EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Adaptación de un patrón de software en seguridad a la arquitectura de un Microservicio

presentada por

Ing. Karen Monica Hernández Guzmán

como requisito para la obtención del grado de

Maestría en Ciencias la Computación

Director de tesis

Dr. Juan Carlos Rojas Pérez

Cuernavaca, Morelos, México. Octubre de 2023.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

SEP TecNM CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
RECIBIDO
03 OCT 2023
SUBDIRECCIÓN ACADÉMICA

Cuernavaca, Mor., 21/Septiembre/2023

OFICIO No. DCC/177/2023

Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14-OFICIO

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial de KAREN MÓNICA HERNÁNDEZ GUZMÁN con número de control M21CE060, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado **"ADAPTACIÓN DE UN PATRÓN DE SOFTWARE EN SEGURIDAD A LA ARQUITECTURA DE UN MICROSERVICIO"** y hemos encontrado que se han atendido todas las observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

JUAN CARLOS ROJAS PÉREZ
Director de tesis

RENÉ SANTAOLAYA SALGADO
Revisor 1

OLIVIA GRACIELA FRAGOSO DÍAZ
Revisor 2

EDUCACIÓN
TECNOLÓGICO NACIONAL DE MÉXICO
06 OCT 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO
EBV

C.c.p. Depto. Servicios Escolares.
Expediente / Estudiante

Cuernavaca, Mor., 03/octubre/2023
No. De Oficio: SAC/158/2023
Asunto: Autorización de impresión de tesis

KAREN MÓNICA HERNÁNDEZ GUZMÁN
CANDIDATA AL GRADO DE MAESTRA EN CIENCIAS DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"ADAPTACIÓN DE UN PATRÓN DE SOFTWARE EN SEGURIDAD A LA ARQUITECTURA DE UN MICROSERVICIO"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Conocimiento y tecnología al servicio de México"

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SUBDIRECCIÓN ACADÉMICA

CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

C. c. p. Departamento de Ciencias Computacionales
Departamento de Servicios Escolares

CMAZ/lmz

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
04 OCT 2023
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS ESCOLARES
RECIBIDO
EBV

Agradezco a Dios por brindarme la oportunidad de cumplir esta meta, por bendecirme con el amor y compañía de mis seres queridos, quienes me han dado fuerzas y ánimo en todo momento. Su apoyo incondicional ha sido mi mayor motivación para alcanzar este logro que también es suyo.

Agradecimientos

"En medio de todas nuestras metas y ambiciones, nunca olvidemos agradecer a quienes nos han tendido la mano en el camino." - John F. Kennedy Por ello quiero agradecer:

Al Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) por brindarme la oportunidad y el apoyo para realizar mis estudios de maestría. Al Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), por las vivencias, y la grata coincidencia con personas maravillosas.

A mis queridos padres María de los Ángeles y Natividad, mi familia Brisa, Ángel, Angélica, Sari, Minette y Vilchis, quienes siempre confiaron en mí y fueron mi principal motivación, gracias por la inspiración.

A mi amado abuelito Serafín y mi adorado tío Alfonso, que, aunque ya no están sé que estarían muy contentos de saber que lo he logrado.

A mi director de tesis, él Dr. Juan Carlos Rojas Pérez y al comité revisor: él Dr. René Santaolaya Salgado y la Dra. Olivia Fragoso Díaz, por contribuir en investigación con su experiencia, retroalimentación, revisión, guía, por sus comentarios y sugerencias fueron invaluables para mejorar la calidad de esta tesis de esta investigación.

A la Dra. Azucena Montes, al Dr. Carlos Astorga, a la Dra. Andrea Magadan, a la Lic. Verónica Sotelo, a la Dra. Gloria Osorio, el Dr. Rodolfo Vargas, a la Lic. Karina Zaldívar, a la Lic. Lorena Ruiz, al Lic. Víctor y al Ing. Christopher Garduño, por él todo el apoyo que me brindaron durante mi estancia en CENIDET.

A mis queridos compañeros de aventura, Keyla, Noemí, Eidy, Itzel, Karina, Cynthia, Santander, Trujillo, Moncada, Eliezer, Uziel, Miguel, Valentín, Fily, Hermain, Ariel, Darién, Jhon, Berny, Jordán y Fausto, quienes estuvieron conmigo en todo tipo de experiencias desde las más dolosas hasta las más fantásticas, gracias por enseñarme que las mejores amistades son espontáneas, el tiempo compartido puede ser fugaz, pero que el cariño será para siempre y que si tienes la más mínima oportunidad de ser feliz y disfrutar, tómala y gózalo a más no poder, porque cosas malas siempre va a haber, pero las buenas deberás crearlas.

No puedo dejar de mencionar a mis conocidos de las diferentes áreas quienes enriquecieron mis conocimientos y me inspiraron a seguir adelante en mi camino académico, ya que, de alguna u otra manera, contribuyeron a esta tesis y a mi formación de vida. Sin el respaldo de todos ustedes, este trabajo no habría sido posible. Estoy profundamente agradecida por todo el apoyo recibido a lo largo de esta investigación.

Solo me resta decir: “cumplan sus metas y solo así alcanzarán sus sueños,” – KMHG

Resumen

Este trabajo investiga los riesgos asociados con la adopción de la arquitectura de microservicios, centrándose en la seguridad. A medida que esta arquitectura gana popularidad debido a su capacidad de mejorar la modularidad y escalabilidad en sistemas de software, se examina su preferencia sobre las arquitecturas monolíticas convencionales.

Los microservicios ofrecen ventajas como modularidad, escalabilidad independiente, entrega continua y flexibilidad tecnológica, lo que impulsa su adopción en aplicaciones modernas. Sin embargo, presentan desafíos significativos en términos de seguridad. La naturaleza descentralizada de los microservicios puede aumentar la superficie de ataque y complicar la seguridad. Esta investigación aborda estos desafíos, resaltando la importancia de prevenir accesos no autorizados.

Es esencial entender que, para aprovechar las ventajas de los microservicios, se requiere un enfoque holístico en seguridad. Una gestión adecuada de la seguridad no solo protege sistemas, sino que también establece bases sólidas para la innovación y el crecimiento en el desarrollo de software.

La importancia de esta investigación radica en la necesidad de diseñar patrones de seguridad específicos para microservicios. Los ataques en sistemas basados en microservicios, incluso en grandes empresas como Netflix, resaltan la necesidad de abordar la seguridad en estas arquitecturas dinámicas.

El objetivo principal fue adaptar y proponer soluciones efectivas para fortalecer la seguridad en sistemas basados en microservicios. Se ha desarrollado un patrón de seguridad llamado "Microservice Security Pattern API Gateway" (MSPAG), implementado en lenguajes como C#, Python y Java. MSPAG utiliza JSON Web Tokens (JWT) y criptografía H256 para proteger los puntos de acceso de los microservicios. Este aporte busca aumentar la confianza en la adopción de esta arquitectura, promoviendo sistemas más seguros en el desarrollo de software actual.

Abstract

This work explores potential risks associated with the microservices architectural approach, with a focus on security during its adoption. The growing popularity of this architecture is attributed to its ability to enhance modularity, independence, and scalability in software systems. In this research, the preference for this approach is analyzed in comparison to conventional monolithic architectures.

Microservices architectures offer advantages such as modularity, independent scalability, continuous delivery, and technological flexibility, which drive their adoption in modern application development. However, they also present considerable challenges in terms of security. The fragmented and decentralized nature of microservices can increase the attack surface and complicate the security landscape. This research addresses these challenges, emphasizing the importance of safeguarding against unauthorized access.

It's crucial to understand that, in order to fully leverage the undeniable benefits of microservices architectures, a holistic security approach is essential. Proper security management not only protects systems but also establishes a strong and reliable foundation for innovation and continuous growth in software development.

The driving force behind this thesis is the pressing need to design specific security patterns for microservices. The observation of massive attacks on microservices-based systems, even by major companies like Netflix, underscores the importance of appropriately addressing security aspects in these dynamic and challenging architectures.

Through a comprehensive analysis of existing security patterns, the main objective is to adapt and propose effective solutions to strengthen the security of microservices-based systems. As a result of this thesis, a specific security pattern for microservices has been developed, named "Microservice Security Pattern API Gateway" (MSPAG). This pattern has been implemented in various programming languages such as C#, Python, and Java. MSPAG employs an adaptation of the API Gateway using JSON Web Tokens (JWT) and the cryptographic algorithm H256 to secure the access points or endpoints of microservices. This contribution aims to instill greater confidence in the adoption of this architectural approach, paving the way for building more secure and reliable systems in the current landscape of software development.

Keywords: Microservices, architecture, security, security patterns, data protection, unauthorized access, MSPAG, API Gateway, JWT, H256, AMS, MVC y Facade.

Contenido

Índice de Figuras

Índice de Tablas

1.Introducción

En la actualidad, el desarrollo de sistemas que ofrezcan modularidad, independencia y escalabilidad está impulsando la adopción de nuevos paradigmas que engloban estas características. La arquitectura de microservicios es un estilo arquitectónico que muchas organizaciones están adoptando para reemplazar las arquitecturas monolíticas. Según [1], los microservicios son un estilo arquitectónico en tendencia que busca diseñar sistemas complejos como colecciones de artefactos de software de grano fino y baja interconexión; cada microservicio implementa una pequeña parte o incluso una única función de la lógica empresarial de las aplicaciones.

Aunque este estilo ofrece muchas ventajas para el desarrollo de sistemas complejos, también presenta desafíos, especialmente en términos de seguridad. Según Fernández [2], la seguridad implica protección contra la divulgación no autorizada de datos, la modificación no autorizada de datos, la denegación de servicio y la falta de responsabilidad. Fernández también describe contramedidas para garantizar la seguridad, como la identificación y autenticación, la autorización y el control de acceso, así como el registro y la auditoría.

El trabajo de Pereira [3], destaca que informes industriales revelan que empresas como Netflix han sufrido ataques masivos en sus sistemas basados en microservicios en los últimos años. Estos ataques abarcan desde servicios específicos hasta toda la arquitectura. Sin embargo, para abordar este problema de seguridad, existen varios mecanismos y patrones que se aplican en arquitecturas monolíticas, como se explica en Fernández [2]. En cuanto a las arquitecturas de microservicios, varios trabajos de mapeo sistemático reportan mecanismos, pautas, detección de amenazas y formas de mitigar o prevenir problemas de seguridad [1], [3]. A pesar de esto, pocos se han centrado en la creación de patrones de seguridad específicos para arquitecturas de microservicios, como se presenta en los trabajos de Barabanov, Xiuyu y Bánáti [4]–[6].

Este trabajo de tesis, se enfoca en el estudio de los patrones de seguridad utilizados en diferentes arquitecturas de software y en la adaptación de un patrón de seguridad a la arquitectura de microservicios.

1.1.1 Antecedentes.

En el Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), se están desarrollando diversas investigaciones orientadas a tecnologías emergentes, como son los microservicios, las investigaciones en curso comprenden las distintas aristas de este tópico como son: “Selección y Adaptación de Métricas para Microservicios” y “Estudio de Mecanismos, Métricas y Patrones de Seguridad en Microservicios”.

1.1.2 Selección y Adaptación de Métricas para Microservicios.

En Joya [7] , se presenta un estudio sobre distintas métricas en el que se busca adaptar una métrica orientada a objetos hacia una arquitectura de microservicios.

1.1.3 Estudio de Mecanismos, Métricas y Patrones de Seguridad en Microservicios.

En el trabajo de Penagos [8], se presenta un estudio de mapeo sistemático de métricas y patrones de seguridad en microservicios, con el propósito de determinar cuáles de estos patrones son aplicables en términos de seguridad. Además, se elabora una guía que proporciona elementos útiles para su implementación en microservicios.

1.2 Descripción del problema

De acuerdo con Fernández [2], un patrón de seguridad describe una solución al problema de controlar (detener o mitigar) un conjunto de amenazas específicas mediante algún mecanismo de seguridad definido en un contexto dado. La ausencia de un patrón de desarrollo de software específico para una solución recurrente conlleva a la duplicación de código.

Existen patrones de software en seguridad, específicos para el desarrollo de microservicios. Al aplicar un patrón de software en seguridad diseñado para otro tipo de arquitectura, se corre el riesgo de brindar una protección inadecuada o insuficiente a la seguridad de los microservicios, dejándolos vulnerables.

La arquitectura de un Microservicio difiere al de un servicio web o a la funcionalidad de un componente de software de una aplicación monolítica [3]. Contar con patrones para resolver problemas de seguridad específicos en Microservicios evita la codificación recurrente, así como propone una correcta protección.

1.3 Objetivos

1.3.1. General

Adaptar un patrón de software en un problema de seguridad específico aplicado en otras arquitecturas de software (monolíticas, orientadas a objetos u orientadas a servicios), hacia una arquitectura de microservicios.

1.3.2. Específicos

a) Disminuir la vulnerabilidad en un problema o de seguridad en la arquitectura de microservicios.
b) Proveer de un patrón en seguridad específico para la arquitectura de microservicios.

1.4 Alcances y Limitaciones

1.4.1 Alcances

a) Implementar/Adaptar un patrón de seguridad en un problema específico de software para Microservicios.
b) Realizar pruebas preliminares del patrón implementado/adaptado.
c) Se utilizan repositorios de software de acceso libre encontrados en la web.

1.4.2 Limitaciones

a) La investigación se enfoca en las principales soluciones en patrones de diseño en arquitecturas de software (monolíticas, orientadas a objetos, orientadas a servicios), para atender la seguridad en desarrollo de software.
b) Se utilizan herramientas de uso libre para las pruebas preliminares.

1.4.3 Estructura del documento

La estructura del documento de esta tesis se organiza de la siguiente manera:

Capítulo 2: Marco Teórico

En este capítulo se presenta el marco teórico, donde se exploran los conceptos fundamentales de microservicios, arquitecturas de software y seguridad en productos de software.

Capítulo 3: Estado del Arte

El capítulo 3 aborda el estado del arte, enfocándose en temas relacionados con la seguridad en microservicios, así como otros aspectos vinculados a la seguridad de arquitecturas de software. Se discuten también las principales medidas de seguridad.

Capítulo 4: Implementación de la Solución

En este capítulo se detalla la implementación de la solución. Se describe el proceso de análisis, selección y adaptación del patrón de seguridad específico para arquitecturas de microservicios.

Capítulo 5: Pruebas y Resultados

El capítulo 5 presenta las pruebas y los resultados obtenidos del patrón de seguridad para microservicios en tres diferentes lenguajes de programación.

Capítulo 6: Conclusiones y Futuros Trabajos

En el capítulo 6, se exponen las conclusiones obtenidas a lo largo de la investigación. Además, se resaltan las contribuciones de esta tesis y se sugieren los posibles trabajos futuros que podrían extender la investigación.

CAPÍTULO

02

Marco Teórico

En este capítulo se abordan las definiciones y conceptos fundamentales de arquitecturas de microservicios, así como de seguridad en arquitecturas de software.

2. Marco teórico

2.1 Arquitectura de Software

Existen diversos términos que definen la arquitectura de software, los cuales presentan diferentes enfoques según el autor. Según la definición de la Universidad Carnegie Mellon [9], la arquitectura de software se refiere al "sistema que representa las decisiones de diseño relacionadas con la estructura y el comportamiento general del sistema".

La arquitectura de software desempeña un papel fundamental al facilitar la comprensión de un sistema de software, además de mejorar la comunicación y permitir una evaluación oportuna del sistema [10], [11].

2.2 Model View Controller (MVC).

El Modelo-Vista-Controlador es un patrón de arquitectura de software que consta de tres componentes fundamentales. Cada uno maneja la entrada, la salida y la ejecución del sistema de manera independiente [12]. Cuando se introduce un dato de entrada, este modifica una parte específica del sistema y construye las vistas de salida correspondientes. Este enfoque permite representar los componentes y los detalles de sus interacciones, abordando tanto los problemas como las soluciones [12], [13].

El patrón MVC desglosa cada elemento en tres partes: el modelo, que gestiona la lógica de los datos; la vista, que se encarga de la presentación visual; y el controlador, que coordina la interacción del usuario, las entradas y la información en pantalla [13].

2.3 Arquitectura Monolítica.

La arquitectura monolítica se caracteriza por ejecutar todos los procesos en un único entorno. Este enfoque se aplica en situaciones donde se utilizan contenedores y, en caso de necesitar un aumento en la capacidad, se realiza una escalabilidad horizontal al replicar la arquitectura en múltiples servidores [14], [15].

2.4 Arquitectura Orientada a Servicios (SOA).

La arquitectura orientada a servicios (SOA) implica la creación de estructuras para una aplicación que se desglosa en diversos servicios. Entre los más habituales se encuentran los servicios HTTP, que se categorizan como subsistemas o niveles. Esta arquitectura se caracteriza por emplear un enfoque de escalabilidad vertical, mediante la utilización de componentes autónomos que establecen comunicación entre sí [16].

2.5 Arquitectura de Microservicios

La arquitectura de microservicios se caracteriza por la composición de servicios autónomos, los cuales se implementan de forma individual y se vinculan a dominios empresariales, evitando depender de un único almacén de datos [17]. Los posibles errores en un servicio particular no tienen impacto en otras áreas, y la comunicación entre estos servicios se establece mediante protocolos como HTTP/HTTPS, WebSockets o el Protocolo Avanzado de Cola de Mensajes (AMQP) [18]. Este enfoque impulsa la autonomía en el desarrollo, así como la agilidad a largo plazo y la integración continua. Al mismo tiempo, refuerza la resiliencia del sistema ante fallos, permitiendo una adaptación sólida y eficiente en entornos cambiantes [19], [20].

2.6 Seguridad

En el contexto de los microservicios, la seguridad según [21], [22], del IEEE, implica mantener la integridad de la información y del servicio de software. Esto conlleva la protección de los elementos del sistema frente a posibles accesos no autorizados, usos indebidos, modificaciones, destrucciones o divulgaciones.

2.7 Diseño

Según el glosario estándar del IEEE [22]. en términos de ingeniería de software, el diseño se define como el proceso de describir las arquitecturas, componentes, interfaces y características del sistema.

2.8 Modularidad

La modularidad se refiere a un sistema compuesto por mecanismos aislados en los cuales los elementos contribuyen al funcionamiento general del sistema, en contraste con un solo componente que proporciona toda la funcionalidad [22].

2.9 Microservicio

Chris Richardson [23] define el término microservicio como "un repositorio de código pequeño, desacoplado, ágil, escalable, autónomo y desplegado independientemente de los demás". Este concepto presenta características específicas, como la independencia, la autonomía, la agilidad, la versatilidad de tecnologías, el aislamiento, la resiliencia y la escalabilidad.

2.10 Patrones de Microservicios

Los patrones de microservicios son estándares probados y validados que se utilizan para resolver problemas específicos en el contexto de microservicios. Estos patrones pueden abordar cuestiones de estructuración, creación y comportamiento, y entre los más utilizados se encuentran Circuit Breaker y Saga [22]–[25].

2.11 Reusabilidad

La reusabilidad se refiere al grado en que un activo puede ser aprovechado en múltiples sistemas de software o en la construcción de otros activos [26].

2.12 Adaptación

La adaptación implica la modificación de un producto con características y condiciones específicas en un entorno variable, manteniendo la sintonía con los cambios del entorno [7]. Existen dos tipos de procesos de adaptación: el clásico, que implica la toma de decisiones y una secuencia de adaptación, y el basado en componentes, que consta de activación, planificación, ejecución y reconfiguración de la adaptación [22], [26].

2.13 Vulnerabilidad

La vulnerabilidad se refiere a la posibilidad de sufrir daños en los puntos débiles de un sistema de software, donde la seguridad informática no cuenta con las defensas necesarias en caso de un ataque [27], [28].

CAPÍTULO

03

Trabajos Relacionados

Este capítulo presenta los trabajos relacionados, explorando temas como la seguridad en microservicios y otros aspectos relacionados con la seguridad en arquitecturas de software. También se analizan las principales medidas de seguridad

3. Trabajos relacionados

3.1 Tablas comparativas de trabajos relacionados

En las siguientes tablas comparativas de trabajos relacionados, se abordan diversas problemáticas de seguridad. Estas tablas presentan propuestas para mitigar problemas de seguridad en diferentes arquitecturas de software, específicamente aplicables a microservicios. Se han organizado estas tablas de acuerdo al tipo de análisis realizado, y para acceder a los resúmenes correspondientes a cada una, se puede consultar el Anexo A.

La Tabla 1 exhibe múltiples propuestas de seguridad para distintas arquitecturas de software, con enfoque en su aplicabilidad a microservicios. Estas propuestas abarcan aspectos como autorizaciones y servicios de autenticación [4], [5]; identificación y autorización [6]; hardware, virtualización, cómputo en la nube, comunicación, servicio, aplicación y orquestación [23]; mejoras de diseño y certificados de seguridad [27]; diseño [28]; orquestación [30]; comunicación [31]; APIs [32]; capas de arquitectura [33], y violación de acceso [34].

La Tabla 2 muestra algunos trabajos relacionados que analizan y sugieren aplicaciones en el uso, desarrollo, implementación y migración de microservicios seguros, abordando políticas de seguridad [21], descomposición [26] y seguridad de acceso [29].

La Tabla 3 presenta artículos relacionados con propuestas de seguridad, algunas orientadas a patrones de seguridad de microservicios. En estos se identifican soluciones potenciales para abordar la seguridad en microservicios, como la aplicación de PaaS [35], acceso a microservicios [36], comprobación de modelos de seguridad [37], control de acceso [38], despliegue y control [39], y mitigación de riesgo de acceso a microservicios [40].

La Tabla 4 detalla una lista de los principales problemas de seguridad relacionados con microservicios que han sido analizados en diversos artículos. También se presentan propuestas de mitigación para afrontar estos problemas. Es esencial señalar que, aunque se han propuesto soluciones, estas no se consideran totalmente efectivas. En lugar de ofrecer soluciones definitivas, los artículos se enfocan en investigaciones y metodologías que pueden ayudar a resolver las distintas problemáticas según los casos particulares presentes en los entornos de microservicio [41]-[53].

Tabla 1: Propuestas de Seguridad en Arquitecturas de Software Aplicables a Microservicios

En el panorama de las arquitecturas de software, las propuestas de seguridad han emergido como un enfoque fundamental para garantizar la integridad y confidencialidad de los sistemas. La Tabla 1 compendia una gama diversa de propuestas, centradas en arquitecturas de microservicios. Desde autorizaciones y servicios de autenticación [4], [5], hasta capas de arquitectura [33] y violaciones de acceso [34], se presentan soluciones que abordan los retos inherentes a la seguridad en estos entornos.

Tabla 1 Cuadro comparativo de trabajos relacionados de propuestas de seguridad para distintas arquitecturas de software aplicables a microservicios.

