y/o casa. Gomesa y Antunesa implementaron una solución tecnológica para constatar dicha facilidad, fue por medio de una aplicación móvil para una institución de salud, en la que la aplicación consume un Web Service (Servicio Web) el cual al recibir los datos de un paciente devolvía la información clínica del paciente solicitado y esta información era desplegada en el dispositivo móvil para que el doctor la visualizara y poder dar una consulta correcta a dicho paciente.

El trabajo anterior utiliza un Web Service, el cual hace referencia a un sistema de comunicación, protocolos y estándares entre dos dispositivos electrónicos conectados a la misma red, para intercambiar datos entre sistemas o aplicaciones. Dicho de otro modo, es un software creado para la comunicación entre maquina y maquina y la interoperabilidad entre ellas (Tipos De Web Service En La Industria 4.0, 2021).

Uno de los elementos claves de los Web Services es el uso de XML (Extensible Markup Language o Lenguaje de Marcado Extensible en español). Es la manera en que están codificados o programados los datos para que se puedan procesar por diferentes sistemas, es decir, es un lenguaje estándar que permite que se dé la comunicación aun cuando las aplicaciones o sistemas usen lenguajes diferentes de programación.

Comprendiendo que es un Web Service, Una API y un Web Service se diferencian debido a que los Web Service usan SOAP (Simple Object Access Protocol o Protocolo Simple de Acceso a Objetos) como protocolo, en el que se deben seguir las reglas para poder usarlo, mientras que las API's usan REST (Representational State Transfer o Transferencia de Estado Representacional) como un estilo de arquitectura, el cual es un conjunto de principios arquitectónicos que se pueden implementar según se requiera, esta diferencia de que SOAP es un protocolo y REST un estilo de arquitectura, permite que las API's sean flexibles y fácil de implementar debido a que no tienen que seguir requisitos específicos como lo es el Web Service con SOAP.

En base a los trabajos previos de Luis Antonio Gama Moreno y de Pedro Gomesa se puede notar la relevancia que tiene el uso de API's para los usuarios finales, pero también para los desarrolladores, debido a que permiten generar interfaces de comunicación para diferentes plataformas y tecnologías, ya sea para celulares, computadoras, wearables, entre otros.

Un problema que surge con las API's es que no cuentan con un estándar de diseño común y tienen poca o nula documentación, lo que provoca que pierdan accesibilidad al no poder entender su funcionamiento.

1. Introducción

Por lo tanto, ha surgido la necesidad de estructurar y/o normalizar el desarrollo de API's esto con la finalidad de que no haya funcionalidad duplicada y su uso sea fácil y seguro, además de evitar errores de comunicación entre el usuario y la API.

Por lo tanto, el objetivo general es proporcionar una lista de buenas practicas para desarrollar API's de buena calidad, para llegar a este objetivo se debe identificar buenas practicas que se emplean en el desarrollo, para después evaluar la complejidad de las mismas y por último seleccionar las de menor complejidad pero que aportan mucho valor al desarrollo.

Por tal motivo, a continuación, se presentan algunos proyectos que siguieron algunas pautas en su desarrollo.

Mrs. Ong-Ang Ai Boon, Sopnendu Mohanty (Ai Boon & Mohanty, 2013) en su artículo relata que las API's deben estar estandarizadas para que los consumidores de estas puedan familiarizarse pronto con la API y hacer uso de ella de manera rápida y sencilla, para lograr la estandarización se siguen ciertas guías de diseño como: deben ser simple la especificación de los formatos de datos, métodos y autenticación, la nomenclatura de los métodos debe ser corta y a la vez explícita, versionar la API ayudará a que si surge alguna modificación los consumidores tengan tiempo de ajustarse a las nuevas funcionalidades.

Annuikka Varteva (Varteva, 2017) comenta las maneras en que se puede partir para construir una API, así también los beneficios y retos de estas maneras, además expresa los elementos que integran la especificación de API's como son el formato de datos y el modelo de datos y el acceso a la misma, esta información es de alto valor a la hora de hacer uso de la API ya que se evitan errores de comunes de tipos de datos y mal uso de métodos.