Ref.	Artículo	Problemática	Mecanismos de Seguridad	Contribución y Relevancia
[4]	Authentication and authorization in microservice-based systems: survey of architecture patterns	Authentication and authorization	Patrones descentralizados, Centralized pattern with single policy decisionpoint, Centralized pattern with embedded policy decision point.	Examina una amplia gama de amenazas de seguridad en entornos de microservicios y propone contramedidas para mitigar estos riesgos.
[5]	Authentication and authorization of end user in microservice architecture	Authentication and authorization	SSO, JSON TOKEN, JWT, API GATEWAY, cross-origin resource sharing (CORS) y Cross-Site Request Forgery (CSRF).	Presenta un enfoque integral para la gestión de identidad y acceso en arquitecturas de microservicios, utilizando estándares y protocolos ampliamente adoptados.
[6]	Authentication and	Authentication and	OAUTH2, SSO, JSON Web Token	Proporciona una solución

Ref.	Artículo	Problemática	Mecanismos de Seguridad	Contribución y Relevancia
	authorization orchestrator for microservice-based software architectures	authorization	(JWT), OpenID, TLS – PSK, MAC, DAC, RBAC, ABAC, XML SAML.	para asegurar la comunicación entre microservicios utilizando una puerta de enlace de API y tokens JWT.
[23]	Overcoming Security Challenges in Microservice Architectures.	Comunicación entre microservicios	Domain Driven Design, APIS WEB, dockers, Denial of Service, Circuit breaker, Security tokens, OpenID, single sign-on, Security Token Service (STS), WS-Trust standard, WS-Security standard, validating security tokens, JSON Web Signature (JWS), Encryption (JWE), Token (JWT), OAuth 2.0 y Netflix MTLS,	Realiza una revisión sistemática de la literatura sobre autorización y control de acceso en microservicios, identificando diferentes enfoques y modelos utilizados en la comunidad de investigación.
[27]	Giving IoT Services an Identity and Changeable Attributes	Control de acceso	Certificados X.509v3, claves públicas, executable Hash y nodos de distribución de red.	Resume las mejores prácticas de seguridad para microservicios identificadas en la literatura científica y proporciona una guía para desarrolladores y

Ref.	Artículo	Problemática	Mecanismos de Seguridad	Contribución y Relevancia
				arquitectos.
[28]	Security and privacy for cloud-based data management in the health network service chain: a microservice approach.	Seguridad y privacidad de datos distribuidos	APIS, estándares de Servicio de Distribución de Datos (DDS), ICT en HDMI, API Gateway, XML JSON REST, MVC pattern, cloud service providers (CSPs), OMG DDS standard.	Examina los desafíos de seguridad específicos asociados con la implementación de microservicios en contenedores y proporciona una visión general de las contramedidas recomendadas.
[30]	CAVAS: Neutralizing Application and Container Security vulnerabilities in the Cloud Native Era.	Capas de aplicación y accesos internos	Security Gateway, APIs restful, Web Application Security Consortium (WASC) Common Weakness Enumeration (CWE), métodos de autentificación automatizados y monitores de vulnerabilidad.	Presenta un enfoque para mitigar vulnerabilidades de seguridad en aplicaciones y contenedores en entornos nativos de la nube, mediante el uso de un security gateway, APIs RESTful y monitores de vulnerabilidad.
[31]	Dokspot – Securely Linking Healthcare Products with Online	Gestión de instrucciones y	Protocolos de comunicación segura TLS, control de acceso basado en roles,	Proporciona una solución segura para vincular productos de atención

Ref.	Artículo	Problemática	Mecanismos de Seguridad	Contribución y Relevancia
	Instructions.	acceso	funciones criptográficas, DNS, CloudFlare, cifrado de extremo a extremo, protocolos de contraseña remota segura.	médica con instrucciones en línea, utilizando mecanismos de seguridad como protocolos de comunicación segura, control de acceso y cifrado de extremo a extremo.
[33]	Patrones de Seguridad Software en el Contexto de la Arquitectura Multicapa para la Plataforma J2EE.	Capa de aplicación y seguridad de datos	WS-Security, Patrones Web-Tier, Secure Base Action, Secure Logger, Business-Tier, container managed security, dynamic service management, obfuscated transfer object, policy delegate, secure service facade, secure session object, Assertion Builder, Credential Tokenizer, single sign-on delegator y certificados X.509	Proporciona una colección de patrones y mecanismos de seguridad para abordar la seguridad en diferentes capas de aplicación y datos en la plataforma J2EE, incluyendo autenticación, autorización, gestión de sesiones y comunicaciones seguras.
[34]	HawkEDA: A Tool for Quantifying Data Integrity Violations in Event-driven	Aislamiento de fallos y violaciones de integridad	REST API Helper, application eShopContainers, JSON y ScenarioInterface.	Presenta una herramienta llamada HawkEDA para cuantificar violaciones de integridad de datos en microservicios basados en

Ref.	Artículo	Problemática	Mecanismos de Seguridad	Contribución y Relevancia
	Microservices.			eventos. Utiliza mecanismos como REST API Helper, application eShopContainers, JSON y ScenarioInterface para mejorar el aislamiento de fallos y garantizar la integridad de los datos en entornos de microservicios.

Tabla 2: Microservicios Seguros: Análisis y Aplicaciones

La Tabla 2 examina trabajos que se sumergen en la exploración de microservicios seguros desde diferentes perspectivas. Desde políticas de seguridad [21] hasta descomposición [26] y seguridad de acceso [29], se examinan análisis que trascienden el mero concepto para abordar la implementación y aplicación de medidas protectoras.

Tabla 2 Cuadro comparativo de trabajos relacionados de análisis y sugerencias de aplicación en el uso, desarrollo, implementación y migración de microservicios seguros.

Ref.	Artículo	Resumen de trabajos relacionados / Propuesta	Contribución y Relevancia
[21]	Implementación de Patrones de Microservicios	JSON Web Tokens JWT, Herramientas ReactJs con CSS en HTML, Node JS, patrones circuit Breaker en Java y SAGA.	Mejora de la seguridad y escalabilidad de los microservicios.
[26]	Microservices Migration Patterns.	(DDD), RESTFull APIs, Situational Method Engineering (SME), and third-party libraries, a Bounded Context, a Service Discovery, Edge Server, a Load Balancer, Circuit Breaker, implementada con Hystrix, Zuul, dockers, Mesos+Marathon y Kubernetes.	Facilita la migración exitosa de sistemas monolíticos a arquitecturas de microservicios.
[29]	Security in Microservices Architectures	Active Directory, OpenID, o Lightweight Directory Access Protocol (LDAP), TOGAF, ITIL, COBIT, BSIMM, MPSL, cortafuegos de carácter local y perimetral.	Refuerzo de la protección de datos y control de acceso en entornos de microservicios.

Tabla 3: Propuestas de Seguridad: Patrones en Microservicios

Los patrones de seguridad son los cimientos sobre los cuales se construyen sistemas robustos en la era de los microservicios. La Tabla 3, muestra un compendio de propuestas de seguridad orientadas a patrones específicos para microservicios. Desde la aplicación de PaaS [35] hasta la mitigación de riesgo de acceso a microservicios [40], Se examina cómo estos patrones ofrecen soluciones ingeniosas para desafíos complejos. A medida que trazamos este recorrido, se descubre cómo la adaptación de patrones puede guiar hacia un terreno seguro en la implementación de microservicios.

Tabla 3 Cuadro comparativo de Artículos relacionados de propuestas de seguridad sugeridas y orientadas a patrones de seguridad de microservicios.

Ref.	Título	Propuestas de solución	Adaptación	Reusabilidad	Estado de la investigación
[32]	Security Patterns for Microservices Located on Different Vendors	Amazon AWSMicrosoft Azure, Google App Engine 3rd Party Communication, AGENCY GUARD, AGENT AUTHENTICATOR, Application Firewall, Cloud Access Security, Broker Integration, Reverse Proxy.		x	Propuesta
[36]	Software Development Activities for Secure Microservices	Implementación de SOA DevOps, orchestration, API Gateway, a service registry, Docker containersREST interface, WS-Security Middleware.		x	Propuesta
[37]	Automated Security Analysis for Microservice	Service Oriented Architecture (SOA), herramientas de modelado Ontology Web Language OWL and Architecture Description Language ADL Query Responsibility		x	Propuesta

Ref.	Título	Propuestas de solución	Adaptación	Reusabilidad	Estado de la investigación
	architecture.	Segregation (CQRS).			
[38]	A survey on security issues in services communication of Microservices-enabled fog applications.	OAuth, SELinux, Spring Cloud Security framework, cifrado avanzado y de clave compartida y algoritmos de encriptación AES y RSA.		x	Propuesta
[39]	Integrating Continuous Security Assessments in Microservices and Cloud Native Applications.	Open API,Tokens WEB JSON, Software Defined Networks (SDN).Continuous Development (CD), API Gateway, Security Enforcement Points (SEP), application firewalls, network routers, WS-SecurityPolicy, OWASP Ruby Cheatsheet, Service-Oriented Architectures (SOA),Web Services Description Language (WSDL), Web Application Description Language (WADL), OpenAPI, RAML Security tools, metrics CVES and CWEs, Firewall-as-a-Service (FWaaS), Integration of vulnerability information into Intrusion Detection Systems (IDS), security Information and Events Management (SIEM),		x	Propuesta

Ref.	Título	Propuestas de solución	Adaptación	Reusabilidad	Estado de la investigación
		Security-as-a-Service (SecaaS), OpenStack Newton			
[40]	Actual Use of Architectural Patterns in Microservices-based Open-Source Projects.	Service-oriented architecture (SOA), APT Gateway, Container, Log aggregator, Database is the service, Enable cont. integration, Result cache, Page cache, Backend for Frontend, Scalable store.		x	Propuesta
“Adaptación de un patrón de software en seguridad a la arquitectura de un Microservicio”		Adaptar un patrón de alguna arquitectura de software, monolíticas, orientadas a objetos, orientadas a servicios.	x		Adaptación

Tabla 4: Problemas de Seguridad y Propuestas de Mitigación en Microservicios

En el tejido de las arquitecturas de microservicios, los problemas de seguridad presentan desafíos ineludibles. En la Tabla 4 se explora un análisis detallado de los problemas clave de seguridad que enfrentan estos entornos dinámicos. Desde la exposición a riesgos hasta vulnerabilidades potenciales, se aborda la realidad de estos problemas. A través de propuestas de mitigación, se delinean caminos hacia la resiliencia. Sin embargo, se nota que la evolución constante exige una respuesta adaptable. A medida que se navega por estos tópicos, se comprende la importancia de una vigilancia continua en la protección de sistemas basados en microservicios.

Tabla 4 Cuadro de trabajos relacionados en Problemas de Seguridad y Propuestas de Mitigación en Microservicios.

Ref.	Artículo	Problemática	Propuesta de mitigación
[41]	Towards Concurrent Audit Logging in Microservices.	Problemática en los registros de auditoria	Aplicación a través del OPEN WEB APPLICATION Security Project.
[42]	Recommendations for Enhancing Security in Microservice Environment Altered in an Intelligent Way.	Problemática en los procesos de desarrollo y diseño de microservicios	Administrator Hierarchy, Building the Server from the Ground Up, The Checkpointed Systems y Controlled Execution Environment.
[43]	Security Design Patterns in Distributed Microservice Architecture	Problemática en las técnicas de diseño e implementación de tecnologías heterogéneas.	Mecanismos de autenticación y autorización, tokens de seguridad, con formatos RESTful, JSON, Oauth 2.0, OPENID y pasarelas API privadas.
[44]	Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications.	Problemática en la implementación de los patrones de diseño	Dockers, PaaS o IaaS o la función de ambos creando una nube OpenSatack, implementación de API Gateway con Framework Spring Cloud, orquestación nativa en la nube.
[45]	Interface quality patterns - Communicating and improving the quality of microservices APIs	Problemática en los procesos de calidad	API Key, Wish List, Rate Limit, Rate Plan, Service Level Agreement.

Ref.	Artículo	Problemática	Propuesta de mitigación
[46]	Analyzing the Relevance of SOA Patterns for Microservice-Based Systems.	Problemática en la aplicación de patrones	Enterprise Inventory, Protocol Bridging, pattern Lightweight Endpoint, Lightweight Communication, patrones SOA, y REST-inspired patterns.
[47]	Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API.	Problemática en el acceso y adaptación de recursos.	OAuth2 y Spring Security Framework.
[48]	Assuring the Evolvability of Microservices: Insights into Industry Practices and Challenges.	Deuda técnica y arquitectura.	Event-Driven Messaging, Service Registry, Strangler, Backends for Frontends, Consumer-Driven Contracts, Tolerant Reader, API Gateway, Messaging.
[49]	Guidelines for Adopting Frontend Architectures and Patterns in Microservices-Based Systems.	Problemática en la aplicación de modelos de calidad y prototipos de aplicación.	SPA XON, pasarelas APIs basadas en BFF, API Gateway, backends for frontends, supplements self-contained systems (SCS).
[50]	How Microservices are Changing the Security Landscape.	Problemática en las comprobaciones de seguridad.	API Gateway Pattern, Sidecar Pattern, OpenAPI, REST APIs, API key-based, OAuth 2.0, OpenID Connect Discovery, frontends pattern, JWT, WebSocets, mTLS with X.509 certificates, JWS y JWE.
[51]	Identifying Availability Tactics to Support Security Architectural Design of Microservice-based Systems.	Problemática en los procesos de calidad, integridad, confidencialidad y disponibilidad.	Circuit breaker, Service Registry, Messaging.
[52]	Detecting Cyber Security Attacks against a Microservices Application using Distributed Tracing.	Ataques de ciberseguridad.	DeathStarBench, Docker, Thrift, Jaeger, ElasticSearch.

Ref.	Artículo	Problemática	Propuesta de mitigación
[53]	Decision Models for Selecting Patterns and Strategies in Microservices Systems and their valuation by Practitioners.	Problemática en los procesos de diseño	Service-level authorization, Scenario-based analysis, Edge-level authorization, API rate limiting, Encrypt and protect secrets, and Scan dependencies strategies, API Gateway, Backend for frontend (BFF).

La observación predominante es que la mayoría de las soluciones sugeridas por los autores giran en torno al aprovechamiento de propuestas de seguridad provenientes de otras arquitecturas de software. Esto sugiere que las áreas más propensas a vulnerabilidades son las de acceso y diseño, una conclusión compartida por distintos autores. Estas áreas se identifican como puntos débiles donde la seguridad puede ser comprometida, y es precisamente aquí donde se plantea la propuesta de solución específica para este proyecto de tesis, denominado "Adaptación de un patrón de seguridad de software a la arquitectura de Microservicios". Esta iniciativa tiene como objetivo abordar la fragilidad en la seguridad al implementar un patrón de seguridad adaptado a la arquitectura de microservicios. A medida que se explora esta propuesta de solución, se vuelve evidente que la adaptación y personalización de enfoques de seguridad existentes pueden desempeñar un papel crucial en fortalecer las defensas y mejorar la resiliencia en este entorno de arquitectura distribuida.

CAPÍTULO

04

Implementación de la solución

En este capítulo, se expone la implementación de la solución, detallando el proceso de análisis, selección y adaptación del patrón de seguridad destinado a arquitecturas de microservicios.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

4.1 Creación/adaptación de un patrón de seguridad para microservicios.

4.1.1 Patrón

Christopher Alexander menciona con respeto a patrones que [29]: "*Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese problema, de tal manera que se puede utilizar esta solución un millón de veces, sin hacerlo nunca de la misma manera dos veces.*"

Aunque este término se acuña en el contexto de la construcción de edificios y ciudades, su alcance es tan amplio que se aplica a definiciones arquitectónicas en diversos campos, incluyendo las arquitecturas de software. En este contexto, en lugar de tratar con objetos tangibles como paredes y estructuras, se interactúa con interfaces y objetos en una arquitectura orientada a objetos. Los patrones comparten la misma premisa: abordar problemas recurrentes y proporcionar soluciones específicas [30].

Un patrón de software está compuesto por elementos para el diseño de software, ofreciendo soluciones a problemas particulares. Los patrones de microservicios se definen por considerar las características arquitecturales de los microservicios, tales como servicios distribuidos, bajo acoplamiento, uso de múltiples lenguajes y bases de datos propias, manteniendo el principio de responsabilidad única [24].

4.1.2 Adaptación

La adaptación implica codificar un producto con características y condiciones específicas en un entorno que es susceptible de cambios. Estos ajustes son esenciales para mantenerse al ritmo de las transformaciones en ese entorno [22]. En este contexto de adaptación, se presenta el patrón de seguridad diseñado para microservicios, partiendo de los elementos cruciales que definen los patrones de microservicios. Uno de estos elementos es la API Gateway, que desempeña la función de gestionar y distribuir el acceso a los distintos microservicios [31].

Al abordar el diseño y desarrollo del patrón propuesto, se toman en cuenta los componentes esenciales necesarios para su adaptación. Se considera el patrón de API Gateway como un marco general, en el cual se presentan conceptos como predecesores, sucesores y mejoras de un API Gateway. Además, se detalla la propuesta de adaptación específica del patrón de seguridad para microservicios [32].

En el siguiente apartado, se exponen con detalle los componentes fundamentales que asumen una importancia primordial en el proceso de generación del patrón de seguridad MSPAG. Estos elementos clave

desempeñan un papel esencial en la estructuración y concepción de este patrón específico, contribuyendo a comprender plenamente su formulación y su aplicabilidad en el ámbito de la seguridad en microservicios.

4.1.3 Creación de patrones

En [30], "Design Patterns: Elements of Reusable OO Software" (CD), Christopher Alexander propone que un patrón consta de cuatro elementos específicos: nombre del patrón, problema, solución y consecuencias. Por otra parte, en [32], "PATTERN LANGUAGES," Alexander propone elementos como patrón, problema, restricciones y solución.

4.2 Componentes para la adaptación de un patrón

4.2.1 Esquema de adaptación de patrón

Después de haber tomado en cuenta los elementos fundamentales para la creación de un patrón, se procede a identificar los elementos predecesores, sucesores y a señalar la mejora deseada. A continuación, en la Figura 1, se presenta un esquema de adaptación que muestra la secuencia de adaptación con los patrones: predecesor, sucesor y mejora [23].

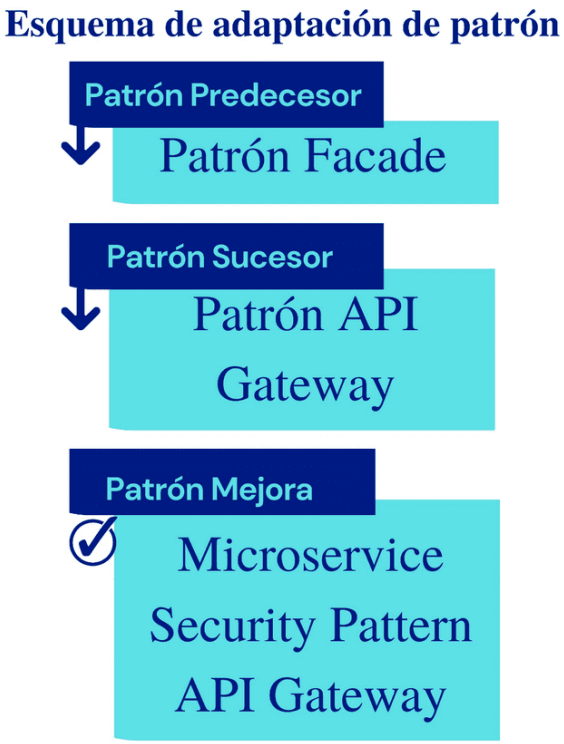


Figura 1 Esquema de adaptación del patrón MSPAG.

4.3 Patrón Predecesor

Toda evolución tecnológica tiene sus raíces en un elemento específico que requiere adaptación. En esta línea, se presenta el patrón Facade como el punto de partida, que precedió al patrón de API Gateway y sentó las bases para su desarrollo posterior [23], [30].

4.3.1 Patrón Facade

Este patrón de diseño, clasificado como estructural, establece un conjunto de interfaces para sistemas previamente existentes. Su objetivo es simplificar la funcionalidad de dichos sistemas y proporcionar una interfaz de usuario que permita la independencia para una respuesta oportuna [2].

4.3.2 Estructura de patrón Facade

La Figura 2, presenta la estructura de patrón Facade, en la cual se observa una interfaz que se organiza en subsistemas. Este patrón se utiliza para definir un punto de entrada a cada nivel de subsistema [33]. Si los subsistemas son dependientes, las dependencias entre ellos pueden simplificarse al hacer que se comuniquen entre sí únicamente a través de sus respectivas fachadas [30].

Es esencial tener presente que este patrón de diseño juega un rol crucial como elemento predecesor en la implementación del MSPAG (Microservices Security Pattern API Gateway).

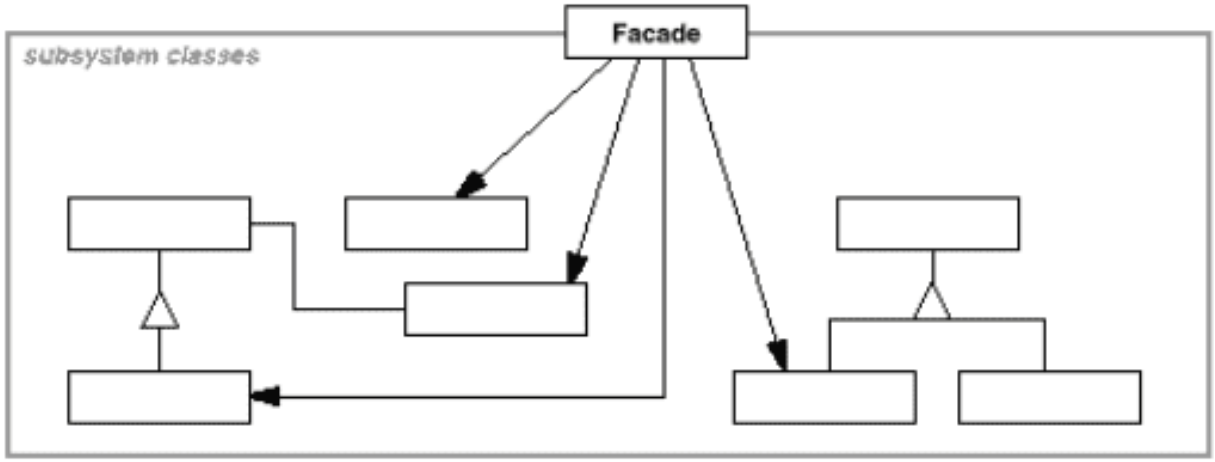


Figura 2 Estructura de patrón Facade [Java Design Patterns, 2022].