Sardar Mudassar Ali Khan (Khan, 2021) menciona la importancia de especificar la API's, precisamente con la herramienta Swagger que permite la especificación mediante YAML, Swagger permite que tanto el sistema como el desarrollador comprendan el funcionamiento y las capacidades

del sistema sin acceso directo al código fuente del proyecto. Además, Swagger minimiza la carga de trabajo para conectarse con el microservicio y reducir el tiempo necesario para documentar con precisión el microservicio.

Jyothi Salibindla (Salibindla, 2018) habla acerca de la relevancia de la seguridad en una API, ya que sin ella, se puede tener filtraciones a los datos o en un caso extremo pérdida de información, para contrarrestar esto menciona varias técnicas de autenticación y autorización en una API, como es la autenticación básica con usuario y contraseña, OAuth 2.0 utilizando algún servicio de un proveedor, uso de JWT (JSON Web Token) para la autorización de uso, y el uso de Whitelist que es un conjunto de IP's o dominios que pueden hacer uso de la API.

Sanjay Gadge y Vijaya Kotwani (Gadge & Kotwani, 2017) relatan la importancia de usar un API Gateway en una arquitectura de microservicios debido a que este funciona como el patrón de diseño de fachada dejando así una ruta pública y todas las API's de manera privada, además de que solo existe un punto acceso, la seguridad solamente se aplica a este y automáticamente se aplica a todas las API's que estén comunicadas con él, por lo que si se aplica algún método de autenticación o autorización únicamente será necesario aplicarlo en el API Gateway, además de que el API Gateway provee herramientas de monitoreo y tráfico de las API's.

Carlos Rodríguez, Marcos Baez, Florian Daniel, Fabio Casati, Juan Carlos Trabucco, Luigi Canali, y Gianraffaele Percannella (Carlos, et al., 2016), en este artículo se menciona las principales características que deben seguir las API's a la hora de crear los endpoints o URL's, cuyas características son modelado, identificación, representación, operaciones y enlaces que al aplicar dichas características se tendrá una API estandarizada además también se habla de la importancia de los factores que existen a la hora de hacer uso de la API como lo es el navegador, tipos de medio, los verbos empleados a la hora de realizar las peticiones a la API.

2. Antecedentes

La metodología de la presente investigación es comparativa del tipo cualitativa dado que se centra en la recopilación y el análisis de resultados en trabajos previos con el fin de identificar técnicas y/o herramientas de desarrollo de API's en base a características particulares como lo son eficiencia, facilidad de uso e impacto. Mediante la recolección de algunos documentos y/o artículos relacionados al desarrollo de API's y posteriormente analizados para la identificación de las técnicas utilizadas, se elabora una tabla comparativa acerca de las pautas y/o procesos que se emplearon en la realización de estos proyectos y de esta forma identificar algunas buenas prácticas en el desarrollo de API's.

Una vez identificadas estas buenas prácticas también se realiza una evaluación para determinar la facilidad de implementación en el desarrollo de API's, esta evaluación consiste en poner en práctica cada técnica y medir su complejidad de implementación, todo esto con la finalidad de proveer una lista de buenas prácticas de baja complejidad que permitan desarrollar API's fáciles de usar, eficientes y seguras.

3. Resultados

Del análisis de los documentos se identifican varias técnicas y/o herramientas que ayudan en el desarrollo de API's, dichas técnicas tienen ciertas características las cuales se describen a continuación:

Documentación: Consiste en redactar que hace la API y cada uno de sus

- elementos, en esta redacción se incluye información como la descripción y ¿cuáles son los criterios de uso tanto de entrada como de salida?, entiéndase como criterios todo aquello referente a tipo de datos, métodos de acceso, autenticación, entre otros.

La importancia de la documentación es que permite que el usuario conozca y entienda las capacidades de la API sin necesidad de ver el código fuente o de inspeccionar su funcionamiento mediante el método de prueba y error.