4.4 Patrón Sucesor

4.4.1 API GATEWAY

El patrón sucesor que se presenta es la API GATEWAY. Este concepto se refiere a un enrutador de servicios y clientes que opera como el frente de una API. Guarda similitudes con el patrón Facade, ya que monitorea, gestiona tareas, y se encarga del enrutamiento y supervisión de los servicios [34]. El propósito fundamental de este patrón es mantener la conexión entre todos los servicios, y por lo general, funciona como el único punto de entrada [35]. Además de esto, se puede entender como una arquitectura modular que consta de dos capas. Estas capas se componen de módulos independientes, los cuales se encargan de gestionar la implementación de un cliente específico. Dichos módulos desempeñan una variedad de funciones, como autenticación, asignación de valores y direccionamiento de solicitudes [36]. Una representación gráfica de este patrón se presenta en la Figura 3, en la que se muestra de forma abstracta, la estructura del patrón API Gateway.

4.4.2 Estructura de patrón API Gateway

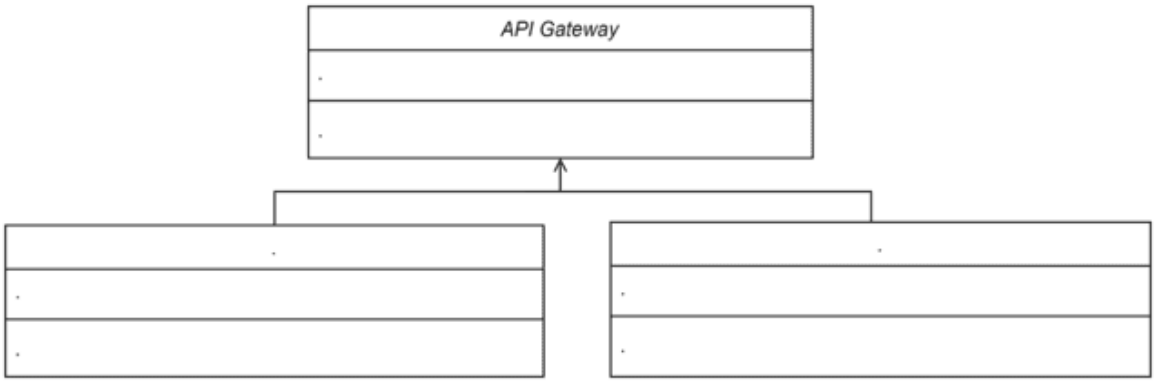


Figura 3 Estructura de patrón API Gateway [Richardson, 2017].

Las puertas de enlace (APIs) ocupan una posición intermedia entre un usuario y una colección de microservicios, desempeñando tres funciones clave [37]:

- Enrutamiento de solicitudes: Una puerta de enlace (API) recibe una solicitud nueva, la divide en múltiples solicitudes, consulta un mapa de enrutamiento para determinar el destino de cada solicitud y las direcciona hacia el o los microservicios internos correspondientes [36].
- Composición de API: La puerta de enlace (API) lleva a cabo la orquestación del flujo de trabajo o proceso, recopila la

información requerida de varios microservicios, agrupa los datos y los devuelve al solicitante de manera compuesta [36].

- Traducción de protocolos: Las puertas de enlace (APIs) son conscientes de que las solicitudes provienen de dispositivos que utilizan diversos protocolos [38]. Ayudan a que las solicitudes de los clientes y los microservicios se comuniquen, traduciendo los protocolos correspondientes. Por ejemplo, una red de área amplia (WAN) y una red de área local (LAN) funcionan de manera diferente y tienen diferentes necesidades de API [36]. Al retornar la información, la puerta de enlace la adapta y la envía de vuelta a los solicitantes en una forma comprensible. Si un microservicio responde en XML, pero la solicitud se efectuó en formato JSON, la puerta de enlace realiza automáticamente la traducción. Una API REST, por ejemplo, emplea el protocolo HTTP para solicitar servicios [18], [36].

4.5 Mejora

4.5.1 Patrón de seguridad para microservicios

Tras un análisis exhaustivo de la literatura y en concordancia con las diversas tablas comparativas presentadas en la sección de trabajos relacionados, se han seleccionado las medidas de seguridad más utilizadas en arquitecturas de software, como lo son JWT, API Gateway y algoritmos criptográficos H256, para su adaptación en una arquitectura de microservicios [5], [6], [12], [18], [24], [30], [32], [35], [39]–[43].

Se ha considerado el patrón de API Gateway y se ha implementado con los siguientes elementos: JSON Web Token (JWT) y el algoritmo de codificación hash (H256) [5], [6], [12], [18], [24], [30], [32], [35], [39]–[43].

4.5.2 JWT

Un JWT es un método de intercambio de datos que se mantiene en formato JSON. Este mecanismo permite la protección digital de la comunicación, logrado a través de una cadena de codificación que se basa en tres elementos: encabezado (header), carga útil (payload) y firma (signature), descritos en la Figura 4 [26]:

Figura 4 Descripción de elementos del JWT.

4.5.3 Hash

El hash es un algoritmo de reducción criptográfica que comúnmente se implementa para el manejo de contraseñas y sistemas de verificación de datos [44]. El algoritmo SHA (Secure Hash Algorithm) transforma una secuencia de elementos de longitud fija en una función que produce otro elemento. Esta función recibe elementos de entrada y los convierte en un rango de valores de salida finito, como el caso de Secure Socket Layer (SSL) [44]. Entre las variantes de SHA se encuentran SHA-0, SHA-1, SHA-2 y SHA-3, siendo este último empleado con frecuencia en sistemas de criptomonedas, a continuación, se observa un ejemplo del código en la Figura 5 [44].

Hash

```
hashMap.put("A", 1);
if (hashMap.containsKey("A")) {
 System.out.printf("Contiene la clave A. Su
valor es: %d\n", hashMap.get("A"));
}
if (hashMap.containsValue(0)) {
 System.out.println("Contiene el valor 0");
}
```

Figura 5 Ejemplo del código Hash.

4. 6 Implementación de patrón de seguridad en microservicio.

4.6.1 Patrón de seguridad para microservicios

Para llevar a cabo la implementación del patrón de seguridad en microservicios, se requiere crear una API Gateway en la que se incorpore el uso del JWT con el algoritmo Hash.

4.7 Patrón de seguridad implementado.

4.7.1 Patrón de seguridad: Microservice Security Pattern API Gateway (MSPAG).

Basándose en los elementos propuestos por distintos autores para la construcción de patrones, se identifican similitudes que comparten características específicas, conduciendo a la creación o adaptación de un patrón. Para obtener detalles más profundos sobre la implementación y las pruebas del MSPAG en lenguajes de programación Python y JAVA, se recomienda revisar los anexos B, C, D, E y F de este documento.

Siguiendo la propuesta de Alexander [29], se incluyen los siguientes elementos:

- Nombre del patrón
- Problema
- Solución
- Consecuencias

Además, en consonancia con la sugerencia de Richard [32], se consideran los siguientes elementos:

- Patrón
- Problema
- Restricciones
- Solución

En las consideraciones preliminares para la creación de un patrón, se contemplan aspectos esenciales para la adaptación del patrón de seguridad de microservicios, como se muestran en la Tabla 5.

Tabla 5 Aspectos Fundamentales en la Elaboración de un Patrón Arquitectónico de Software.

Aspectos Fundamentales en la Elaboración de un Patrón Arquitectónico de Software

1 | PATRÓN →	2 | PROBLEMA →	3 | SOLUCIÓN →	4 | RESTRICCIONES ✓
Microservice Security Pattern API Gateway	Acceso no autorizado	Adaptación de una API Gateway mediante la Implementación de un JWT y Algoritmo de Codificación Hash	Patrón Arquitectónico Específico para Mitigar Accesos No Autorizados en Microservicio

4.8 Categoría

El patrón de API Gateway es una adaptación del patrón Facade, el cual es un patrón de estructura, que describe la forma común en que los objetos se pueden organizar para trabajar con otros. Por lo que se deduce que un API Gateway es un patrón de estructura [45].

4.9 Visión general

El MSPAG es un patrón de seguridad que está directamente enfocado a la arquitectura de software de microservicios, éste describe una pauta para la protección de la puerta de enlace de las APIs de microservicios, frente a amenazas externas en el proceso de comunicación de microservicios, considerando que una arquitectura de microservicios se caracteriza por su distribución en aplicaciones diversas, componentes desacoplados y comunicación por protocolos HTTP/S [30].

Por lo que el **Microservice Security Pattern API Gateway,** permite la protección de la puerta de enlace de los microservicios, contra las posibles vulnerabilidades en los end points.

4.10 Desafíos típicos.

- Dado que los microservicios se caracterizan por su débil acoplamiento aumenta la complejidad de protección.
- Los End points quedan expuestos por la cantidad de interacciones para los múltiples consumidores del recurso, los cuales se aprovechan por clientes internos o públicos.

- Dado que una arquitectura de microservicios (AMS) proporciona una holgura de desarrollo más rápida y ágil, suelen presentarse una serie de fallas lógicas difíciles de controlar por sus sistemas distribuidos.

4.11 Propuesta

Implementación de JWT con algoritmo de codificación hash en el patrón de API Gateway.

Estas medidas de seguridad serán adaptadas al patrón de microservicios API Gateway, para crear un patrón de seguridad, diseñado específicamente para una arquitectura de microservicios y proporcionar una medida de seguridad en los puntos de acceso a los microservicios a continuación, en la Figura 6, se muestra de forma gráfica la adaptación de MSPAG.

Figura 6 Implementación de API Gateway con JWT y algoritmo hash.

4.11.1 Descripción de elementos del MSPAG en lenguaje alejandrino

A continuación se describe en lenguaje alejandrino los elementos del MSPAG que constan de: nombre, intención, motivación, aplicabilidad, estructura, consecuencias y detalles de implementación, tomando como referencia el libro Design Patterns: Elements of Reusable Object-Oriented Software [30], en la Figura 7 se muestra la escripción de elementos del MSPAG en lenguaje alejandrino.

Microservice Security Pattern API Gateway

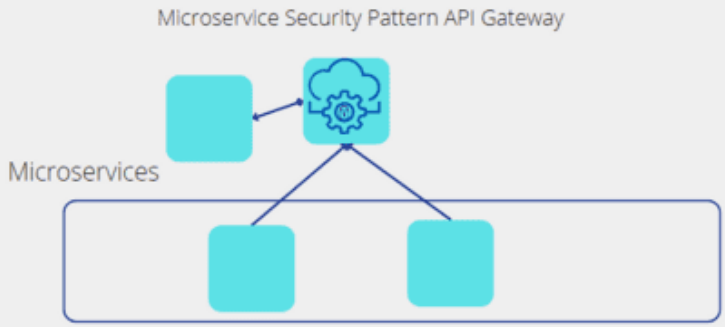
1 | NOMBRE →	Microservice Security Pattern API Gateway.
2 | INTENCIÓN →	Proporcionar una puerta de enlace API unificada a un conjunto de microservicios de un subsistema. El Microservice Security Pattern API Gateway. Define una puerta de enlace de nivel superior que provee una medida de seguridad para el uso de microservicios.
3 | MOTIVACIÓN →	Estructurar un sistema. con una sola puerta de enlace a un sistema de microservicios ayuda a reducir las problemáticas de seguridad en microservicios. Un objetivo común de diseño y seguridad en microservicios es minimizar la comunicación y las dependencias entre subsistemas. así como mantenerlas seguras. Una forma de lograr este objetivo es introducir un objeto Microservice Security Pattern API Gateway. que proporcione una puerta de entrada única y simplificada para el acceso a los diversos microservicios.
4 | APLICABILIDAD →	Desea proporcionar una puerta de enlace API implementada con algoritmos de seguridad Hash y JWT para proporcionar un acceso seguro a microservicios
5 | ESTRUCTURA →	
6 | CONSECUENCIAS →	El patrón Microservice Security Pattern API Gateway. ofrece las siguientes ventajas: a. Proporciona el acceso a los diferentes microservicios facilitando el uso de la puerta de enlace. b. Favorece el débil acoplamiento entre microservicios y clientes. c. Proporciona una medida de seguridad en el proceso de autenticación del API Gateway al microservicio.
7 | DESTALLES DE IMPLEMENTACIÓN →	El patrón Microservice Security Pattern API Gateway. Involucra la implementación de JWT y algoritmos de codificación H256 para la protección de la puerta de enlace API Gateway.

Figura 7 Descripción de elementos del MSPAG en lenguaje alejandrino.

4.12 Adaptación de patrón

La implementación del patrón se compone por cuatro fases, la primera es la codificación de una API Gateway, la segunda es la codificación del JWT, la tercera es la codificación del algoritmo hash y la cuarta es la unión de elementos, como se presenta en la Tabla 6.

Tabla 6 Flujo de adaptación del patrón MSPAG

Flujo de adaptación del patrón MSPAG

1 | API GATEWAY →	2 | JWT →	3 | AGORITMO HASH →	4 | UNIÓN ✓
Codificación de API Gateway	Codificación de JWT	Codificación de algoritmo hash 256	Unión de elementos para la implementación del patrón de seguridad MSPAG aplicable a microservicios

4.13 Diagrama de clases Microservice Security Pattern API Gateway

Descripción del diagrama de clases MSPAG y su funcionamiento:

El diagrama representa un sistema que utiliza una API Gateway para gestionar el acceso a dos microservicios diferentes, utilizando un JWT (JSON Web Token) encriptado con el algoritmo H256 para generar y validar los tokens.

14.13.1 Clase API Gateway:

Esta clase representa la puerta de entrada a los servicios del sistema.

Tiene una relación de composición con la clase TokenManager y una relación de asociación N-aria con la clase Microservicio y contiene los métodos que se presentan en la Figura 8 [46]–[49].

Figura 8 Métodos de API Gateway.

14.13.2 Clase TokenManager:

Esta clase se encarga de generar y validar los tokens JWT utilizados en el sistema. Tiene una relación de composición con la clase API Gateway, y contiene los métodos que se presentan en la Figura 9.

Figura 9 Clase TokenManager.

14.13.3 Clase Microservicio:

Esta clase representa el microservicio del sistema. Tiene una relación de asociación N-aria con la clase API Gateway, y contiene los métodos que se presentan en la Figura 10.

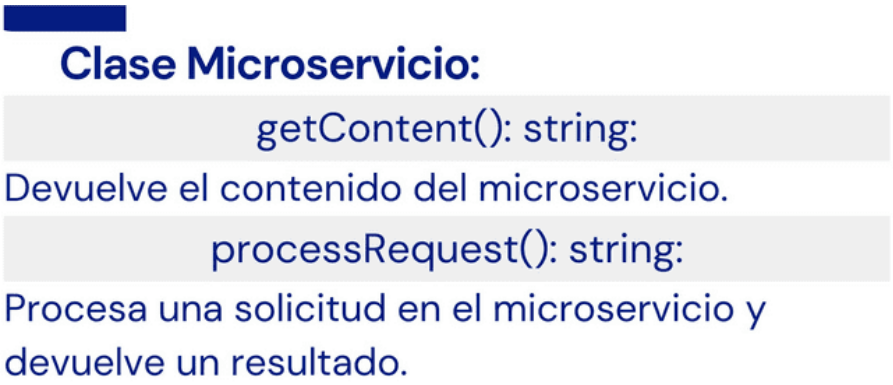


Figura 10 Clase Microservicio.

14.13.4 Funcionamiento:

Cuando un cliente realiza una solicitud a través de la API Gateway, el token enviado junto con la solicitud se valida utilizando el método validateToken() de la clase API Gateway.

La clase API Gateway utiliza el TokenManager para generar y validar tokens JWT. Esto implica invocar los métodos generateToken() y validateToken() de la clase TokenManager.

Una vez que el token se valida con éxito, el cliente puede acceder al microservicio, mediante los métodos accessMicroservicio(), de la clase API Gateway, respectivamente.

La clase API Gateway establece una comunicación con el microservicio a través de las asociaciones existentes. Esto implica invocar los métodos getContent() y processRequest() de la clase Microservicio según sea necesario.

El microservicio realiza las operaciones correspondientes y devuelve los resultados a la clase API Gateway, que a su vez los entrega al cliente que realizó la solicitud.

En la sección 14.13.5, Figura 11 se muestra el diagrama de clases UML, que muestra la adptación del MSPAG con la API Gateway para gestionar el acceso a microservicios y validar tokens JWT encriptados con H256. Donde la API Gateway interactúa con el Token Manager y los microservicios, permitiendo el acceso a sus contenidos y procesando las solicitudes enviadas por los clientes para mostrar el contenido de cada microservicio sin importar el lenguaje de programación ni el IDE de desarrollo que el desarrollador elija para implementarlo.

14.13.5 Diagrama de clases Microservice Security Pattern API Gateway

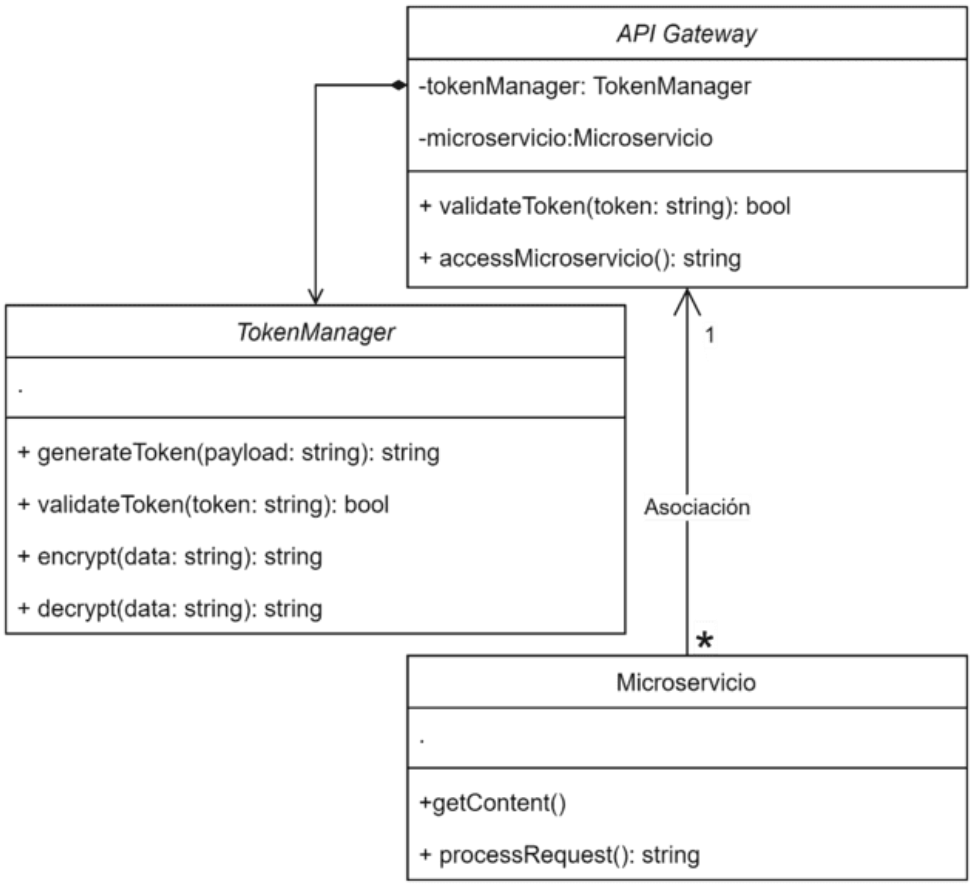


Figura 11 Diagrama de clases MSPAG.

CAPÍTULO

05

Pruebas

En este capítulo se exponen las pruebas realizadas y los resultados obtenidos al implementar el patrón de seguridad de microservicios en tres lenguajes de programación distintos.

5.1 Plan de pruebas

Revise anexo G para mayor detalle.

5.2 Condiciones de aplicación

En la Tabla 7 se muestran los requisitos técnicos mínimos para ejecutar las pruebas del Patrón de seguridad: "MSPAG" descritos en esta investigación.

Tabla 7 Requisitos de aplicación para la implementación del MSPAG

Ambiente	
Hardware	**Software**
• AMD Ryzen 3 3250U with Radeon Graphics • CPU 2.60 GHz • 12,0 GB (9,88 GB usable)	• Sistema operativo Windows 11. • Visual Studio Code Versión: 1.75. • Node.js: 16.14. • JDK 17 • Python 3.11.3 • C# 25.9 • Postman 10.14.9

5.3 Personal para aplicación

Los roles involucrados en la aplicación del presente plan de pruebas son los siguientes:

Autora: Responsable de la aplicación del plan de pruebas, además de generar los reportes de las mismas.

5.4 Riesgos y contingencias

En la Tabla 8, se enlistan los riesgos y contingencias para la ejecución del presente plan de pruebas.

Tabla 8 Riesgos y contingencias.

No.	Riesgo	Contingencia
1	No disponibilidad del ambiente de trabajo.	Se debe replicar con las versiones correspondientes.
2	Planeación incorrecta en el diseño de los casos de prueba.	Rediseñar los objetivos para cumplir con las pruebas en el tiempo establecido.
3	El proyecto de prueba no se encuentra funcionando para llevar a cabo las pruebas.	Corregir el MSPAG.
4	El intervalo definido para el Patrón de seguridad: "MSPAG" no protege el acceso al microservicio.	Documentar los errores generados para un posible ajusté en futuras implementaciones.

5.5 Implementación de patrón MSPAG en tres lenguajes de programación

Las pruebas del Patrón de seguridad: "MSPAG," se elaboran en tres etapas, todas de forma local.

En la primera etapa, se crea una API Gateway y se corrobora el funcionamiento correcto.

En la segunda etapa, se crea el JWT, con el algoritmo HASH y se corrobora el funcionamiento correcto.

En la tercera etapa, se integran los elementos de seguridad a la API Gateway creada, para conformar el MSPAG y se corrobora el funcionamiento correcto, como se observa en la Tabla 9.

Tabla 9 Sistema de pruebas del MSPAG.

ETAPA	Descripción
1	Se pudo verificar que cuando el usuario tiene conocimiento de la ruta o URL del microservicio y la ingresa en el buscador, la puerta de enlace API Gateway niega el acceso.

2	Se confirma que el JWT genera, refresca y envía el token para ser encriptado, con el algoritmo hash, para finalmente permitir o denegar el acceso.
3	Se confirma que el MSPAG, cumple con el mismo funcionamiento de los elementos cuando fueron implementados de forma parcial, demostrando que el patrón de seguridad cumplió con su objetivo.

5.6 Diseño de Pruebas

Las pruebas que se realizaron a continuación, validan que la adaptación del patrón de seguridad para la arquitectura de microservicios, "MSPAG," cumple con los objetivos planteados en este trabajo de investigación, las cuales se mencionan en la sesión de planteamiento de problema, cabe mencionar que estos casos de prueba se ejecutaron de la siguiente forma: los casos de prueba CP01, CP02 y CP03 comprenden al lenguaje de programación en C#, mientras que los casos de prueba CP04, CP05 y CP06 comprenden al lenguaje de programación en Python, finalmente los casos de prueba CP07, CP08 y CP09 comprenden al lenguaje de programación en JAVA.

5.6.2 Propósito

Detallar el diseño de las pruebas efectuadas, con diferentes casos de prueba de esta investigación para determinar si son admitidos.

5.6.3 Especificación

Durante la elaboración del plan de pruebas se ejecutó un diseño de prueba, en la Tabla 10 se muestra el diseño comprendido en el plan de pruebas. Mismo que fue utilizado en la realización de todas las pruebas, ya que, el objetivo fue probar que el patrón funcionara correctamente de acuerdo al propósito inicial.

Tabla 10 Diseño de prueba MSPAG-DP-01.

MSPAG-DP-01
Nombre: Evaluación de funcionamiento correcto.
Objetivo: Evaluar el funcionamiento adecuado de cada parte de la implementación, así como la implementación en conjunto y la réplica en tres lenguajes de programación distintos.

<table>
<tr><td colspan="2">Características a probar:
La API Gateway, gestiona la solicitud, permitiendo o denegando el acceso.
El JWT, genera, crea y refresca el token, para ser encriptado por el algoritmo de codificación H256, una vez que el usuario ingresa sus datos.</td></tr>
<tr><td>Caso de prueba</td><td>Criterios de aceptación/rechazo</td></tr>
<tr><td>MSPAG-CP-01</td><td rowspan="9">En cada caso de prueba se especificarán los datos precisos que se requieren y los resultados obtenidos.
Un caso de prueba debe considerarse válido cuando los resultados cumplan exitosamente con su propósito.
Para que se pase la prueba, cada elemento debe pasar todos sus casos de prueba.</td></tr>
<tr><td>MSPAG-CP-02</td></tr>
<tr><td>MSPAG-CP-03</td></tr>
<tr><td>MSPAG-CP-04</td></tr>
<tr><td>MSPAG-CP-05</td></tr>
<tr><td>MSPAG-CP-06</td></tr>
<tr><td>MSPAG-CP-07</td></tr>
<tr><td>MSPAG-CP-08</td></tr>
<tr><td>MSPAG-CP-09</td></tr>
<tr><td colspan="2">NOTA: Estos casos de prueba se ejecutan de la siguiente forma: los casos de prueba CP01, CP02 y CP03 comprenden al lenguaje de programación en C#, mientras que los casos de prueba CP04, CP05 y CP06 comprenden al lenguaje de programación en Python, finalmente los casos de prueba CP07, CP08 y CP09 comprenden al lenguaje de programación en JAVA.</td></tr>
</table>

5.7 Ejecución de Pruebas

Una vez determinado el plan de pruebas y concluida la definición de los elementos de seguridad de la adaptación de patrón, “MSPAG,” se llevó a cabo la ejecución de las pruebas con la creación por etapas de la solución mencionada en el plan de pruebas, en la Tabla 11 se muestra el Ambiente de ejecución de pruebas.

Los ambientes utilizados en las pruebas se describen en las Tablas siguientes.

En el Anexo de ejecución de pruebas, se presentan capturas de pantalla, de la ejecución de cada prueba.

Tabla 11 Ambiente de ejecución de pruebas.

<table>
<tr><th colspan="2">Ambiente de ejecución de pruebas</th></tr>
<tr><th>Hardware</th><th>Software</th></tr>
<tr><td>Laptop
AMD Ryzen 3 3250U with Radeon Graphics
CPU 2.60 GHz
12,0 GB (9,88 GB usable)
Escritorio
Sistema operativo Windows 11.
Visual Studio Code Versión: 1.75.
Node.js: 16.14.</td><td rowspan="2">Sistema operativo Windows 11.
Visual Studio Code
Node.js (v8)
Chrome (v96.0.46)
.Net Core 3.1
C# for Visual Studio Code
Ocelot 16.0.1
Microsoft.AspNetCore.Authentication.JwtBearer 3.0.0
Microsoft.EntityFrameworkCore.SqlServer 3.0.0
Microsoft.EntityFrameworkCore.Tools 3.0.0
Microsoft.Extensions.Logging.Debug 3.0.0
Microsoft.VisualStudio.Azure.Containers.Tools.Targets 1.9.5
Microsoft.VisualStudio.Web.CodeGeneration.Design 3.0.0</td></tr>
</table>

5.8 Bitácora de pruebas

En la Tabla 12 se observa la plantilla para la ejecución del plan de pruebas, que sirvió para registrar los resultados obtenidos en cada prueba.

Tabla 12 Plantilla para los resultados de casos de prueba.

Nombre de prueba	**Identificador del caso de prueba.**		
Autor	Nombre del ejecutor del caso de prueba.	**Fecha**	Fecha de elaboración.
Tipo de prueba	Tipo de prueba.		
DP a evaluar	Diseño de pruebas evaluado con el caso de prueba.		
Etapa 1	Nombre del elemento que será probado.		
Objetivo	Objetivo buscado con la correcta ejecución del caso de prueba.		
Descripción			
Nombre de la etapa	**Descripción de prueba**	**Resultado**	
Etapa	Se hace una descripción de la prueba	Resultado obtenido después de ejecutar la instancia de prueba.	
Observaciones	Observaciones sobre la ejecución de la instancia de prueba 1.		

Las pruebas se realizaron en tres etapas: La etapa uno, consistió en probar que la API Gateway, cumpla con su principal propósito de gestionar y redirigir las solicitudes a los microservicios de los clientes. La etapa dos residió en probar que el JWT, genera, refresca y proporciona el token, también que el algoritmo de codificación H256, recibe y encripta el token. La etapa tres corroboró que las anteriores etapas funcionaran de la forma correcta al integrarse la implementación del MSPAG en diferentes lenguajes de programación.

A continuación, en la Tabla 19, se muestra únicamente la ejecución de una prueba, las restantes se pueden consultar en el Anexo de Bitácora de pruebas de este documento.

Ejecución de la prueba MSPAG-DP-01

En la Tabla 13 se observa la plantilla muestra aplicada a cada caso de prueba, el resto de los resultados se presenta en el documento de reporte de resultados.

Tabla 13 Resultados de casos de prueba etapa 1.

Nombre de prueba	MPSAG-CP-01		
Autor	Karen Monica Hernández Guzmán	**Fecha**	23/noviembre/2023
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MPSAG-CP-01		
Etapa 1	API Gateway: La API Gateway se encarga de gestionar el acceso a los microservicios		
Objetivo	• Evaluar el funcionamiento correcto de API Gateway.		
Descripción			
Nombre de la etapa	**Descripción de prueba**	**Resultado**	
Etapa 1	En esta prueba se ingresan las URL de los diferentes microservicios, para probar que la API Gateway, está cumpliendo adecuadamente con su función.	La API Gateway cumple con las características adecuadas gestionando y redirigiendo el acceso a los microservicios	
Observaciones	No se presentó problema.		

5.9 Resultados de las Pruebas

A continuación, se describen cada una de las implementaciones realizadas por etapas utilizadas en las pruebas, cada una con su entorno de ejecución y lenguajes de desarrollo.

5.9.1 Etapas

5.9.1.1. Etapa 1 API GATEWAY

En la Figura 12 se muestran los elementos de API Gateway.

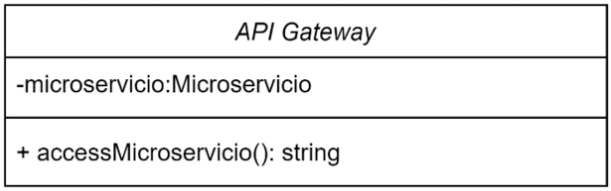


Figura 12 Esquema de API Gateway y Diagrama de clases C# [Java Design Patterns, 2022].

5.9.1.2. Etapa 2 JWT Y H256

En la Figura 13 se muestra el funcionamiento del JWT con el algoritmo hash.

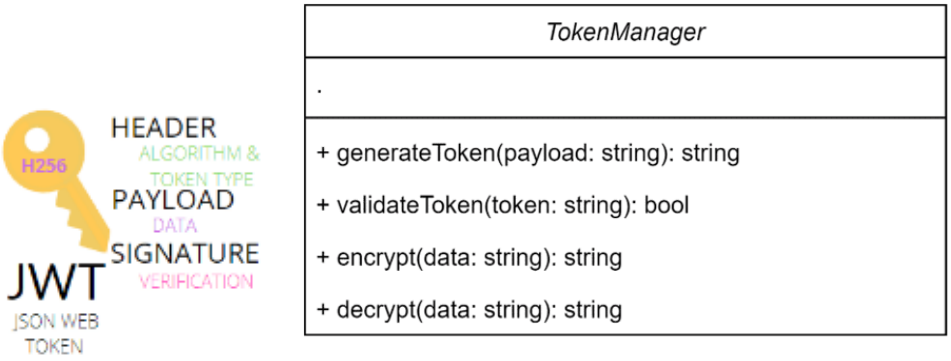


Figura 13 Esquema de JWT-H256 y Diagrama de clases [Java Design Patterns, 2022].

5.9.1.3. Etapa 3 MSPAG

5.9.1.3.1. Lenguaje de programación C#

En la Figura 14 se muestra el funcionamiento del MSPAG.

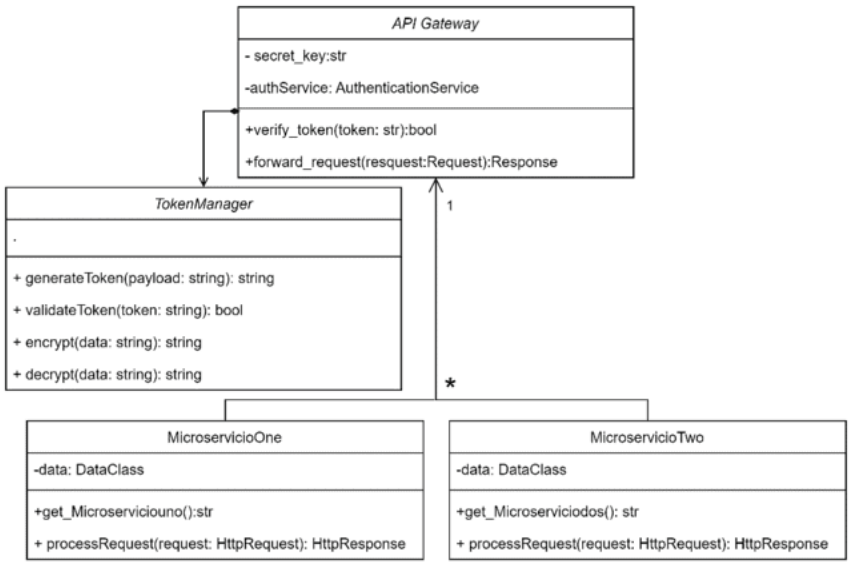


Figura 14 Esquema del MSPAG y Diagrama de clases en C# [Java Design Patterns, 2022].

5.10 Análisis de resultados

Tras la finalización de las pruebas, se procede al análisis de los resultados a la adaptación de la API Gateway con JWT y H256.

El resultado alcanzado en esta investigación se traduce en la adaptación exitosa de un patrón de seguridad en la arquitectura de un microservicio. Los casos de prueba brindan una comprensión más precisa del funcionamiento en cada etapa del patrón, ofreciendo solución al problema planteado. Estos resultados han confirmado que:

5.10.1. Etapa 1

- La API Gateway implementada, cumple con su propósito. Como es gestionar y redirigir las solicitudes planteadas.
- Si el cliente accede directamente con la URL del microservicio prueba esta API Gateway, denegó el acceso al microservicio.
- Si el cliente accede a la API Gateway, solo se muestra que el cliente está accediendo a una interfaz de API Gateway

5.10.2. Etapa 2

- El JWT genera, refresca y comprueba el token.
- Los resultados que se envían al sistema no son los correctos, de no serlos el sistema denegó el acceso al sistema.
- Los resultados que se envían al sistema son los correctos, por los tanto el sistema permite el acceso al sistema.

- El algoritmo HASH recibe el token y lo encripta.

5.14.3. Etapa 3

- El MSPAG se ha implementado exitosamente en otros lenguajes de programación además de C#, como Python y JAVA. Esta demostración valida que el lenguaje de implementación resulta completamente irrelevante para el control de la seguridad. Esto se debe a que el patrón de seguridad actúa como una capa externa que se sitúa por encima de los detalles específicos de la implementación de los microservicios, convirtiéndolo así en un patrón de seguridad versátil y específicamente aplicable a arquitecturas de microservicios.
- El MSPAG protege el acceso al microservicio conforme al objetivo planteado en los lenguajes de programación Python y JAVA.

5.11 Ejecución de pruebas

Documentación de Ejecución de pruebas

En este apartado se presentan los casos de prueba (CP) ejecutados en los escenarios mencionados en el documento del plan de pruebas. Así como la muestra del antes y él después de la implementación en código en los lenguajes de programación C#, PYTHON Y JAVA, cabe mencionar que estos casos de prueba se ejecutan de la siguiente forma: los casos de prueba CP01, CP02 y CP03 comprenden al lenguaje de programación en C#, mientras que los casos de prueba CP04, CP05 y CP06 comprenden al lenguaje de programación en Python, finalmente los casos de prueba CP07, CP08 y CP09 comprenden al lenguaje de programación en JAVA.

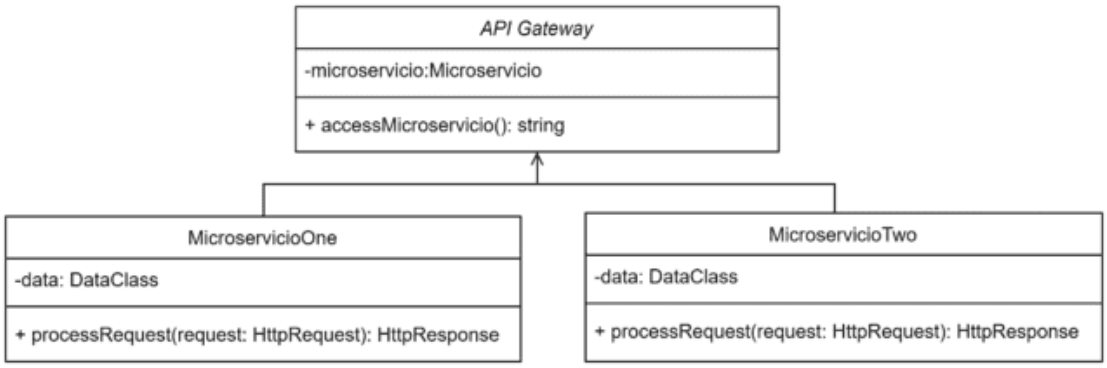
5.11.1 Diagramas de clase del patrón del MSPAG

En la Figura 15 se muestra una serie de diagramas de clases con un antes y un después del patrón de seguridad, con su respectiva implementación en código en los lenguajes de programación C#, PYTHON Y JAVA.

5.11.2 C#

DIAGRAMA DE CLASES IMPLEMENTACIÓN EN C#

API GATEWAY ANTES



MSPAG DESPUÉS

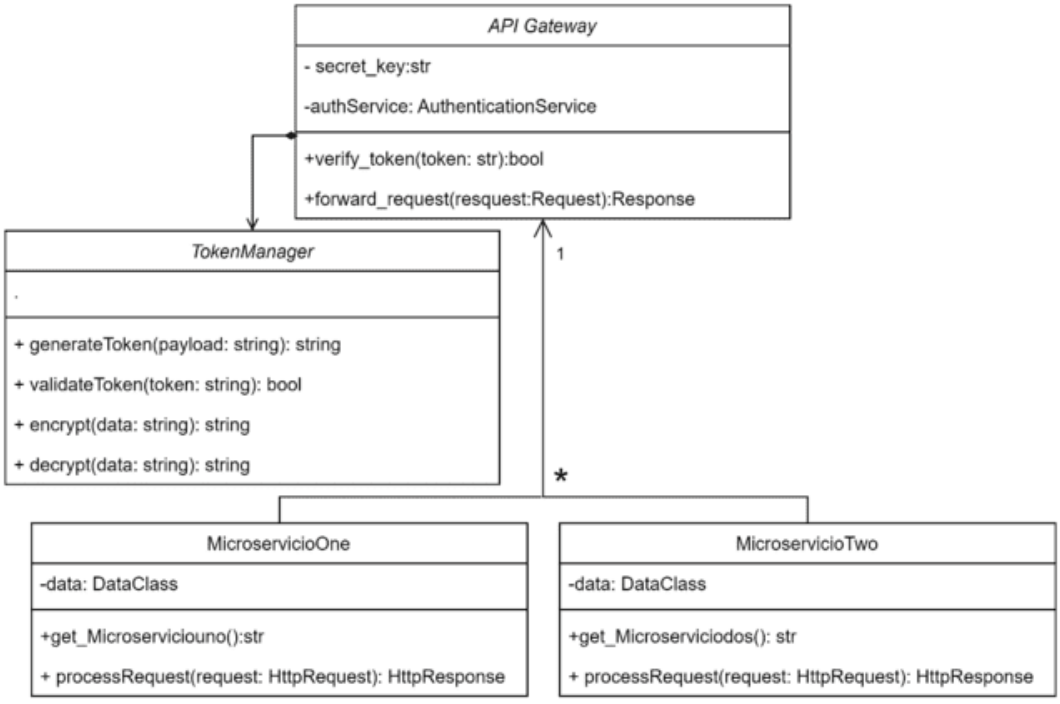


Figura 15 Diagramas de clases implementación en C# [Java Design Patterns, 2022].

5.11.3 CP01. Implementación de API GATEWAY en C#.

ID y Nombre	**CP01. Implementación de API GATEWAY en C#.**	
Datos generales de la prueba		
Fecha		
23 noviembre 2022		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Mónica Hernández Guzmán	Ing. Karen Mónica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		

Objetivo:
- **Evaluar el funcionamiento correcto de API Gateway.**

Prioridad:
Alta

Precondiciones:
Existencia de microservicios funcionando

Postcondiciones:
La API Gateway gestiona y permite el acceso a los microservicios a través del puerto seleccionado para la API Gateway

Valores/Datos de entrada:
1. **URL de acceso:**
   a. API Gateway http://localhost:5000

Resultados esperados:
Al ingresar las diferentes URL se mostrará el acceso a los microservicios a través del puerto de API Gateway

Resultados obtenidos:
La Figura 16 muestra el proceso de solicitud del correcto funcionamiento de API Gateway.

Figura 16 Funcionamiento de API Gateway.

Estado de la prueba:
Aprobada

Acciones correctivas:
Sin ninguna observación

Aprobación:

Ing. Karen Mónica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Comentarios: **La función de API Gateway cumple con la funcionalidad adecuada.**	

5.11.4. CP02. Implementación de Microservicios en C#.

ID y Nombre	**CP02. Implementación de microservicios en C#.**	
Datos generales de la prueba		
Fecha		
23 noviembre 2022		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Mónica Hernández Guzmán	Ing. Karen Mónica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		
Objetivo: • **Evaluar el funcionamiento correcto de la Implementación de microservicios.**		
Prioridad: **Alta**		
Precondiciones: **Existencia de microservicios funcionando.**		
Postcondiciones: **Se accede de forma correcta a los microservicios y se muestra el contenido de estos.**		
Valores/Datos de entrada: **1. URL de acceso:** **a. Microservicio 1 http://localhost:5001/oneservice** **b. Microservicio 2 http://localhost:5002/twoservice**		
Resultados esperados: **Al ingresar las diferentes URL se mostrará el acceso a los microservicios a través de API Gateway y se mostrará el contenido de estos.**		
Resultados obtenidos: **La Figura 17 muestra el correcto funcionamiento de los microservicios y el acceso a través de API Gateway.**		

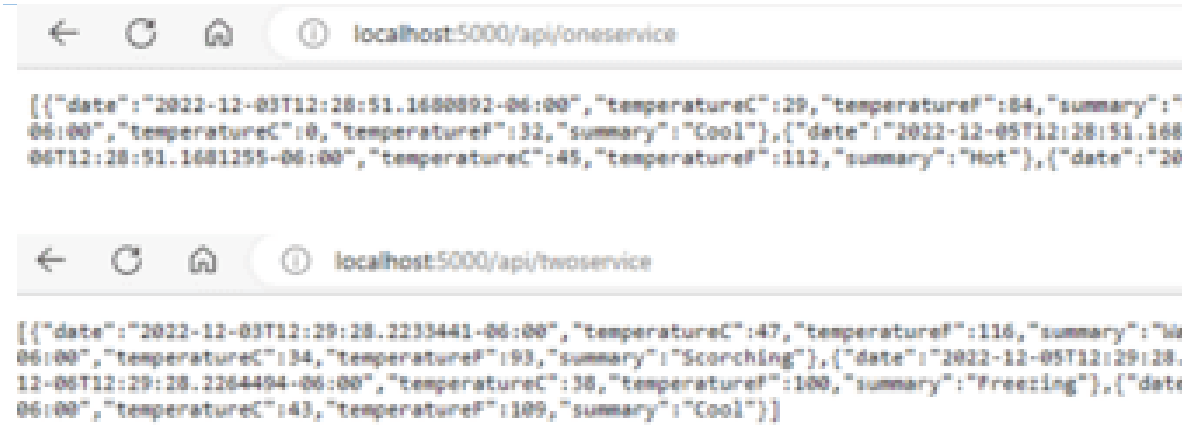


Figura 17 Acceso a microservicio a través de API Gateway.

Estado de la prueba: **Aprobada**	
Acciones correctivas: **Sin ninguna observación**	
Aprobación:	
Ing. Karen Mónica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Comentarios: **La función de API Gateway cumple con la funcionalidad adecuada para el acceso a los microservicios.**	

5.11.5. CP03. Implementación del MSPAG por medio de JWT y H256 en C#.

ID y Nombre	CP03.Implementación del MSPAG por medio de JWT y H256 en C#	
Datos generales de la prueba		
Fecha		
23 noviembre 2022		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Mónica Hernández Guzmán	Ing. Karen Mónica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		
Objetivo: • **Implementación del MSPAG por medio de JWT y H256 de forma efectiva.**		
Prioridad: **Alta**		
Precondiciones: **Existencia de un sistema de microservicios funcionando con un patrón de API Gateway.**		
Postcondiciones: **Al integrar el MSPAG se accede de forma segura, generando un JWT encriptado con H256, para acceder a los microservicios y una vez autenticado el JWT generado, se muestra el contenido de los microservicios.**		
Valores/Datos de entrada: **1. URL de acceso:** **a. Microservicio 1 http://localhost:5001/oneservice** **b. Microservicio 2 http://localhost:5002/twoservice**		
Resultados esperados: **Al ingresar las diferentes URL el sistema se genera un JWT encriptado con H256, el cual se utilizar para autenticar el acceso a los microservicios a través de API Gateway y muestra el contenido de estos.**		
Resultados obtenidos: **La Figura 18 muestra el proceso de generación y autenticación del JWT encriptado con el algoritmo de codificación H256. En la Figura 19 muestra el uso del JWT, en la Figura 20 muestra el acceso a través de API Gateway a cada microservicio de forma segura y en la Figura 21 muestra los datos de los microservicios.**		

```
 * Restarting with stat
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImttaGcifQ.OVXli
rpDwNBa-R98AsZFW96IWCKkBQfiDL345m9mqmc
```

Figura 18 Generación de JWT encriptado con H256.