- Estandarización: Es el proceso mediante el cual se busca que las API's sean semejantes y así tengan un comportamiento predictivo, esto se logra a través del uso de pautas que se deberían seguir para homologar la API con otras, algunas de estas pautas son:
- Uso correctamente de los métodos HTTP para realizar acciones específicas (POST para crear, GET para consultar, PUT o PATCH para modificar, DELETE para eliminar).
- Definir los puntos de acceso con sustantivos y no hacer uso de verbos como "ver – usuario", "eliminar – usuario" o "buscar – usuario", como se menciona en el punto anterior el método HTTP indica la acción a realizar por lo que no es necesario especificar la acción en el path o endpoint.
- A cada uno de los endpoints o paths de consulta que tengan como respuesta un listado de registros agregar las opciones de filtrado (filtering), paginación (paging) y ordenamiento (sorting), con la finalidad de evitar la definición de paths con estas opciones y tener varias funciones redundantes.
- Manejo correcto de los códigos de respuestas HTTP para cada petición, y en caso de error brindar un mensaje breve y claro del error.
- Manejo de versiones para que los nuevos cambios o actualizaciones y corrección de bugs o errores, puedan ser desplegados a producción y manteniendo al mismo tiempo la integridad del entorno de producción existente.
- API Gateway: Un API Gateway es el gestor de tráfico que interactúa con los datos o el servicio backend real y aplica políticas, autenticación y control de acceso general para las llamadas de una API para proteger datos valiosos. Un API Gateway fue diseñado para optimizar la comunicación entre los clientes externos y sus servicios de backend (TIBCO, 2022).

A grandes rasgos es un componente software el cual funciona como un enrutador para la o las API's, es decir, las aplicaciones realizan peticiones al API Gateway, y este redirecciona esa petición a la API que realiza la acción solicitada (Figura 1), al hacer uso de un API Gateway permite que las API's tengan una IP o dominio privado por lo que sean más seguras al ser aisladas, además el API Gateway contiene herramientas para el control de monitoreo y tráfico por lo que ayuda en la toma de decisiones a la hora de escalar las aplicaciones. Algunas herramientas son Tyk y Kong.

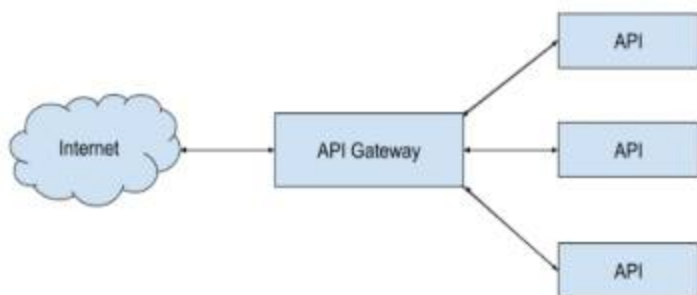


Figura 1: Funcionamiento del API Gateway.

- Autenticación Básica: Consiste en el uso de las credenciales de inicio de sesión que permiten a los usuarios iniciar sesión y verificar sus identidades en sus cuentas. Las credenciales de usuario suelen ser una combinación de nombre de usuario y contraseña que se utiliza para iniciar sesión en una cuenta (Fortinet, 2022).

Tanto el usuario y la contraseña tienen características que deben cumplir para ser útiles:

- Usuario: debe ser como una identificación del usuario por lo que debe ser único, es decir no deben existir dos cuentas con el mismo usuario.
- Contraseña: Es una combinación secreta de caracteres que puede incluir letras, números y símbolos, se recomienda una longitud mínima de 8 caracteres y que se use al menos un número, letra y símbolo para aumentar la seguridad de la contraseña y no sea fácil de descifrar.

- Autenticación JWT: Un JSON Web Token o JWT es un estándar abierto que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto JSON. Esta información se puede verificar y confiar porque está firmada digitalmente (JWT, 2022).
- Es una autenticación basada en un token el cual se forma con los datos del solicitante y contiene una fecha de expiración, este token se valida con cada petición que se realiza, revisando si aún es válido y vigente, en caso de que el token haya expirado se le solicita que genere uno nuevo y así sucesivamente.
- Autenticación OAuth2.0: OAuth es un protocolo que permite una autorización segura a través de un método simple y estándar para la web, móvil, y aplicaciones de escritorio (OAuth, 2022).

Este protocolo consiste en un inicio de sesión tradicional con usuario y contraseña, pero se realiza la validación de los datos a través de un sistema tercero (SSO - Single Sign On), es decir, la aplicación realiza una autenticación en un sistema tercero el cual nos regresa el resultado de la petición y dependiendo de la respuesta se permite acceder a la aplicación.