Figura 19 Uso de JWT.

Figura 20 Acceso permitido.

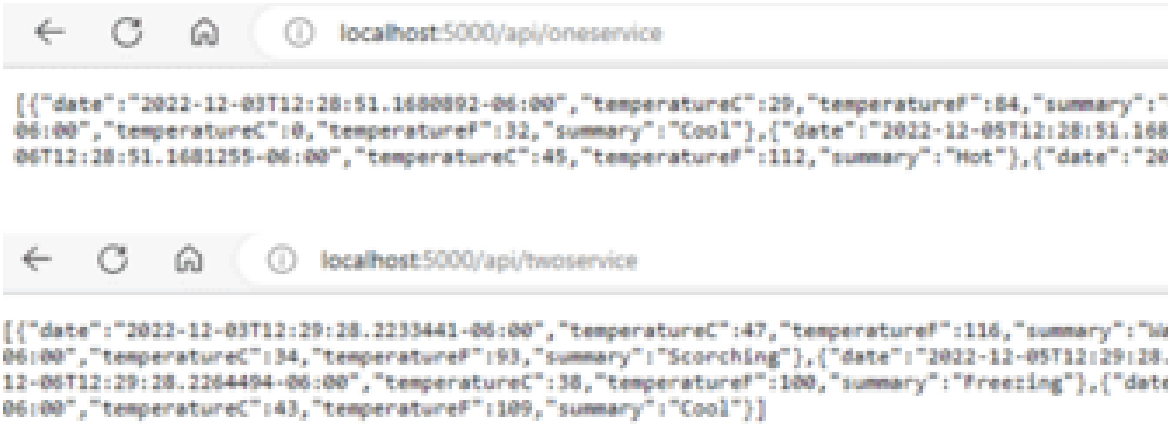


Figura 21 Muestra de datos al acceder a los microservicios.

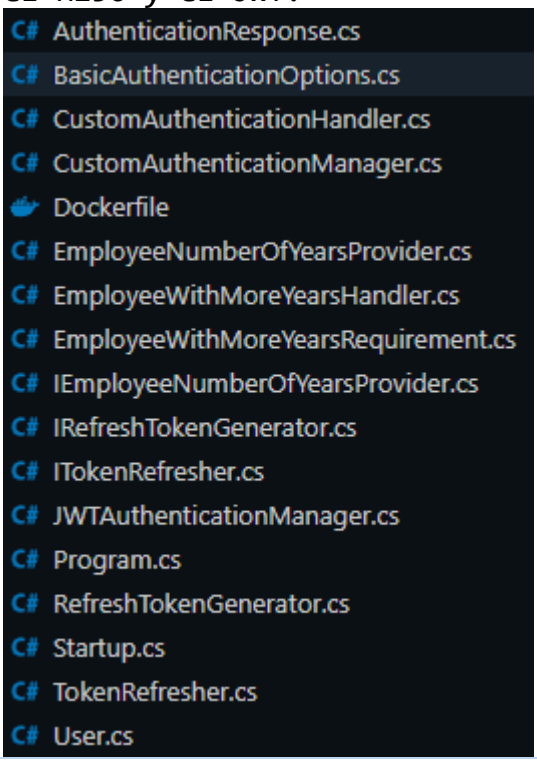
Estado de la prueba: **Aprobada**	
Acciones correctivas: **Sin ninguna observación**	
Aprobación:	
Ing. Karen Mónica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Comentarios: **La función del el MSPAG cumple de forma adecuada para el acceso seguro a los microservicios.**	

5.11.1 Diferencias de Código agregados en API Gateway C#

La implementación de código implica la adición de una serie de clases en la API Gateway, donde se llevan a cabo las configuraciones necesarias para el MPSAG y la garantía del acceso a los microservicios. Es relevante señalar que en este contexto, se exponen exclusivamente las clases que presentan diferencias en la API

Gateway. La configuración específica del JWT y H256 tiene lugar dentro del propio API Gateway, como se muestra en la Tabla 14. Para obtener una comprensión detallada de la implementación, se sugiere revisar los detalles específicos en el anexo A.

Tabla 14 Diferencia de código antes y Después.

API GATEWAY C#	MSPAG C#
La API gate way solo cuenta con una clase de controllers que gestionara el acceso a los microservvicios 	Al implementar el MSPAG se agregan las clases que gestionaran el acceso Seguro con el H256 y el JWT. 

NOTA* La Ejecución de pruebas de los casos de prueba CP04- CP09 se encuentran en el Anexo F.

5.11.6. CP04. Implementación de API GATEWAY PYTHON.

ID y Nombre	**CP04. Implementación de API GATEWAY PYTHON.**	
Datos generales de la prueba		
Fecha		
30 mayo 2023		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Monica Hernández Guzmán	Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		

Objetivo: • **Evaluar el funcionamiento correcto de API Gateway.**
Prioridad: **Alta**
Precondiciones: **Existencia de microservicios funcionando**
Postcondiciones: **La API Gateway gestiona y permite el acceso a los microservicios a través del puerto seleccionado para la API Gateway**
Valores/Datos de entrada: **2. URL de acceso:** a. API Gateway http://localhost:5000
Resultados esperados: **Al ingresar las diferentes URL, se mostrará el acceso a los microservicios a través del puerto de API Gateway.**
Resultados obtenidos: **La Figura 22 muestra el proceso de solicitud correcto funcionamiento de API Gateway.**

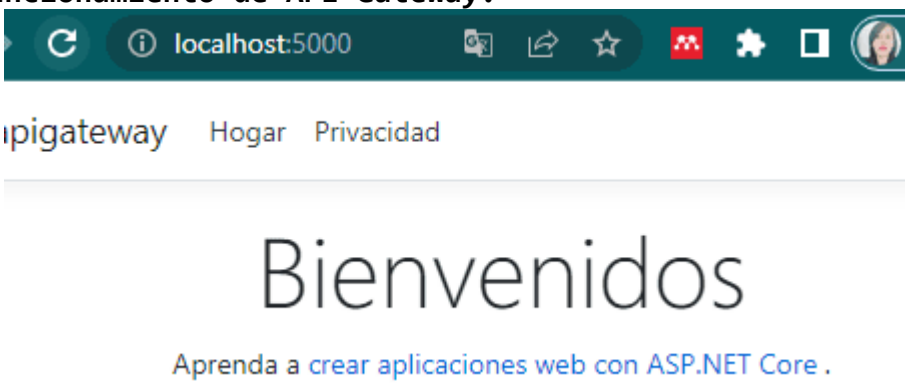


Figura 22 Funcionamiento de API Gateway.

Estado de la prueba: **Aprobada**	
Acciones correctivas: **Sin ninguna observación**	
Aprobación:	
Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Comentarios: **La función de API Gateway cumple con la funcionalidad adecuada.**	

5.11.7. CP05. Implementación de Microservicios Python.

ID y Nombre	**CP05. Implementación de microservicios Python.**	
Datos generales de la prueba		
Fecha		
30 mayo 2023		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Monica Hernández Guzmán	Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		
Objetivo: • **Evaluar el funcionamiento correcto de la Implementación de microservicios.**		
Prioridad: **Alta**		
Precondiciones: **Existencia de microservicios funcionando.**		
Postcondiciones: **Se accede de forma correcta a los microservicios y se muestra el contenido de estos.**		
Valores/Datos de entrada: **2. URL de acceso:** **a. Microservicio 1 http://127.0.0.1:5000/skincare/night** **b. Microservicio 2 http://127.0.0.1:5000/skincare/day**		
Resultados esperados: **Al ingresar las diferentes URL se mostrará el acceso a los microservicios a través de API Gateway y se mostrará el contenido de estos.**		
Resultados obtenidos: **La Figura 23 muestra el proceso de solicitud correcto del funcionamiento de los microservicios y el acceso a través de API Gateway.**		

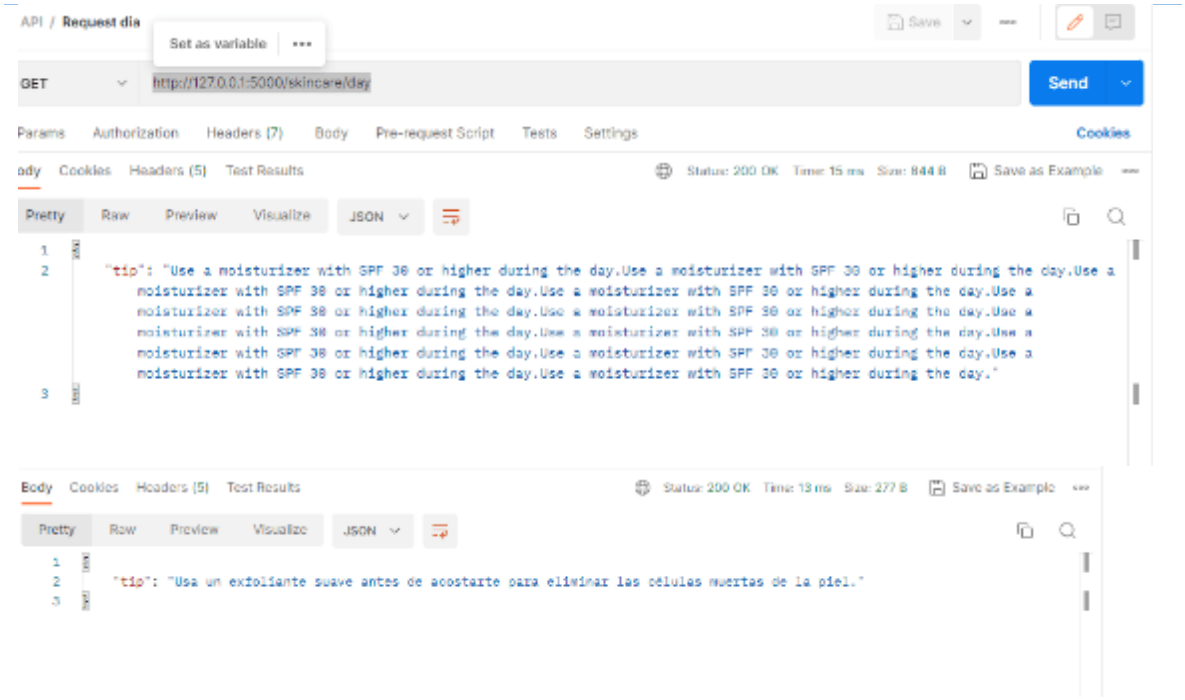


Figura 23 Acceso a microservicio a través de API Gateway.

Estado de la prueba: **Aprobada**	
Acciones correctivas: **Sin ninguna observación**	
Aprobación:	
Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Comentarios: **La función de API Gateway cumple con la funcionalidad adecuada para el acceso a los microservicios.**	

5.11.8. CP06. Implementación del MSPAG por medio de JWT y H256 en Python.

ID y Nombre	**CP06.Implementación del MSPAG por medio de JWT y H256 en Python.**	
Datos generales de la prueba		
Fecha		
30 mayo 2023		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Monica Hernández Guzmán	Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		
Objetivo: • **Implementación del MSPAG por medio de JWT y H256 de forma efectiva.**		
Prioridad: **Alta**		
Precondiciones: **Existencia de un sistema de microservicios funcionando con un patrón de API Gateway.**		
Postcondiciones: **Al integrar el MSPAG se accede de forma segura, generando un JWT encriptado con H256, para acceder a los microservicios y una vez autenticado el JWT generado, se muestra el contenido de los microservicios.**		
Valores/Datos de entrada: **2. URL de acceso:** **a. Microservicio 1 http://127.0.0.1:5000/skincare/night** **b. Microservicio http://127.0.0.1:5000/skincare/day**		
Resultados esperados: **Al ingresar las diferentes URL el sistema se genera un JWT encriptado con H256, el cual se utilizar para autenticar el acceso a los microservicios a través de API Gateway y muestra el contenido de estos.**		
Resultados obtenidos: **La Figura 24 muestra el proceso de generación y autenticación del JWT encriptado con el algoritmo de codificación H256. En la Figura 25 muestra el uso del JWT, y en la Figura 26 permite el acceso a través de API Gateway a cada microservicio de forma segura.**		

```
 * Restarting with stat
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImttaGcifQ.OVXli
rpDwNBa-R98AsZFW96IWCKkBQfiDL345m9mqmc
```

Figura 24 Generación de JWT encriptado con H256.

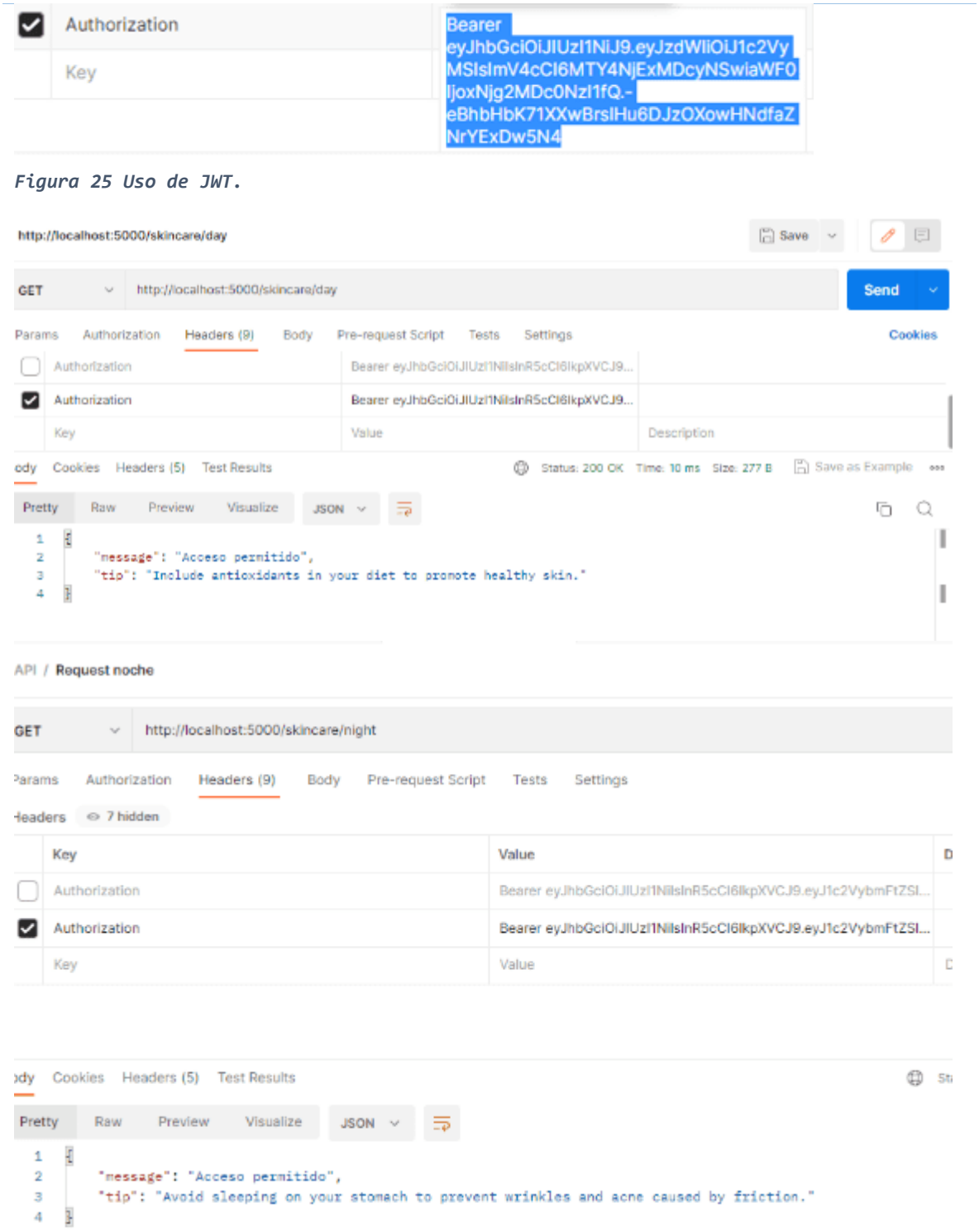


Figura 25 Uso de JWT.

Figura 26 Acceso permitido.

Estado de la prueba:
Aprobada

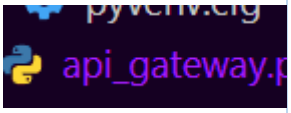
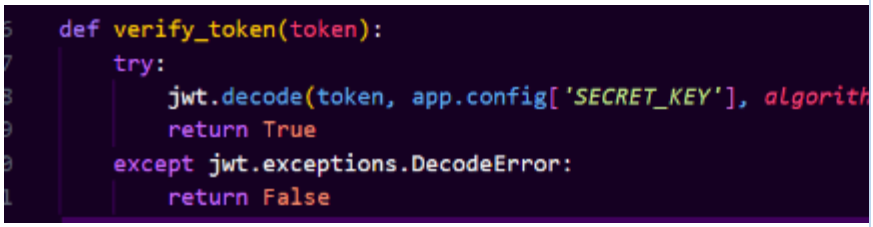
Acciones correctivas:
Sin ninguna observación

Aprobación:	
Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Comentarios: **La función del el MSPAG cumple de forma adecuada para el acceso seguro a los microservicios.**	

5.11.8.1. Diferencias de Código agregados en API Gateway Python

Para la implementación de código se agregan una serie de clases en la API Gateway que es en donde se llevan a cabo las configuraciones para el MPSAG y se asegurará el acceso a los microservicios, como se muestra en la Tabla 15, es importante mencionar que solo se muestra la diferencia de clases en la API gateway ya que la configuración del JWT y H256 es dentro de API Gateway, los detalles de implementación se aprecian en el anexo B.

Tabla 15 Diferencia de código en Python.

API GATEWAY Python	MSPAG Python
La API gate way solo cuenta con una API_gateway.py que gestionara el acceso a los microservvicios 	Al implementar el MSPAG se agregan las clases dentro de la clase API_gateway que gestionaran el acceso Seguro con el H256 y el JWT.  

5.11.9. CP07. Implementación de API GATEWAY JAVA.

ID y Nombre	**CP07. Implementación de API GATEWAY JAVA.**

Datos generales de la prueba		
Fecha		
10 de mayo 2023		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Monica Hernández Guzmán	Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		

Objetivo:

- **Evaluar el funcionamiento correcto de API Gateway.**

Prioridad:

Alta

Precondiciones:

Existencia de microservicios funcionando

Postcondiciones:

La API Gateway gestiona y permite el acceso a los microservicios a través del puerto seleccionado para la API Gateway

Valores/Datos de entrada:

3. URL de acceso:

a. API Gateway http://localhost:9192

Resultados esperados:

Al ingresar las diferentes URL se mostrará el acceso a los microservicios a través del puerto de API Gateway

Resultados obtenidos:

La Figura 27 muestra el correcto funcionamiento de API Gateway.

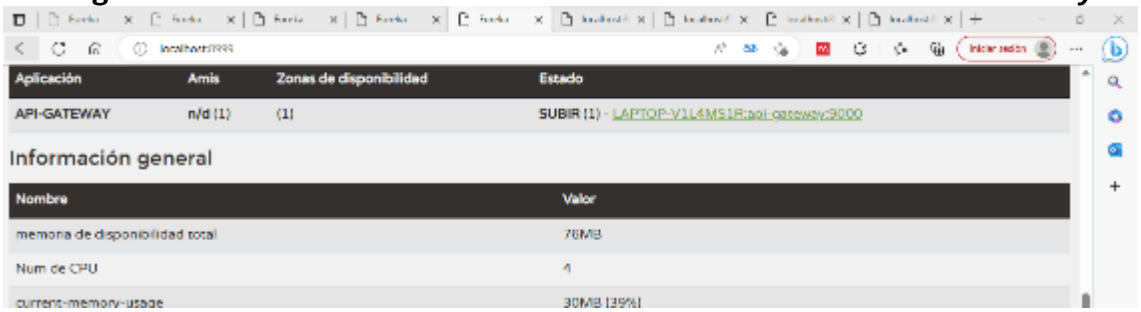


Figura 27 API Gateway funcionando y montada en el servidor de eureka.

Estado de la prueba:

Aprobada

Acciones correctivas:

Sin ninguna observación

Aprobación:

Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.

Comentarios:

La función de API Gateway cumple con la funcionalidad adecuada.

5.11.10. CP08. Implementación de Microservicios en JAVA.

ID y Nombre	**CP08. Implementación de microservicios en JAVA.**	
Datos generales de la prueba		
Fecha		
30 mayo 2023		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Monica Hernández Guzmán	Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		
Objetivo: • **Evaluar el funcionamiento correcto de la Implementación de microservicios.**		
Prioridad: **Alta**		
Precondiciones: **Existencia de microservicios funcionando.**		
Postcondiciones: **Se accede de forma correcta a los microservicios y se muestra el contenido de estos.**		
Valores/Datos de entrada: **3. URL de acceso:** **a. Microservicio 1** http://localhost:9192/service2/say **b. Microservicio 2** http://localhost:9192/service2/say		
Resultados esperados: **Al ingresar las diferentes URL mostrarán el acceso a los microservicios a través de API Gateway y se muestra el contenido de estos.**		
Resultados obtenidos: **La Figura 28 muestra el correcto funcionamiento de los microservicios y el acceso a través de API Gateway.**		

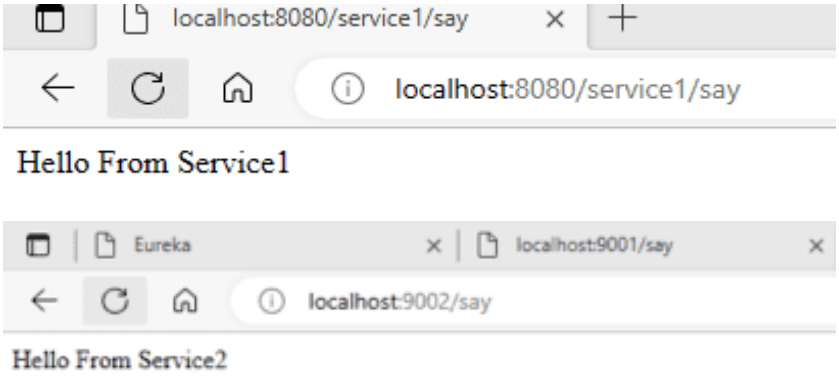


Figura 28 Acceso a microservicio a través de API Gateway

Estado de la prueba: **Aprobada**	
Acciones correctivas: **Sin ninguna observación**	
Aprobación:	
Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Comentarios: **La función de API Gateway cumple con la funcionalidad adecuada para el acceso a los microservicios.**	

5.11.11. CP09. Implementación del MSPAG por medio de JWT y H256 en JAVA.

ID y Nombre	**CP0.Implementación del MSPAG por medio de JWT y H256 en JAVA**	
Datos generales de la prueba		
Fecha		
30 mayo 2023		
Ejecutor	**Evaluador por desarrolladores**	**Evaluador por cliente**
Ing. Karen Monica Hernández Guzmán	Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Desarrollo		
Objetivo: • **Implementación del MSPAG por medio de JWT y H256 de forma efectiva.**		
Prioridad: **Alta**		
Precondiciones:		

Existencia de un sistema de microservicios funcionando con un patrón de API Gateway.