- Autorización por Whitelist/Blacklist: son dos metodologías para controlar el acceso a sitios web, correo electrónico, software y direcciones IP en las redes (National Cyber Security Society, 2018).

Whitelist consiste en negar el acceso a todos los recursos y solo los elementos en la lista pueden tener acceso.

Blacklist caso contrario de la *Whitelist* esta otorga el acceso a todos los recursos, excepto a los elementos de esta lista, a ellos se les niega el acceso.

Esta configuración se podrá realizar en los servidores web de cada aplicación o en los firewalls.

A continuación, se presenta la siguiente tabla que contiene una comparativa de las técnicas descritas anteriormente, las cuales se clasifican en dos grupos: en diseño y seguridad para la API (Tabla 1).

Tabla 1: Identificación y clasificación de técnicas.

Autor	Diseño			Seguridad			
	Documentación	Estandarización	Api Gateway (Arquitectura)	Basica	JWT	OAuth 2.0	WhiteList / BlackList
Mr. Qing-Ang Ai Boon, Sepenuh Voluntary (2015)	X	X					
Amalia Varley (2016)		X					
Sadia Midouat Ali Khan (2021)	X						
Jyothi Subhadra (2018)				X	X	X	X
Sangryul Gopal (2017)			X	X	X		
Carlos Rodriguez, Marcos Barr, Florian Dussel, Fabio Casati, Juan Carlos Trujillo, Lugo Casali, Gonzalo Pérez (2019)	X	X					

Además de la identificación de técnicas también se determina la complejidad del uso, esta complejidad se clasifica en 3 niveles (alta, media y baja) basándose en los recursos que requieren para su uso o implementación, que pueden ser tiempo de implementación, software o hardware adicional, dejando como complejidad baja las que menos recurso requieren y viceversa (Tabla 2).

Tabla 2: Complejidad de integración de técnicas.

Técnica	Complejidad		
	Baja	Media	Alta
Documentación	X		
Estandarización	X		
Api Gateway (Arquitectura)			X
Basica	X		
JWT	X		
OAuth 2.0		X	
WhiteList / BlackList		X	

Esta clasificación de complejidad se obtiene de la implementación de cada una de las técnicas en un proyecto que consiste en el desarrollo de una API para un gestor de documentos, a continuación, se detalla cómo fue la implementación:

- **Documentación:** Se realiza con Swagger que es una serie de reglas, especificaciones y herramientas que ayudan a documentar API's, se utiliza la versión OpenAPI 3.0 en la cual ofrece una estructura con 3 secciones obligatorias:
- **Openapi:** Se especifica la versión de openapi a utilizar.

- **Info:** Sección donde se detalla los datos generales del API como es el nombre o título, descripción, versión, entre otras.
- **Paths:** Se desglosa cada uno de los endpoints con el detalle de cada uno ellos, se especifica la ruta, el método, los parámetros que recibe, las respuestas que puede regresar, entre otros.

Las secciones anteriores son las requeridas por Swagger, existen otras donde se definen más características como lo son: los componentes, seguridad, documentación externa, entre otras.

La documentación en Swagger se realiza de manera fácil y sencilla, debido a que utiliza una estructura sencilla, ya sea en formato YAML o JSON para su elaboración (Figura 2), esta estructura es bastante legible y fácil de comprender, en ella rápidamente se puede identificar todos los elementos que tiene la API como lo son los path o endpoints, los métodos, la descripción, datos del request o petición así también ver las posibles respuestas que pudiera arrojar la petición. Además, con la herramienta Swagger UI se puede visualizar la documentación de manera grafica lo que facilita más la lectura de la documentación.

```

openapi: 3.0.1
info:
  title: API de Documentos
  version: 1.0.0
paths:
  /documentos:
    get:
      tags:
        - documentos
      summary: Listado de documentos
      description: Listado de documentos registrados.
      operationId: listarDocumentos
      responses:
        200:
          description: Operación Exitosa
          content:
            application/json:
              schema:
                type: object
                properties:
                  data:
                    description: Listado de Documentos.
                    type: array
                    items:
                      $ref: '#/components/schemas/Documento'
      security:
        - bearerAuth: []
  /documentos/crear:
    post:
      tags:
        - documentos
      summary: Crear un nuevo documento
      description: Crear un nuevo documento con los datos recibidos.
      operationId: crearDocumento
      requestBody:
        description: Datos del documento a crear
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Documento'
      responses:
        201:
          description: Operación Exitosa
          content: {}
      security:
        - bearerAuth: []
  /documentos/eliminar:
    delete:
      tags:
        - documentos
      summary: Eliminar un documento
      description: Eliminar un documento por su ID.
      operationId: eliminarDocumento
      requestBody:
        description: ID del documento a eliminar
        content:
          application/json:
            schema:
              type: object
              properties:
                id:
                  type: string
      responses:
        204:
          description: Operación Exitosa
          content: {}
      security:
        - bearerAuth: []
components:
  schemas:
    Documento:
      type: object
      properties:
        id:
          type: string
        titulo:
          type: string
        contenido:
          type: string
        fechaCreacion:
          type: string
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

```

Figura 2: Estructura de documento Swagger.

También Swagger en su página oficial provee un ejemplo completo de una API, el cual se puede usar como una plantilla, además no requiere la instalación de algún programa para su uso.

- Estandarización: Se revisa que los endpoints (URL's) se encuentren correctamente formados, es decir, que utilicen el verbo HTTP correcto, uso de sustantivos, todas sus respuestas definidas, así también la jerarquía que existe entre los endpoints, esta actividad se realiza fácilmente debido a que se toma como base el documento de Swagger (Figura 3), en dado caso que no se cuente con dicho archivo la mayoría de los frameworks tienen comandos con los cuales se puede visualizar los endpoints o las rutas disponibles que tiene la API.



Figura 3: Representación de endpoints con Swagger.

En el caso de la Figura 3, se visualiza el correcto uso sustantivos, de los verbos tanto para consultar, agregar, modificar y eliminar información, pero existe un error en el tercer endpoint "/documentos/search-dependencia/{id}", debido a que usa verbos en su definición y además no respeta la jerarquía de las entidades, es decir la entidad padre desglosa las entidades hijas, en este caso una dependencia tiene documentos por lo que la correcta redacción del endpoint sería como el último de la imagen "/dependencias/{id}/documentos" interpretándose que de la dependencia se van obtener los documentos.

Además, se agrega la opción de filtrar, ordenar y paginar a los paths de consulta (GET) que regresan un conjunto de registros, evitando la necesidad de generar un path para dichas acciones, y de esta forma evitar la funcionalidad redundante.



Figura 4: Especificación de filtros, ordenamiento y paginación.

También se especifica la versión de la API, dicha información se detalla en dos secciones del documento de Swagger: en la sección de info y en la de servers (Figura 5).

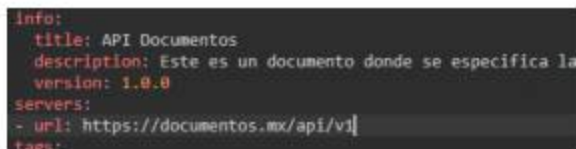


Figura 5: Especificación de versión.

- En info: se indica el número o nombre de versión, cabe señalar que este dato es requerido.
- En servers: se especifica la URL haciendo referencia a la nueva versión de la API.
- API Gateway: Se utiliza el API Gateway Kong el cual se instala en un servidor aparte a la API, una vez instalado se configura para tener acceso a él y además se configura cada uno de los endpoints de la API para que el API Gateway pueda redireccionar correctamente, API Gateway ofrece la herramienta de monitor de tráfico para visualizar y tomar decisiones si hay alguna API o endpoint con sobrecarga, además permite establecer la capa de seguridad, por lo que no es tan necesario aplicar dicha capa a las API's.
- El API Gateway se vuelve complejo de implementar debido a que requiere un servidor (hardware) y una herramienta (software) como lo es Kong para su uso, lo que se traduce en costos tanto en hardware como en capacitación.
- Autenticación Básica: Sencilla de implementar es la autenticación tradicional con el uso de un usuario y una contraseña, la mayoría de los frameworks ofrecen módulos o componentes que permiten integrar un módulo de autenticación básica, donde solamente se configura la conexión a la base de datos de los usuarios que pueden acceder a la aplicación, por lo tanto la implementación es rápida y sencilla, debido a que solamente se requiere leer un poco de documentación del framework que se esté usando para implementar este tipo de autenticación.