Postcondiciones:
Al integrar el MSPAG se accede de forma segura, generando un JWT encriptado con H256, para acceder a los microservicios y una vez autenticado el JWT generado, se muestra el contenido de los microservicios.

Valores/Datos de entrada:

3. **URL de acceso:**
    a. **Autenticación http:/localhost/9192/authenticate**
    b. **Microservicio 1 http://localhost:9192/ service1/**
    c. **Microservicio 1 http://localhost:9192/ service2/**

Resultados esperados:
Al ingresar las diferentes URL el sistema se genera un JWT encriptado con H256, el cual se utilizar para autenticar el acceso a los microservicios a través de API Gateway y muestra el contenido de estos.

Resultados obtenidos:
La Figura 29 muestra el proceso de generación y autenticación del JWT encriptado con el algoritmo de codificación H256, la Figura 30 muestra el uso del JWT, y en la Figura 31 muestra el acceso a través de API Gateway a cada microservicio de forma segura.

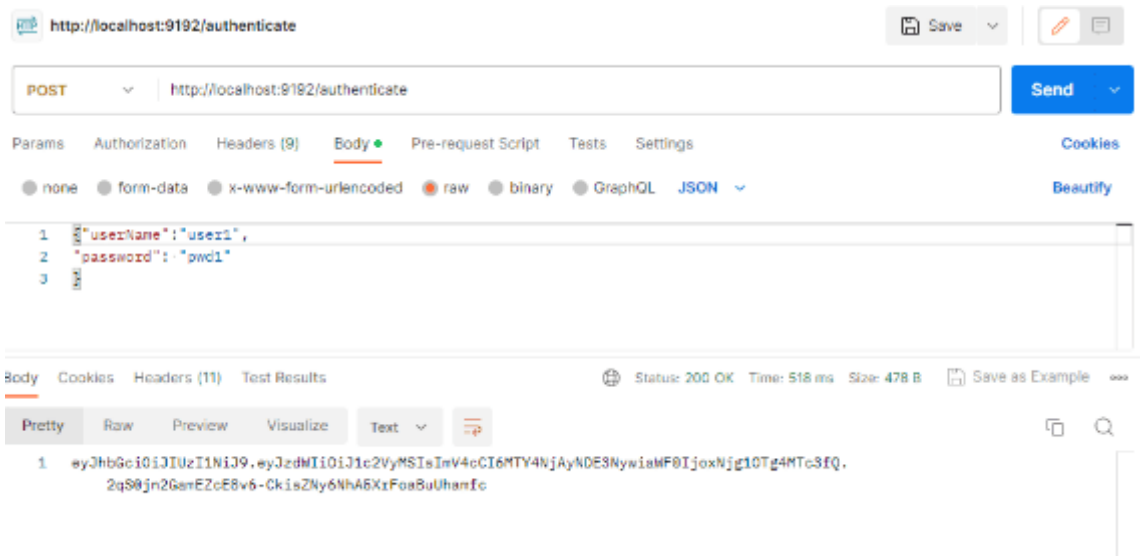


Figura 29 Generación de JWT encriptado con H256.

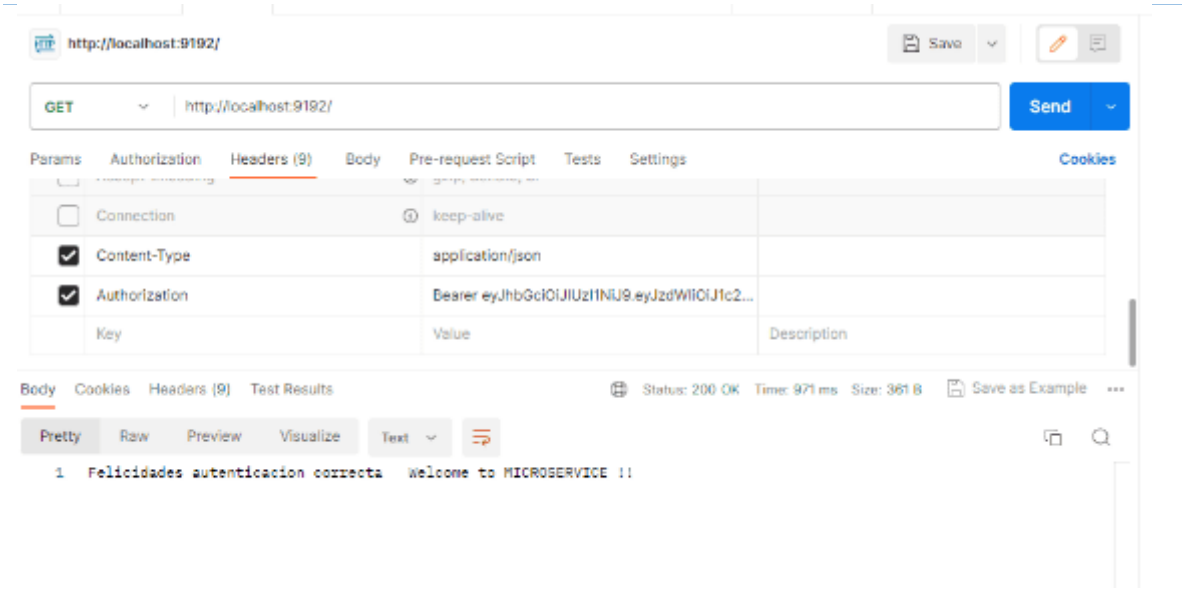


Figura 30 Acceso correcto a microservicios.

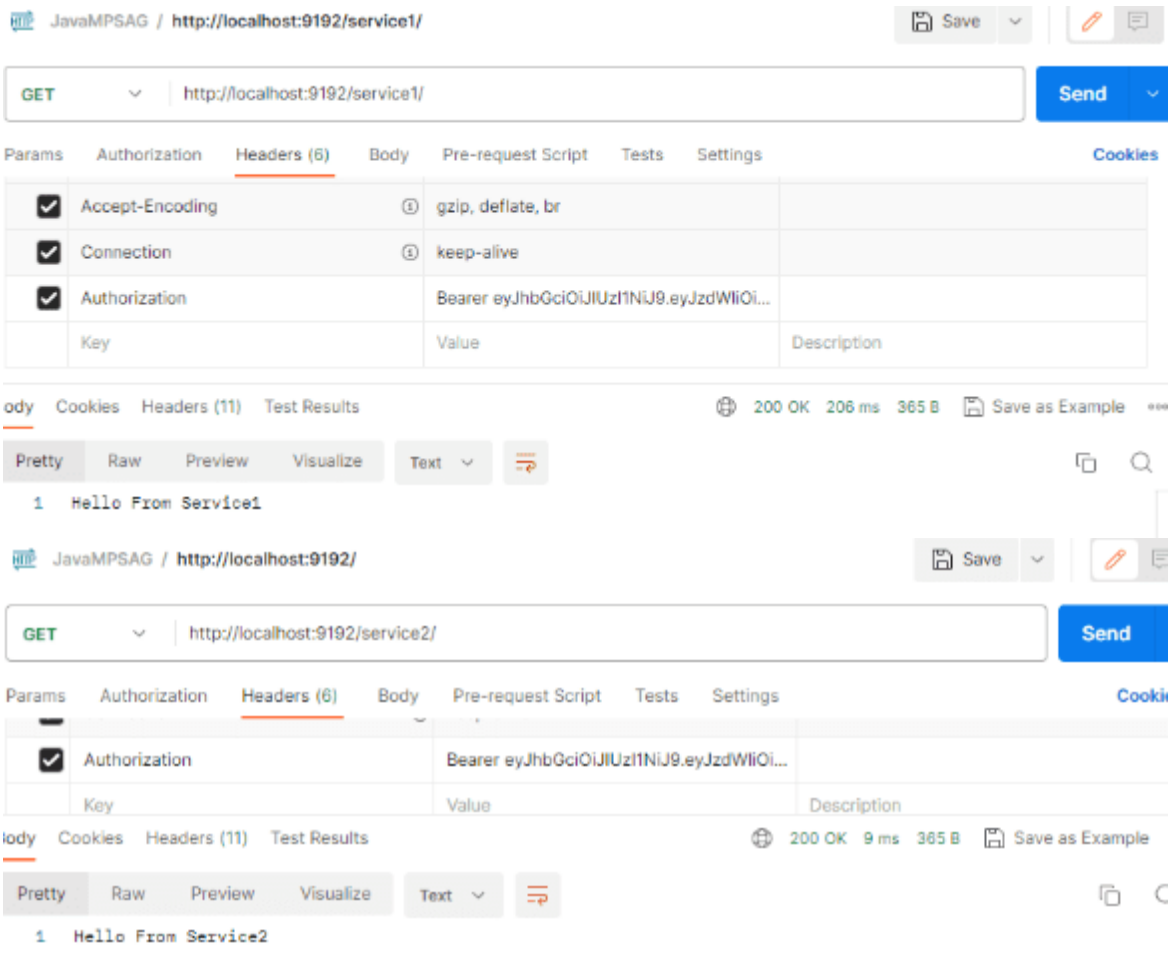
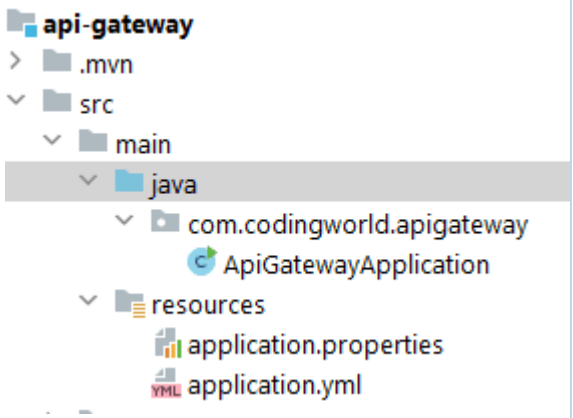
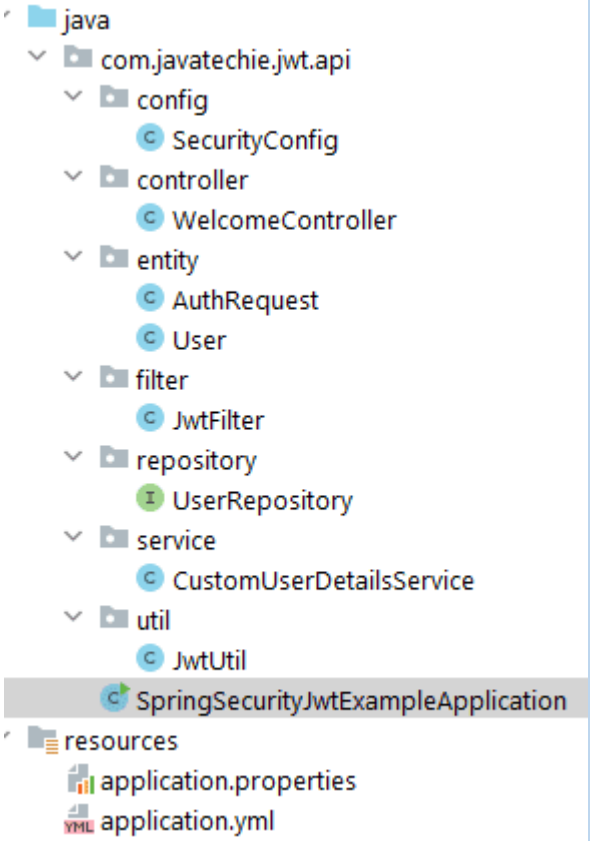


Figura 31 Información de los microservicios.

Estado de la prueba: **Aprobada**	
Acciones correctivas: **Sin ninguna observación**	
Aprobación:	
Ing. Karen Monica Hernández Guzmán	Dr. Juan Carlos Rojas Pérez.
Comentarios: **La función del el MSPAG cumple de forma adecuada para el acceso seguro a los microservicios.**	

5.11.1 Diferencias de Código agregados en API Gateway JAVA

Tabla 16 Diferencia de código en JAVA.

API GATEWAY JAVA	MSPAG JAVA
La API gate way solo cuenta con una clase de controllers que gestionara el acceso a los microservvicios 	Al implementar el MSPAG se agregan las clases dentro de el paquete de API-gateway, que gestionaran el acceso Seguro con el H256 y el JWT. 

CAPÍTULO

06

Conclusiones

En este capítulo se exponen las conclusiones derivadas del desarrollo de la investigación, así como algunas recomendaciones para posibles trabajos futuros.

6.1 Conclusiones

En este trabajo de tesis, se abordó una problemática de seguridad en patrones de microservicios. Se atendieron los objetivos planteados para esta tesis, documentando la adaptación de un patrón de seguridad para arquitecturas de microservicios, en conformidad con la metodología de solución propuesta en esta investigación.

Se abordó la vulnerabilidad existente en los puntos de acceso a los microservicios llegando así a la adaptación del MSPAG que facilita o proporciona una medida de seguridad en los End - Points del API Gateway. Es importante mencionar que, debido a la variedad de lenguajes en los que se implementan los microservicios, los microservicios implementados fueron aquellos que resultaron viables en el transcurso del período de la investigación.
Como resultado, se ha logrado la adaptación del MSPAG y su implementación como patrón de seguridad, el cual se implementó en tres lenguajes de programación, como C#, JAVA y Python, en los que se protege el punto de acceso a los microservicios por medio de la implementación de JWT y algoritmo criptográfico H256 en API Gateway.

Dentro de los principales hallazgos de esta investigación se considera que es fundamental adaptar la implementación de seguridad a los requisitos del proyecto y aprovechar las características y bibliotecas disponibles en cada lenguaje de programación. El patrón MSPAG permite ajustar la implementación según las necesidades específicas, agregando extensiones o bibliotecas adicionales para la seguridad y añadir funcionalidades extras según sea el caso del lenguaje en el que se decida trabajar.

La lógica de implementación utilizando JWT y el algoritmo criptográfico H256 se replicó exitosamente en los tres lenguajes, lo que demuestra la independencia del patrón de seguridad respecto al lenguaje de programación. Las diferencias en los IDEs, como Visual Studio Code e IntelliJ IDEA, también pueden influir en la forma en que se desarrolla y se implementa la seguridad. Esta implementación contribuye de manera significativa a la seguridad de los sistemas basados en microservicios, protegiendo el acceso a los recursos de manera segura. Se logró una implementación exitosa de JWT en los lenguajes de C#, JAVA y Python, con generación y verificación eficiente de tokens.

La verificación de la firma H256 se realizó con éxito, garantizando la integridad de los tokens. Los resultados reflejaron que JWT es un método altamente efectivo para autenticar usuarios y aplicaciones en todos los lenguajes implementados. Se confirmó que la verificación

de la firma HMAC-SHA256 garantizaba la integridad de los tokens, previniendo cualquier manipulación no autorizada.

Las políticas de autorización basadas en roles y permisos definidos en los JWT funcionaron sin problemas en todos los lenguajes. Se destacaron las ventajas de utilizar atributos personalizados en los tokens JWT para controlar el acceso a recursos específicos. Se identificaron posibles vulnerabilidades y amenazas, como la divulgación de claves secretas o la suplantación de identidad. Los resultados indican que la seguridad de los tokens JWT y la integridad de los datos se mantuvieron en todos los lenguajes implementados.

En esta investigación, se cumplieron los alcances que inicialmente se establecieron, enfocándose en las soluciones de seguridad en patrones de diseño en diversas arquitecturas, ya que se emplearon herramientas de código abierto para llevar a cabo las pruebas. En consecuencia, se puede afirmar que se cumplió de manera exitosa esta parte fundamental de la investigación. En este proyecto, tras el desarrollo de los microservicios, la adaptación e implementación del MPSAG y las pruebas satisfactorias, se demuestra que se puede atender la vulnerabilidad existente en los puntos de acceso a microservicios.

6.2 Trabajos futuros

Evaluación en otros lenguajes de programación: Aunque se demostró el funcionamiento del patrón de seguridad MSPAG en Java, C# y Python, sería interesante investigar su aplicabilidad en otros lenguajes de programación más recientes. Esto permitiría ampliar el alcance y la flexibilidad del patrón, brindando opciones adicionales a los desarrolladores que trabajan en diferentes entornos.

Mejora de la integración con IDEs: Dado que los IDEs desempeñan un papel importante en el desarrollo de software, sería beneficioso explorar y evaluar las capacidades de integración del patrón MSPAG con IDEs específicos, como Visual Studio Code e IntelliJ IDEA. Esto podría incluir la creación de extensiones o complementos que simplifiquen y optimicen el proceso de implementación de seguridad en el patrón MSPAG en cada IDE.

Investigación sobre bibliotecas y frameworks adicionales: La implementación de seguridad en el patrón MSPAG puede mejorar mediante la incorporación de bibliotecas y frameworks adicionales que proporcionen funcionalidades de seguridad avanzadas. Sería interesante explorar opciones como bibliotecas de autenticación y autorización, frameworks de seguridad específicos del lenguaje o soluciones de seguridad de terceros que se integren bien con el patrón MSPAG.

Implementación en proyectos reales: sería valioso aplicar el patrón de seguridad MSPAG en proyectos reales para evaluar su viabilidad y eficacia en entornos de producción. Esto permitiría recopilar datos y experiencias prácticas que respalden aún más la aplicabilidad y los beneficios del patrón.

Referencias

[1] A. Hannousse and S. Yahiouche, “Securing microservices and microservice architectures: A systematic mapping study,” *Computer Science Review*, vol. 41. 2021. doi: 10.1016/j.cosrev.2021.100415.

[2] E. Fernandez-Buglioni, *Security Patterns in Practice*, Reimpresa., vol. 1. 2013.

[3] A. Pereira-Vale, G. Marquez, H. Astudillo, and E. B. Fernandez, “Security mechanisms used in microservices-based systems: A systematic mapping,” *Proc. - 2019 45th Lat. Am. Comput. Conf. CLEI 2019*, no. June, 2019, doi: 10.1109/CLEI47609.2019.235060.

[4] A. Barabanov and D. Makrushin, “Authentication and Authorization in Microservice-Based Systems: Survey of Architecture Patterns,” *Vopr. kiberbezopasnosti*, no. 4(38), pp. 32–43, 2020, doi: 10.21681/2311-3456-2020-04-32-43.

[5] X. He and X. Yang, “Authentication and Authorization of End User in Microservice Architecture,” *J. Phys. Conf. Ser.*, vol. 910, no. 1, 2017, doi: 10.1088/1742-6596/910/1/012060.

[6] A. Banati, E. Kail, K. Karoczkai, and M. Kozlovszky, “Authentication and authorization orchestrator for microservice-based software architectures,” *2018 41st Int. Conv. Inf. Commun. Technol. Electron. Microelectron. MIPRO 2018 - Proc.*, pp. 1180–1184, 2018, doi: 10.23919/MIPRO.2018.8400214.

[7] I. D. Joya de la Cruz, “Selección y adaptación de métricas para Microservicios,” CENIDET, 2022.

[8] J. Victoria, P. Sánchez, and M. C. M. G. Rodríguez, “Centro Nacional de Investigación y Desarrollo Tecnológico Departamento de Sistemas Computacionales Maestría en Ciencias de la Computación PROPUESTA DE TESIS ‘ Estudio de Mecanismos , Métricas y Patrones de Seguridad en Microservicios ’ . Dr . Juan Carlos ,” 2021.

[9] Carnegie Mellon University, “Software Architecture | Software Engineering Institute.” https://www.sei.cmu.edu/our-work/software-architecture/ (accessed May 24, 2023).

[10] M. Cristi, “Introduccion a la Arquitectura de Software no en el ciclo de vida del sistema,” no. January 2008, 2014.

[11] Maximiliano Cristia, “Introducción a la Arquitectura de Software”, Accessed: May 24, 2023. [Online]. Available: https://www.researchgate.net/publication/251932352_Introduccion_a_la_Arquitectura_de_Software

[12] C. K. Rudrabhatla, “Security Design Patterns in Distributed Microservice Architecture.” [Online]. Available: https://sites.google.com/site/ijcsis/

[13] J. Q. Ning, “ADE - an architecture design environment for component-based software engineering,” 1997. doi: 10.1145/253228.253500.

[14] L. Bass, P. Clements, and R. Kazman, “Software Architecture in Practice (3rd Edition),” *Architecture*, 2012.

[15] E. Poort, J. Klein, and X. Xu, “Software Architecture in Practice Track,” 2022. doi: 10.1109/ICSA-C54293.2022.00007.

[16] C. P. Jeremy Likness, “Serverless apps: Architecture, patterns, and Azure implementation.” https://learn.microsoft.com/en-us/dotnet/architecture/serverless/ (accessed May 24, 2023).

[17] T. Tenev and S. Tsvetanov, “Recommendations for Enhancing Security in Microservice Environment Altered in an Intelligent Way,” 2020, doi: 10.23919/SoftCOM50211.2020.9238277.

[18] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira, “Security in microservices architectures,” *Procedia Comput. Sci.*, vol. 181, no. 2019, pp. 1225–1236, 2021, doi: 10.1016/j.procs.2021.01.320.

[19] Microsoft Corporation, “Architecture styles - Azure Application Architecture Guide | Microsoft Learn.” https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/ (accessed May 24, 2023).

[20] B. W. M. R. Cesar de la Torre, .“NET Microservices. Architecture for Containerized .NET Applications | Microsoft Learn.” https://learn.microsoft.com/en-us/dotnet/architecture/microservices/ (accessed May 24, 2023).