- Autenticación JWT: El uso de token al igual que la autenticación básica es simple de implementar, ya que algunos frameworks ofrecen esta opción por default o si no la ofrecen se puede hacer uso de algún plugin, complemento, extensión que permite generar y validar, por ejemplo JWT.io que se encuentra disponible para distintos lenguajes de programación.
- Autenticación OAuth2.0: Su integración es sencilla debido a que solamente se requiere registrar la aplicación con el proveedor de servicio y después se configura en la API este proveedor de servicio con los datos que se obtienen del registro realizado, pero se clasifica en término medio debido a que depende de con cuales servicios se desea iniciar sesión, entre más servicios más configuración, tanto del lado de la API como del prestador del servicio, su mayor inconveniente es que requiere de acceso a internet por lo que si la API se encuentra montada o instalada en un servidor sin conexión a Internet, esta opción no es viable por el hecho de que no podrá conectarse al servicio del tercero y no podrá realizar la autenticación.
- Uso de Whitelist/Blacklist: Su implementación es simple debido a que solamente se requiere tener conocimientos del servidor web que se este usando ya sea un Apache o un NGINX, en el cual se configura la lista ya sea una Whitelist o Blacklist en la cual se desglosa los elementos a bloquear o a permitir el acceso, esto según el tipo de lista que se este configurando, se considera complejo debido a que es una técnica en la que se necesita tener conocimiento de la configuración de los servidores Web y además se tiene que ir haciendo adecuaciones continuamente, es decir, se tiene que ir actualizando las listas para permitir o denegar acceso a nuevos dominios o IP's según se vaya necesitando.

4. Discusión

Partiendo de los resultados encontrados se entiende que en el desarrollo de las API's se puede hacer uso de pautas o buenas prácticas que permiten mejorar el desarrollo y la operatividad de las API's con la finalidad de que el desarrollo sea de calidad y tenga la mínima cantidad de defectos.

Las buenas prácticas pueden ser aplicadas a los proyectos de desarrollo ya sea de manera individual o en conjunto, la gran mayoría de ellas pueden coexistir es decir una no perjudica a la otra, a excepción de que esta buena práctica este enfocada a misma área a mejorar, entonces si se podría generar conflicto una con otra, aunque no se encontró evidencia sustancial de dicho suceso, porque en base a la autenticación y autorización de usuarios se encontraron varias buenas prácticas, pero ninguna de ellas excluye una de otra.

5. Conclusiones

En base al análisis de los documentos encontrados el hacer uso o aplicar estas buenas prácticas brinda diferentes beneficios como minimizar errores, facilitar el uso, mínima redundancia de funciones y seguridad, estos beneficios son alcanzables debido a que las buenas prácticas hacen que las API's tengan un comportamiento predecible de esta manera el usuario y el desarrollador puede familiarizarse rápidamente.

Las buenas prácticas no siempre se pueden aplicar, va a depender en gran medida del proyecto tanto de la complejidad del mismo como de los recursos que se dispongan para el mismo, ya que algunas pueden requerir de uso de hardware específico, tiempo de implementación o de alguna capacitación en alguna tecnología, por lo que si no contamos con estos recursos difícilmente se podrán implementar.

Dicho lo anterior se encontraron algunas buenas prácticas que requieren de pocos recursos y son de baja complejidad para su implementación lo que las vuelve idóneas para desarrolladores principiantes, estas buenas prácticas son:

- Documentación.
- Estandarización.
- Autenticación básica.
- Autenticación JWT

Debido a que la complejidad de estas tecnologías es baja son de rápida implementación, lo cual no representan un cambio drástico en el diseño, ni en el desarrollo de la API, por lo que siguiendo estas buenas prácticas se podrán generar API's de un nivel aceptable, debido a que será estandarizada, no

Debido a que la complejidad de estas tecnologías es baja son de rápida implementación, lo cual no representan un cambio drástico en el diseño, ni en el desarrollo de la API, por lo que siguiendo estas buenas prácticas se podrán generar API's de un nivel aceptable, debido a que será estandarizada, no tendrá endpoints redundantes y tendrá una capa de seguridad.

Además de que contara con su respectiva documentación para que los usuarios puedan conocer y entender el funcionamiento de la API sin necesidad de inspeccionar el código fuente o bien de hacer el ensayo y error.

6. Conclusiones

El presente documento contempla algunas técnicas empleadas en dos rubros que son diseño y seguridad en el desarrollo de API's, de los cuales no son todas las técnicas que se puedan aplicar, pueden surgir mejoras o nuevas técnicas que puedan aplicar en estos mismos rubros o en otros como es la codificación, pruebas, despliegue, entre otras.

Por lo que el trabajo a futuro sería complementar el trabajo con otras técnicas y a partir de esto, establecer un conjunto de buenas prácticas generales que puedan ser aplicados a cualquier API y cubriendo todas o la mayoría de las fases del desarrollo.

7. Agradecimientos

Agradezco mucho el apoyo de los docentes del Instituto Tecnológico de Zacatecas por brindarme sus puntos de vista y correcciones para enriquecer y mejorar este trabajo.

A mis familiares, amigos, compañeros de clase y de trabajo que al pedirle su ayuda siempre trataron de apoyarme pese a no ser su área de especialidad.

Referencias bibliográficas

Ai Boon, M. O.-A., & Mohanty, S. (2013). Finance-as-a-Service: API Playbook. Obtenido de ABS-MAS: <https://abs.org.sg/docs/library/abs-api-playbook.pdf>

Carlos, R., Baez, M., Florian, D., Casati, F., Trabucco, J. C., Canali, L., & Percannella, G. (2016). REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. Recuperado el 2021, de Springer: https://link.springer.com/content/pdf/10.1007%2F978-3-319-38791-8_2.pdf

Fortinet. (2022). Fortinet. Obtenido de <https://www.fortinet.com/resources/cyberglossary/login-credentials>

Gadge, S., & Kotwani, V. (2017). Microservice Architecture: API Gateway Considerations. Obtenido de Globallogic: [http://mainlab.cs.ccu.edu.tw/presentation/pdf/\(2017\)Microservice-Architecture-API-Gateway-Considerations.pdf](http://mainlab.cs.ccu.edu.tw/presentation/pdf/(2017)Microservice-Architecture-API-Gateway-Considerations.pdf)

Gama Moreno, L. A., Torres Rangel, J. A., & Martínez Hernández, C. (29 de 10 de 2014). Plataforma de aplicaciones para documentos electrónicos a través de cómputo móvil. Instituto tecnológico de Tlajomulco, Jal. Obtenido de <https://revistas.uis.edu.co/index.php/revistagti/articloe/download/4870/4995>

Gomesa, P., & Antunesa, M. (2012). Mobile edoclink: A Mobile Workflow and Document Management Application for Healthcare Institutions. Obtenido de ScienceDirect: <https://www.sciencedirect.com/science/article/pii/S2212017312005348>

JWT. (2022). JWT. Obtenido de <https://jwt.io>

Khan, S. M. (02 de 2021). How to Implement Swagger in Asp.net Core Web Api. Obtenido de ResearchGate: https://www.researchgate.net/publication/349304580_How_to_Implement_Swagger_in_Aspnet_Core_Web_Api

Referencias bibliográficas

National Cyber Security Society. (03 de 2018). National Cyber Security Society. Obtenido de https://nationalcybersecuritysociety.org/wp-content/uploads/2018/03/FACT-Whitelist_Blacklist-FINAL.pdf

OAuth. (2022). OAuth. Obtenido de <https://oauth.net/>

RedHat. (31 de 10 de 2017). Qué es una API. Obtenido de RedHat: <https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces>

Salibindla, J. (2018). IJERT-Microservices api security. (Academia.edu) Obtenido de Academia: https://www.academia.edu/39254064/IJERT_Microservices_api_security

TIBCO. (2022). TIBCO. Obtenido de <https://www.tibco.com/es/reference-center/what-is-an-api-gateway>

Tipos De Web Service En La Industria 4.0. (2 de 02 de 2021). Obtenido de ATVISE: <https://atvise.vesterbusiness.com/news/tipos-web-service-en-la-industria/>

Varteva, A. (2017). API toolkit. Obtenido de 6Aika: https://6aika.fi/wp-content/uploads/2017/11/20171102_APItoolkit_WEB.pdf