[21] RAE, “seguridad | Definición | Diccionario de la lengua española | RAE - ASALE.” https://dle.rae.es/seguridad (accessed Oct. 13, 2022).

[22] IEEE, “International Standard International Standard - ISO 527-1,” *IEEE Softw.*, vol. 2012, 2012, doi: 10.1109/IEEESTD.2020.9238526.

[23] C. Richardson, *Microservices patterns : with examples in Java*. NEW YORK, 2017. [Online]. Available: https://learning.oreilly.com/library/view/microservices-patterns/9781617294549/

[24] G. Verdier, Diego Rodriguez, “Implementación de Patrones de Microservicios,” Universidad de la República Montevideo, uruguay, 2020. [Online]. Available: https://www.colibri.udelar.edu.uy/jspui/handle/20.500.12008/25418

[25] W. Hasselbring and G. Steinacker, “Microservice architectures for scalability, agility and reliability in e-commerce,” *Proc. - 2017 IEEE Int. Conf. Softw. Archit. Work. ICSAW 2017 Side Track Proc.*, pp. 243–246, 2017, doi: 10.1109/ICSAW.2017.11.

[26] T. Yarygina and A. H. Bagge, “Overcoming Security Challenges in Microservice Architectures,” *Proc. - 12th IEEE Int. Symp. Serv. Syst. Eng. SOSE 2018 9th Int. Work. Jt. Cloud Comput. JCC 2018*, pp. 11–20, 2018, doi: 10.1109/SOSE.2018.00011.

[27] RAE, “vulnerabilidad | Definición | Diccionario de la lengua española | RAE - ASALE.” https://dle.rae.es/vulnerabilidad (accessed Aug. 28, 2023).

[28] L. Feito, L. Feito Universidad Rey Juan Carlos Madrid, and U. Rey Juan Carlos Madrid, “Vulnerabilidad,” *An. Sist. Sanit. Navar.*, vol. 30, pp. 07–22, 2007, Accessed: Aug. 28, 2023. [Online]. Available: https://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1137-66272007000600002&lng=es&nrm=iso&tlng=en

[29] C. Alexander, S. Ishikawa, and M. Silverstein, “PATTERNS LANGUAGES,” p. 1171, 1977, doi: 10.1186/s40410-017-0073-1.

[30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software (Addison Wesley professional computing series)*. 1994.

[31] Q. Nguyen and O. Baker, “Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API,” *J. Softw.*, pp. 257–264, Jun. 2019, doi: 10.17706/JSW.14.6.257-264.

[32] G. Richard, “Patterns of Software Tales from the Software Community,” *book*, pp. 1–239, 2003, [Online]. Available: papers://61aea551-8d9f-4bd3-b82b-d7c3e55e6fa7/Paper/p4036

[33] Java Design Pattern, “Facade,” 2022. https://java-design-patterns.com/patterns/facade/ (accessed Oct. 15, 2022).

[34] Richardson Chris, “API Gateway | Java Design Patterns,” 2017. https://java-design-patterns.com/patterns/api-gateway/ (accessed Sep. 19, 2023).

[35] T. Americo Leonardo and O. R. Filipe, “API DESIGN-THE MODERN DEVELOPMENT GUIDE Sensedia Content”.

[36] Java Design Patterns, “API Gateway,” 2022. https://java-design-patterns.com/patterns/api-gateway/ (accessed Oct. 15, 2022).

[37] IBM, “API Gateway,” 2022. https://www.ibm.com/docs/en/b2b-integrator/6.1.1?topic=help-api-gateway&mhsrc=ibmsearch_a&mhq=api gateway (accessed Oct. 16, 2022).

[38] Microsoft, “API gateways - Azure Architecture Center | Microsoft Learn,” 2022. https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway (accessed Oct. 16, 2022).

[39] K. A. Torkura, M. I. H. Sukmana, and C. Meinel, “Integrating continuous security assessments in microservices and cloud native applications,” *UCC 2017 - Proc. the10th Int. Conf. Util. Cloud Comput.*, pp. 171–180, 2017, doi: 10.1145/3147213.3147229.

[40] M. Waseem, P. Liang, A. Ahmad, M. Shahin, A. A. Khan, and G. Márquez, “Decision

Models for Selecting Patterns and Strategies in Microservices Systems and their Evaluation by Practitioners; Decision Models for Selecting Patterns and Strategies in Microservices Systems and their Evaluation by Practitioners," 2021, doi: 10.1145/xxxxxxx.xxxxxxx.

[41] C. Esposito, A. Castiglione, C. A. Tudorica, and F. Pop, "Security and Privacy for Cloud-Based Data Management in the Health Network Service Chain: A Microservice Approach," *IEEE Commun. Mag.*, vol. 55, no. 9, pp. 102–108, 2017, doi: 10.1109/MCOM.2017.1700089.

[42] K. A. Torkura, M. I. H. Sukmana, F. Cheng, and C. Meinel, *CAVAS: Neutralizing application and container security vulnerabilities in the cloud native era*, vol. 254. Springer International Publishing, 2018. doi: 10.1007/978-3-030-01701-9_26.

[43] P. Das, R. Laigner, and Y. Zhou, *HawkEDA: A tool for quantifying data integrity violations in event-driven microservices*, vol. 1, no. 1. Association for Computing Machinery, 2021. doi: 10.1145/3465480.3467838.

[44] D. E. 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," Sep. 2001, doi: 10.17487/RFC3174.

[45] F. J. MARTÍNEZ JUAN, "GUÍA DE CONSTRUCCIÓN DE SOFTWARE EN JAVA CON PATRONES DE DISEÑO," ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA EN INFORMÁTICA DE OVIEDO, Oviedo, 2018. Accessed: Oct. 06, 2022. [Online]. Available: https://drive.google.com/drive/my-drive

[46] A. Maña, D. Ray, F. Sánchez, and M. I. Yagüe, "Integrando la Ingeniería de Seguridad en un Proceso de Ingeniería Software *," 2022.

[47] F. José, G. Peñalvo, and C. Pardo Aguilar, "Diagramas de Clase en UML 1.1".

[48] C. Pardo and F. J. Garcia, "Diagrama de Clase en UML," pp. 1–8, 2017, [Online]. Available: https://repositorio.grial.eu/bitstream/grial/353/1/DClase.pdf

[49] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML class diagrams," *Artif. Intell.*, vol. 168, no. 1–2, pp. 70–118, Oct. 2005, doi: 10.1016/J.ARTINT.2005.05.003.

[50] M. O. Pahl and L. Donini, "Giving IoT services an identity and changeable attributes," *2019 IFIP/IEEE Symp. Integr. Netw. Serv. Manag. IM 2019*, no. section II, pp. 455–461, 2019.

[51] K. Lapagna, M. Zollinger, M. R.-… 2018, undefined Nice, undefined France, and undefined 2018, "Dokspot: securely linking healthcare products with online instructions," *Digitalcollection.Zhaw.Ch*, no. c, pp. 55–64, 2018, doi: 10.21256/zhaw-5000.

[52] T. Tenev and D. Birov, "SECURITY PATTERNS FOR MICROSERVICES LOCATED ON DIFFERENT VENDORS," *J. Tech. Univ. - Sofia Plovdiv branch, Bulg. "Fundamental Sci.*

Appl., vol. 24, p. 4, 2018, [Online]. Available: https://scholar.google.com/scholar?hl=pt-BR&as_sdt=0%2C5&q=SECURITY+PATTERNS+FOR+MICROSERVICES+LOCATED+ON+DIFFERENT+VENDORS&btnG=

[53] P. Nkomo and M. Coetzee, "Software Development Activities for Secure Microservices," in *Computational Science and Its Applications -- ICCSA 2019*, 2019, pp. 573–585.

[54] N. Chondamrongkul, J. Sun, and I. Warren, "Automated Security Analysis for Microservice Architecture," *Proc. - 2020 IEEE Int. Conf. Softw. Archit. Companion, ICSA-C 2020*, pp. 79–82, 2020, doi: 10.1109/ICSA-C50368.2020.00024.

[55] D. Yu, Y. Jin, Y. Zhang, and X. Zheng, "A survey on security issues in services communication of Microservices-enabled fog applications," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 22. 2019. doi: 10.1002/cpe.4436.

[56] S. Amir-Mohammadian and A. Y. Zowj, "Towards Concurrent Audit Logging in Microservices", doi: 10.1109/COMPSAC51774.2021.00191.

[57] K. A. Torkura, M. I. H. Sukmana, F. Cheng, and C. Meinel, "Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications," *Proc. - 2nd IEEE Int. Conf. Smart Cloud, SmartCloud 2017*, pp. 90–97, Nov. 2017, doi: 10.1109/SMARTCLOUD.2017.21.

[58] M. Stocker, O. Zimmermann, U. Zdun, D. Lübke, and C. Pautasso, "Interface quality patterns - Communicating and improving the quality of microservices APIs," *ACM Int. Conf. Proceeding Ser.*, 2018, doi: 10.1145/3282308.3282319.

[59] J. Bogner, A. Zimmermann, and S. Wagner, "Analyzing the Relevance of SOA Patterns for Microservice-Based Systems", Accessed: Sep. 09, 2022. [Online]. Available: http://ceur-ws.org/Vol-2072

[60] E. Shmeleva, "How Microservices are Changing the Security Landscape," 2020.

[61] G. Márquez and H. Astudillo, "Identifying Availability Tactics to Support Security Architectural Design of Microservice-based Systems", doi: 10.1145/3344948.3344996.

[62] S. Jacob, Y. Qiao, and B. Lee, "Detecting Cyber Security Attacks against a Microservices Application using Distributed Tracing", doi: 10.5220/0010308905880595.

[63] M. Summerfield, "Programming in Python 3," *Text*, p. 644, 2010, Accessed: Apr. 18, 2023. [Online]. Available: papers3://publication/uuid/90BA8865-B2D4-462E-A717-3F71636BBC6C

[64] ORACLE, "What is python?," 2022. https://developer.oracle.com/es/learn/technical-articles/what-is-python (accessed Apr. 18, 2023).

[65] P. S. Foundation, “Flask · PyPI,” 2023. https://pypi.org/project/Flask/ (accessed Apr. 19, 2023).

[66] Python, “venv — Creation of virtual environments — Python 3.11.3 documentation.” https://docs.python.org/3/library/venv.html (accessed May 03, 2023).

[67] “Welcome to Python.org.” https://www.python.org/ (accessed May 03, 2023).

[68] “Definición de API REST: ¿Qué son las API REST (API RESTful)?” https://www.astera.com/es/tipo/blog/definición-de-la-API-de-descanso/ (accessed Sep. 27, 2022).

[69] R. Padmanaban, M. Thirumaran, P. Anitha, and A. Moshika, “Computability evaluation of RESTful API using Primitive Recursive Function,” *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 2, pp. 457–467, Feb. 2022, doi: 10.1016/J.JKSUCI.2018.11.014.

[70] C. E. de Oliveira, G. L. Turnquist, and A. Antonov, “Developing Java Applications with Spring and Spring Boot: Practical Spring and Spring Boot solutions for building effective applications,” 2018, Accessed: Apr. 18, 2023. [Online]. Available: https://books.google.cz/books?id=01NxDwAAQBAJ

[71] Microsoft, “Un paseo por C#: información general | Microsoft Learn,” 2023. https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/ (accessed Apr. 18, 2023).

[72] Microsoft, “Why Visual Studio Code?,” 2023. https://code.visualstudio.com/Docs/editor/whyvscode (accessed Apr. 19, 2023).

[73] G. P. Insights, “What is Postman? Postman API Platform,” 2022. https://www.postman.com/product/what-is-postman/ (accessed Apr. 19, 2023).

[74] M. R. Cesar de la Torre, Bill Wagner, .“NET Microservices: Architecture for Containerized .NET Applications.” https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot (accessed Apr. 19, 2023).

[75] Microsoft, “Visual Studio Code - Code Editing. Redefined.” https://code.visualstudio.com/ (accessed May 03, 2023).

[76] Microsoft, “C# | Lenguaje de programación moderno y de código abierto para .NET.” https://dotnet.microsoft.com/es-es/languages/csharp (accessed May 03, 2023).

Anexo A – Trabajos Relacionados

Resúmenes de trabajos relacionados

A.1. Authentication and authorization in microservice-based systems: survey of architecture patterns

Barabanov y Makrushin realizan un análisis de las principales problemáticas de seguridad en microservicios, y parten de tres preguntas de investigación que fungen como punto clave para desarrollar su investigación, como patrones, ventajas o desventajas, y consideraciones al implementar una arquitectura de microservicios [4].

Tras esta investigación analizan los patrones encontrados como: Centralized pattern with single PDP high-level architecture, Centralized pattern with embedded policy decision point, en los que se deben considerar las políticas de acceso, autorizaciones de nivel, reglas de control específicas, actualizaciones, autorizaciones y servicios de autentificación (EAS) a través de tokens [4].

Las recomendaciones propuestas en este artículo son: el uso de patrones descentralizados, Centralized pattern with single policy decision point, Centralized pattern with embedded policy decision point, con aplicaciones peculiares para su implementación. Cabe mencionar que estas aplicaciones llevadas a microservicios conllevan una serie de retos en específico para la protección de los sistemas de autenticación y autorización de microservicios por lo que se debería considerar la innovación de los métodos de monitoreo y autorización [4].

A.2 Authentication and authorization of end user in microservice architecture

Xiuyu y Xudong hacen una comparativa de las técnicas tradicionales de seguridad, y proponen estrategias de autenticación, para arquitecturas de microservicios, considerando que un sistema seguro proporciona identificaciones seguras, para las cuales se sugiere el uso de base de datos compartidas [5].

Proponen algunas estrategias para contribuir a la seguridad de un microservicio [5]. Como los servicios de autentificación independiente, para mejorar la escalabilidad del sistema, uso de servicios SSO para validar tokens que cuentan con un inicio único de sesión, con un conjunto de credenciales de acceso, que validan y obtiene los datos del usuario, tokens JavaScript Object Notation (JSON) que son protocolos de transferencia de intercambio de información, JavaScript Object Notation Web Tokens+ Application Programming Interfaces (JWT+API) que envían el almacenamiento de sesiones, verificar los permisos de usuario, se puede implementar

para cierres de sesión y evita ataques cross-origin resource sharing (CORS) y Cross-Site Request Forgery(CSRF) [5].

Este trabajo explora varias soluciones de autenticación y autorización en la arquitectura de microservicios. La solución para la sesión distribuida es similar a la forma tradicional de autenticación, pero es compleja de implementar y mantener. El servidor SSO causará SPOF y problemas de tráfico. El uso de la pasarela API para mejorar la solución JWT del lado del cliente puede resolver el problema del cierre de sesión. Así que no hay una solución correcta absoluta, sino sólo opciones adecuadas para su sistema [5].

A.3 Authentication and authorization orchestrator for microservice-based software architectures

En este trabajo se implementa un orquestador para el manejo de tokens para la autenticación y autorización en microservicios, a través de una API cliente en java, para un sistema de diagnósticos médicos [6].

El modelo de autenticación se emplea en un marco de AAA, tokens, contraseñas y certificados. Las contraseñas parten de requerimientos como usuarios y contraseñas de sistema, que al ser un elemento empleado desde tiempos remotos se vuelve vulnerable para sistemas modernos. Para los que se emplea en mecanismos Transport Layer Security pre-shared key ciphersuites (TLS – PSK) que verifican el funcionamiento de las entidades, a partir de Json Web Token (JWT) componente de llamado de petición como Policy Match Gate, Policy Control Module y Policy Validation module, Autorización [6].

Estos incluyen distintos modelos de control de acceso de atributos, obligatorio (MAC), discrecional (DAC) y basado en roles (RBAC), control de acceso basado en atributos (ABAC), lenguaje de marcado de aserción de seguridad basado en XML (SAML), OAuth que permite acceso por medio de propietario u orquestación de recurso [6].

Para ello se utiliza, el OAUTH2, SSO, JWT JSON Web Token (JWT), y OpenID, implementados en un orquestador y una API en módulos de autenticación y autorización, que pudiera funcionar en microservicios [6].

A.4 Implementación de Patrones de Microservicios

Vernier y Rodríguez [24], proponen una implementación guiada por patrones de arquitectura para microservicios, donde se analizan las funcionalidades de los patrones actuales, para mitigar los distintos problemas que se presentan al hacer uso de una arquitectura de esta índole, además muestran las ventajas de hacer uso de las arquitecturas y las desventajas de las misma como son: la complejidad

de descomposición en microservicios, consistencia, observación y seguridad, ya que exponen que al coexistir distintos microservicios se deben aplicar políticas de seguridad que mantengan y prevalezcan la seguridad de cada uno de ellos lo cual es altamente complejo ya que no solo se debe centrar en una área en específica, si no que se centra en cada elemento que integra este microservicio [24].

El uso de patrones aporta soluciones puntuales a las distintas problemáticas que pudieran surgir, es importante destacar que cada patrón cuenta con características y relaciones propias las cuales son predecesoras, sucesoras, alternativas y complementarias donde se analizan las funcionalidades de los patrones actuales [24]. La propuesta consta de una asistencia para la toma de decisiones desde antes de que comience su desarrollo hasta el término del microservicio, partiendo de una plataforma enfocada a roles con tokens de acceso, como los JSON Web Tokens JWT, a partir de tres strings: header,payload y signature, con el uso de las Herramientas ReactJs con CSS en HTML, para el backend se utilizó el NodeJS y mongoDB, Con patrones circuit Breaker en Java y SAGA [24].

A.5 Overcoming Security Challenges in Microservice Architectures

Yarygina describe un marco de seguridad orientado a microservicios, que deba cumplir con características específicas como son escalabilidad y facilidad de automatización [26].

Además, se menciona la implementación y los motivos por los que en distintas ocasiones se hace referencia de microservicios como SOA ya que cuentan con muchas similitudes [26].

Propone unikernerls que proporcionan alta seguridad en los microservicios y expone los problemas de seguridad en seis etapas como son: Hardware, virtualización, nube, comunicación, servicio/aplicación y orquestación [26].

También propone el uso de Dockers Intel Software Guard Extensions (SGX) ya que proporcionan seguridad a nivel de sistema operativo y consideran los Docker Swarm, el Mutual Transport Layer Security (MTLS) y Public Key Infrastructure (PKI) con certificados de autoidentificación y TLS para prevalecer la seguridad en los microservicios propuestas como son el Domain Driven Design, APIS WEB, implementación con dockers, Denial of Service, Circuit breaker, Security tokens, OpenID, single sign-on, Security Token Service (STS), WS-Trust standard, WS-Security standard, validating security tokens, JSON Web Signature (JWS), Encryption (JWE), Token (JWT), OAuth 2.0 y uso de Netflix MTLS, con el propósito de mantener segura la comunicación entre microservicios, esto partiendo de un tipo de arquitectura SOA [26].

A.7 Giving IoT Services an Identity and Changeable Attributes

Pahl propone un modelo de arquitectura enfocado a seguridad directamente en microservicios, desde el desarrollo hasta la implementación [50].

Con ayuda de certificados para proveer la seguridad en los metadatos y en la ejecución del servicio. Estos certificados son con implementación que brinda protección, a través de firmas de verificación, en distintos niveles del ciclo de vida [50].

En primera instancia se hace un registro con claves públicas. En segunda instancia se hace un ejecutable executableHash, en el que se añade un certificado X.509v3 firmado con la clave privada y se envía para comprobar lógicamente la integridad del certificado, por último, se instala el servicio, y se modifican los valores del mismo, como las políticas de acceso [50].

Una vez que se generan los certificados se crea un control de acceso con ciclos de vida cortos y automatizados. Para verificar la funcionalidad de los certificados se ejecutan los microservicios y se evalúa el tráfico en un rango de tiempo específico. Posteriormente se mitiga la periodicidad no deseada y se renuevan los certificados con intervalos aleatorios y efectos backoff [50].

A.9 Security in Microservices Architectures

Mateus coincide con muchos autores al sugerir que una arquitectura de microservicios es muy similar a una arquitectura SOA y proponen una serie de soluciones para las problemáticas de seguridad encontradas en los microservicios [18].

Además, puntualiza los distintos elementos donde la seguridad se considera endeble, por los ataques exhaustivos [18].

Los riesgos más notables son el uso excesivo de privilegios, ya que, si se desea hackear un microservicio en particular, por la falta de abstracción, se consigue fácilmente debilitando ese segmento en particular [18].

En el aspecto de la seguridad de contraseña se emplea Active Directory, OpenID, o Lightweight Directory Access Protocol (LDAP), en los que evitan características específicas comunes. Aunado a esto se implementa la autenticación con intentos reducidos [18].

Para incrementar la seguridad el autor sugiere Open Group Architecture Framework (TOGAF), Information Technology Infrastructure Library (ITIL), Control Objectives for Information and related Technology (COBIT), o Building Security in Maturity Model (BSIMM) los cuales son marcos específicos. También propone el uso de

Multiprotocol Label Switching (MPSL), cortafuegos de carácter local y perimetral [18].

A.10 Security and privacy for cloud-based data management in the health network service chain: a microservice approach

Esposito aborda los diferentes problemas observados en el desarrollo de microservicios, en una cadena de servicios en una red sanitaria, como son la seguridad y privacidad. Donde aborda la gestión de manejo, intercambio de datos y acoplamiento en la nube para servicios de asistencia sanitaria [41].

En este interactúan distintos elementos que comparten información sensible del paciente, como datos personales, tratamientos, entre otros, los cuales deben ser compartidos en distintas áreas para su seguimiento. Estas áreas se dividen dentro de la cadena como el sistema de registro médico, sistema de historial clínico, que se modifican por los médicos patólogos o enfermeros, y los datos que se entregan al paciente [41].

Al hacer una actualización de información cuando un paciente es atendido, la información se distribuye en los departamentos preliminares para su atención, por lo que pasa por distintas áreas, en el caso de requerir un traslado, la información debe llegar íntegra [41]. Por lo que se exponen las problemáticas dentro de este microservicio y proponen la implementación de una orquestación de APIS, con un patrón modelo, vista, controlador (MVC), en distintas áreas de aplicación con el uso de estándares de Servicio de Distribución de Datos(DDS) de Object Management Group (OMG), además de ICT en HDMI, API gateway, XML JSON REST, cloud service providers (CSPs) [41].

Para el resguardo de la información en su distribución, un vez implementado este patrón se puede mitigar la modificación de los datos, pero la implementación de este patrón y de los estándares no garantiza la seguridad total de la información [41].

A.11 Dokspot – Securely Linking Healthcare Products with Online Instructions

Dokspot es un servicio de gestión de instrucciones para personal médico que realiza diagnósticos del cuerpo humano. Estas instrucciones se manejan de forma virtual por medio de microservicios. Donde se insta que la integridad del archivo no sea corrompida ya que el acceso es de carácter público sin embargo el contenido es vital para que los usuarios efectúen una actividad segura y correspondiente a las necesidades del paciente [51].

Lapagna realiza una serie de observaciones donde la seguridad es tan débil, en una cadena de dokspot en la vinculación de servicios, donde

enmarca las distintas problemáticas en un microservicio, como son las seguridad, disposición e integridad. Y enlista diversos puntos de ataque en la cual describe las posibles fallas, como la interacción directa con el servidor [51].

Para mitigar estas problemáticas propone el uso de protocolos de comunicación segura TLS, control de acceso basado en roles, funciones criptográficas, DNS, CloudFlare, cifrado de extremo a extremo, protocolos de contraseña remota segura [51].

A.12 CAVAS: Neutralizing Application and Container Security vulnerabilities in the Cloud Native Era

Torkura expone los diferentes problemas de seguridad que predominan en la computación en la nube. Los cuales incluyen vulnerabilidad en las imágenes, personalización de contenedores, actualizaciones no automáticas, imágenes de fuentes no confiables, accesos internos, evaluaciones de seguridad, y pruebas de seguridad en REST [42].

Propone formas distintas de mitigar los problemas de seguridad como sustituir los API Gateway por contenedores, estándares de aplicaciones Web Application Security Consortium (WASC) Common Weakness Enumeration (CWE), métodos de autentificación automatizados y monitores de vulnerabilidad [42].

Aunque estas propuestas contribuyen a mitigar las debilidades de seguridad en los microservicios, es importante destacar que no eliminan completamente dichas vulnerabilidades de manera permanente [42].

A.14 HawkEDA: A Tool for Quantifying Data Integrity Violations in Event-driven Microservices

El propósito de HawkEDA es reflejar las características de una arquitectura de microservicio, basada en eventos, que ofrezca flexibilidad y adaptación a flujos, permitir la especificación de invariantes y restricciones además de prevenirlas, la detección de violaciones de integridad del microservicio dado que un microservicio puede ser políglota, y su función es desacoplada su interacción puede producirse a través de protocolos basados en HTTP con módulos independientes que promueven el aislamiento de fallos [43].

El aislamiento de funciones es un aspecto fundamental en el funcionamiento de los microservicios, ya que, al operar de manera independiente, las fallas en un microservicio no afectan a los demás. La capacidad de identificar una falla en un microservicio de manera individual es una ventaja inherente al aislamiento [43]. Una vez que se ha identificado una falla, se pueden explorar las posibles

soluciones a través de la supervisión continua de la integridad de los datos [43].

A.15 Security Patterns for Microservices Located on Different Vendors

En el contexto del trabajo de TENEV [52], se abordan diversas problemáticas inherentes a la seguridad en sistemas de microservicios. Estas problemáticas emergen debido a que los microservicios se basan en patrones de diseño y se implementan en segmentos específicos con restricciones y soluciones de seguridad predefinidas. Los patrones de seguridad establecen las bases para garantizar la seguridad en elementos específicos de los microservicios.

El estudio se enfoca en un análisis exhaustivo de la vulnerabilidad de seguridad utilizando el enfoque de modelo de amenazas STRIDE. Este modelo aborda amenazas clave como la suplantación de identidad, manipulación, repudio, divulgación de información y denegación de servicio. Para cada una de estas amenazas, se propone un patrón de seguridad que aborda una parte del modelo mencionado [52].

Entre los patrones de seguridad propuestos se encuentran: Amazon AWSMicrosoft Azure, Google App Engine 3rd Party Communication, AGENCY GUARD, AGENT AUTHENTICATOR, Application Firewall Cloud Access Security Broker Integration Reverse Proxy [52], Estos patrones ofrecen enfoques específicos para contrarrestar las amenazas mencionadas y proporcionar una mayor seguridad en entornos de microservicios ubicados en diferentes proveedores.

A.16 Software Development Activities for Secure Microservices

El trabajo de Nkomo y Coetzee, aborda un análisis detallado de los procedimientos de desarrollo de arquitecturas de microservicios con el propósito de garantizar la seguridad a lo largo de todo el ciclo de vida de desarrollo, en un enfoque ágil y de apertura. El estudio también resalta los desafíos principales que surgen al implementar arquitecturas de microservicios, como son la integridad y confiabilidad, dado que los servicios de middleware no proveen funciones de seguridad [53].

El proceso de crear una instancia de microservicio expone una interfaz que puede ser vulnerable a ataques a través de la red. La falta de definición de parámetros de seguridad puede llevar a la exposición no deseada del microservicio, y la configuración estática dificulta la gestión de posibles vulnerabilidades en las bibliotecas de software subyacentes [53].

Para abordar estos desafíos, los autores proponen una serie de enfoques y prácticas. Entre estas se encuentran la documentación de

requisitos de seguridad, la mejora de las prácticas de programación, la configuración dinámica de la infraestructura, la supervisión continua de componentes y el empleo de mecanismos de adaptación como SOA, DevOps, orquestación, API gateway, registro de servicios, contenedores Docker, interfaces REST y middleware de seguridad WSSecurity [53].

En resumen, el trabajo destaca la importancia de considerar la seguridad en todas las etapas del desarrollo de microservicios y proporciona recomendaciones y enfoques concretos para garantizar la integridad y confiabilidad en este tipo de arquitecturas.

A.17 Automated Security Analysis for Microservice Architecture.

Chondamrongkul y colaboradores identifican la vulnerabilidad de la seguridad en los microservicios a través de un análisis ontológico de herramientas de modelado como OWL y ADL, los cuales sirven para detectar diferentes problemáticas en el desarrollo de arquitecturas de software [54].

Tras esta investigación se hace énfasis a actuar con antelación a las posibles fallas, antes de que estas se presenten desde una fase inicial en la etapa de diseño, para conseguirlo hacen uso de distintas herramientas de análisis extensible y enfoques específicos de seguridad con técnicas de comprobación de modelos, para demostrar cómo se originan los ataques[54].

En el contexto del análisis de comportamiento, se propone la utilización de dos modelos para comprender y representar de manera efectiva la dinámica y la arquitectura de los sistemas. Estos modelos son el Ontology Web Language (OWL) y el Architecture Description Language (ADL) [54].

Un patrón sugerido es Event-Sourcing and Command and Query Responsibility Segregation CQRS el cual es un patrón de modelado de arquitectura que se aplica a partir de jerarquías [54].

Para detectar características de seguridad, se lleva a cabo la selección de elementos específicos que se encuentran en el diseño del sistema. Estos elementos son evaluados para determinar su nivel de seguridad y su susceptibilidad a vulnerabilidades. Una vez que se identifican los elementos que presentan vulnerabilidades, es posible aplicar patrones de seguridad específicos con el propósito de mantener la seguridad de los microservicio [54].

A.18 A survey on security issues in services communication of Microservices-enabled fog applications.

Yu y colaboradores mencionan los motivos por los cuales los microservicios son vulnerables en lo que respecta a seguridad, ya que las interfaces están distribuidas en distintos microservicios lo cual vuelve complejo garantizar la seguridad total de los mismos y proponen soluciones para estas problemáticas [55].

Dentro de las problemáticas encontradas mencionan: vulnerabilidad en contenedores, seguridad en los datos, protección en la procedencia de datos, autentificación y autorización de los servicios [55].

Las soluciones propuestas son: implementación de estándares abiertos como OAuth que mantiene un acceso controlado, SELinux que se implementa para seguridad en contenedores, uso de Spring Cloud Security framework para autentificación, estándares de cifrado avanzado y de clave compartida y algoritmos de encriptación como AES y RSA para conseguir un nivel de seguridad, mencionan que aun con todas estas implementaciones es difícil garantizar la seguridad total de los microservicios [55].

A.19 Integrating Continuous Security Assessments in Microservices and Cloud Native Applications.

Torkura muestra las formas más utilizadas comúnmente para evaluar la seguridad de un microservicio y las vulnerabilidades que se manifiestan dentro de este, como son: comunicación distribuida, despliegue dinámico, interacción entre microservicios, y múltiples interfases [39].

A partir de un estudio de aplicaciones nativas en la nube (CNA) y la integración de microservicio, concluyen que existen muchas similitudes en los ataques que recibe un CNA y un microservicio ya que la comunicación es multifacética. Actualmente empresas como Heroku, UBER o Netflix son consideradas aplicaciones nativas en la nube, ya que se despliegan utilizando técnicas optimizadas de datos virtuales, que se ejecutan en un solo proceso, denominadas Spring PetClinic. Para lo que se sugiere la implementación de microservicios y así obtener un despliegue independiente [39].

También el uso de puntos de aplicación de seguridad (SEP) en tiempos de ejecución, API Gateway para el enrutamiento de tráfico, patrones de seguridad como: Meath Monitoring Pattern, que emplean métricas de instanciación especificas a través de solicitudes GET y mitigar la inseguridad a través de configuraciones automatizadas [39].

A.21 Towards Concurrent Audit Logging in Microservices.

En este artículo se menciona que el principal problema de seguridad que surge en los microservicios es en los registros de auditoría,

por lo que estudian el despliegue de una herramienta de instrumentación basada en modelos de implementación, con frameworks en java spring que se especifica en JSON [56].

Sugiriendo que el problema de violación de datos puede mitigarse con una aplicación a través del OPEN WEB APPLICATION Security Project, donde propone un marco semántico de información algebraica, que garantiza el registro exclusivo necesario de datos evitando datos innecesarios, que se implementa en procesos lineales, este registro de datos se condiciona a un mismo hilo de ejecución del programa, para ello sugieren un algoritmo que instrumenta sistemas recurrentes, y genera registros de auditoría, que recibe una especificación formal a partir de cláusulas Horn [56].

El sistema recibe código fuente de microservicios, con las especificaciones JSON, a través de LogInst que utiliza un SWI Prolog y programación orientada a aspectos (AOP) [56].

A.22 Recommendations for Enhancing Security in Microservice Environment Altered in an Intelligent Way

En este trabajo hacen un análisis de las problemáticas de seguridad en los procesos de desarrollo y diseño de microservicios, estos parten de hacer una sugerencia para la mitigación de las amenazas de seguridad en lo que respecta a la divulgación y manipulación de datos sensibles [17].

Este parte de un estilo arquitectural ágil que usualmente se emplea para diseñar aplicaciones distribuidas, así como los microservicios, en la que se sugiere la implementación de patrones de diseño con enfoque a patrones de seguridad. Se analizan las principales amenazas STRIDE y con base a estas se proporciona una solución según la categoría de la amenaza [17].

Los patrones sugeridos son: Administrator Hierarchy, Building the Server from the Ground Up, The Checkpointed Systems, Controlled Execution Environment, Controlled Execution Environment, Documenting the Server Configuration, Patching Proactively, Pathname Canonicalization, Protection Rings, Testing on a Staging Server, y Secure IaaS/open IaaS/OpenStack [17].

Posteriormente aplican la metodología CIM (Common Information Model), toman un patrón y lo aplican a microservicios alojados en sistemas operativos independientes que se desarrollan en contenedores, es importante destacar que los autores sugieren que la aplicación de cada patrón se aplique según las necesidades del sistema, partiendo del análisis STRIDE para un tipo de solución aplicable según las necesidades de cada microservicio [17].

A.23 Security Design Patterns in Distributed Microservice Architecture

En este artículo se sugiere que la problemática de seguridad recurrente en los microservicios parte de las técnicas de diseño y la implementación de tecnologías heterogéneas, además del uso de técnicas tradicionales que anteriormente se empleaban en aplicaciones web monolíticas parten de la implementación del MVC modelo vista controlador que parte de un modelo estático, lo cual implica que al ser arquitecturas totalmente distintas las medidas de seguridad son débiles [12].

Para ello proponen una implementación en el área de frontend a través de un mecanismo de autenticación y autorización, a través de tokens de seguridad, a través de formatos RESTful, JSON, Oauth 2.0, OPENID Y pasarelas API privadas diseñadas con el principio de privilegios mínimos, realizar un cifrado de datos con Vormetric transparent encryption con claves de rotación, escaneo CAST y DAST, implementaciones DevSecOps [12].

A.24 Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications

En este trabajo realizan un análisis de las principales problemáticas de seguridad en los microservicios partiendo de las aplicaciones de una arquitectura monolítica, con patrones de diseño [57].

En este emplean modelos de software como servicio SaaS y patrones de diseño nativos de la nube, donde proporcionan una serie de sugerencias de adopción de patrones nativos en la nube [57].

Para ello se deben de considerar los distintos requisitos de migración, como son la seguridad al descomponer una aplicación en componentes más pequeños, la seguridad en la red, la seguridad de los datos, la supervisión de la seguridad [57].

Al considerar los factores anteriores proponen la implementación de contenedores como Dockers, entornos en la nube PaaS o IaaS o la función de ambos creando una nube OpenSatack, implementación de API Gateway con Framework Spring Cloud, orquestación nativa en la nube, para conseguir un OpenSTACK Data base-as-a-Service DbaaS, motores de orquestación de aplicaciones en la nube TOSCA y AWS [57].

A.25 Interface quality patterns - Communicating and improving the quality of microservices APIs

En este trabajo realizan un estudio a diversos APIs WEB, en los que analizan los distintos patrones encontrados y aplicados a microservicios, en las cuales se gestiona la calidad del servicio, y el costo del mismo, para garantizar la calidad del servicio al emplear los recursos existentes [58].

Donde exponen que la mayor parte de sensibilidad en lo que respecta a seguridad parte de la ligereza encontrada en la parte del backend, ya que se usa generalmente una arquitectura orientada a servicios [58].

Para ello sugieren patrones de representación de interfaces de estructuras de mensajes en APIs remotas que soluciona en sus distintas fases la problemática de definición de datos repetitivos, anidados, primitivos, simples y compuestos tales como son: Atomic Parameter, Atomic Parameter List, Parameter Tree, Parameter Forest [58].

Los patrones sugeridos que prevén la calidad en lo que concierne a comunicación y calidad de la interfaz, donde consideran los aspectos de fiabilidad, escalabilidad y rendimiento que son denominadas propiedades de calidad de servicio QoS como son: API Key, Wish List, Rate Limit, Rate Plan, Service Level Agreement [58].

A.26 Analyzing the Relevance of SOA Patterns for Microservice-Based Systems

En este trabajo analizan 118 patrones de arquitecturas SOA, que se aplican en microservicios, de los cuales se derivó que al menos el 163% de los patrones son totalmente aplicables a microservicios, a su vez el 25% parcialmente aplicables y solo el 12% no son aplicables, lo cual deriva innumerables problemáticas de seguridad, descentralización y de sistema único. En lo que concierne a la seguridad catalogados como patrones de seguridad y gestión solo se encuentran 5 patrones [59].

Usualmente tras este análisis coinciden que una arquitectura de microservicios comparte muchas similitudes con una arquitectura de sistemas basados en servicios, además exponen que SOA provee un sinfín de beneficios tales como la encapsulación, interoperabilidad, reutilización, todos como tal beneficios a una arquitectura en específico pero no para microservicios por lo que proponen los pros y contras de emplearlos a microservicios y las similitudes que se deben explotar sin descuidar las lagunas que no se cubren totalmente [59].

Las principales diferencias son la independencia, descentralización, la aplicación en sistemas pequeños o medianos, reutilización, abstracción y grado pequeño de centralización [59].

Los patrones existentes más usados en microservicios son: Enterprise Inventory, Protocol Bridging, pattern Lightweight Endpoint, Lightweight Communication, patrones SOA, y REST-inspired patterns [59].

A.27 Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API.

En esta investigación se realiza un análisis de OAuth2 y Spring Security Framework para su aplicación en arquitecturas de microservicios, en los que realizan pruebas de seguridad de concepto POC donde analizan la vulnerabilidad para ver qué tan fiables son, con el propósito de reducir las innumerables interrogantes respecto a la seguridad en microservicios y que tan recomendable es su uso en MSA [31].

Las principales problemáticas de seguridad encontradas en las MSA son en el acceso y adaptación de recursos para una función, la cual se encuentra en la comunicación de APIs por lo que podríamos decir que es la parte más vulnerable y en donde según la literatura existente hasta el momento no se considera específicamente, para asegurarlos. Donde sugiere una aplicación basada en microservicios desarrollados en Java e implementar OAuth2 y Spring Security Framework en conjunto [31].

A.30 How Microservices are Changing the Security Landscape.

En este documento se habla de las necesidades de seguridad especificas en microservicios, dónde y cuáles son las mejores opciones de implementación para cubrir las problemáticas de seguridad en microservicios, además expone cuales son las problemáticas latentes que aún no consiguen una solución óptima para cubrir esa necesidad [60].

Los principales retos de seguridad son contenedor, datos, permisos, red, comunicación, aplicación, orquestación y seguridad de red [60].

En este trabajo aclaran que existen muchas similitudes de ataque dentro de una arquitectura monolítica y una arquitectura de microservicios, pero en hecho de que los ataques sean similares no indica que la mitigación o prevención de estas sean igual, ya que menciona que en una arquitectura monolítica se hacen comprobaciones de seguridad en una solo ocasión, mientras que en una arquitectura de microservicios lo ideal es que sea de forma continua, en cada punto de acceso, lo cual implica que se envíen solicitudes a servicios remotos [60].

Dentro de los patrones utilizados en microservicios se considera: API Gateway Pattern, Sidecar Pattern, Service Mesh Pattern, implementación de protocolos de transporte TLS, HTTPS, DoS/DDoS protection, OpenAPI, REST APIs, API key-based, OAuth 2.0, OpenID Connect Discovery, frontends pattern, JWT, WebSocets, mTLS with X.509 certificates, JWS, JWE y se sugiere la aplicación del principio mínimo de privilegios, además de implementar claves y certificados de seguridad [60].

A.31 Identifying Availability Tactics to Support Security Architectural Design of Microservice-based Systems.

En este artículo se analizan el código fuente de código abierto y la documentación de sistemas para proporcionar las tácticas de prevención de fallas para sistemas basados en microservicios, considerando atributos de calidad, integridad, confidencialidad y disponibilidad, para proveer una ayuda en la toma de decisiones, en el diseño de arquitectura de microservicios [61].

Los patrones suelen proporcionar soluciones a problemas específicos y son empleados en gran medida para mantener los atributos de calidad de los sistemas, al proporcionar soluciones genéricas estos construyen un marco de tácticas para su desarrollo [61].

En el que se concluye que la mayoría de las técnicas arquitectónicas se centran en la prevención de problemas de disponibilidad, prevención de fallos, pero las soluciones son más concentradas en la parte de disponibilidad, los patrones propuestos son: Circuit breaker, Service Registry, Messaging [61].

A.32 Detecting Cyber Security Attacks against a Microservices Application using Distributed Tracing.

En este trabajo emplean un rastreo distribuido para ataque de ciberseguridad, que se ejecutan en los usuarios finales, detectando eventos o patrones irregulares, para ello utiliza un sistema de rastreo distribuido que monitorea el comportamiento [62].

El rastreo distribuido es un proceso de monitorización perfilado y registro de la ruta de ejecución, que proporciona una respuesta a la solicitud de usuarios, a través de aplicaciones nativas en la nube, que se puede llevar de forma supervisada clasifica y etiqueta los datos obtenidos, o no supervisada es aquella que no requiere datos entrenados, estas se pueden clasificarse por anomalías puntuales y de grupo, en este trabajo consideran que existen formas de aprender el comportamiento de los datos temporales por medio de redes neuronales que detectan los datos anómalos, sin embargo no proporciona soluciones para prevenir el tipo de ataque encontrado [62].

Dentro de las herramientas encontradas se observan: DeathStarBench, Docker, Thrift, Jaeger, ElasticSearch [62].

A.33 Decision Models for Selecting Patterns and Strategies in Microservices Systems and their valuation by Practitioners.

En este trabajo se realiza un análisis de las distintas estrategias de diseño y seguridad que se han adoptado a arquitecturas de microservicios, así como la implementación de patrones y los retos que conllevan estas nuevas tecnologías, con el fin de proponer una

serie de modelos de decisión conforme a las etapas de diseño de una arquitectura de microservicios, este análisis evalúa la integridad, comprensión y utilidad de los modelos, a través de una serie de entrevistas a 24 profesionales en el ámbito y desarrollo de microservicios de los cinco continentes en al menos 12 países, con el fin de proporcionar una guía para la selección de estrategias y patrones para una arquitectura de microservicios [40].

Esta arquitectura consta de elementos esenciales de diseño como son: descomposición, seguridad, comunicación, y descubrimiento de servicios [40].

Los principales retos que se han encontrado son principalmente en el área de diseño que implica la definición de un microservicio hasta sus medidas de seguridad [40].

Este proporciona una serie de propuestas de mitigación como son: Service-level authorization, Scenario-based analysis, Edge-level authorization, API rate limiting, Encrypt and protect secrets, and Scan dependencies strategies, API gateway, Backend for frontend (BFF) patterns, Anti-corruption layer, Idempotent consumer, Asynchronous requestreply, Publish-asynchronous messaging, Publish-subscribe messaging, Asynchronous messaging, Remote procedure invocation, Proxy microservices [40].

Anexo B – Bitácora de pruebas

Documentación de pruebas

En la Tabla 17 se observan la plantilla para la ejecución del plan de pruebas, que sirvió para registrar los resultados obtenidos en cada prueba.

Tabla 17 Plantilla para los resultados de casos de prueba

Nombre de prueba	Identificador del caso de prueba.		
Autor	Nombre del ejecutor del caso de prueba.	**Fecha**	--/----/----
Tipo de prueba	Tipo de prueba.		
DP a evaluar	Diseño de pruebas evaluado con el caso de prueba.		
Etapa 1	Nombre del elemento que será probado.		
Objetivo	Objetivo buscado con la correcta ejecución del caso de prueba.		
Descripción			
Nombre de la etapa	**Descripción de prueba**	**Resultado**	
Etapa	Descripción de la prueba	Resultado obtenido después de ejecutar la instancia de prueba.	
Observaciones	Observaciones sobre la ejecución de la instancia de prueba 1.		

Las pruebas se realizaron en cuatro etapas: La etapa uno, consistió en probar que la API Gateway cumpla con su principal propósito de gestionar y redirigir las solicitudes a los microservicios de los clientes. La etapa dos consistió en probar que el JWT genera, refresca y proporciona el token, también que el algoritmo de codificación H256 recibe y encripta el token. La etapa tres corroboró que las anteriores etapas funcionaran de la forma correcta al integrarse. Finalmente, la etapa cuatro consistió en implementar el Microservice Security Pattern API Gateway en diferentes lenguajes de programación.

De la Tabla 18 a Tabla 20 se muestra la ejecución de pruebas.

B.1 Ejecución de la prueba MSPAG-DP-01

Tabla 18 Resultados de casos de prueba etapa 1

Nombre de prueba	MSPAG-DP-CP-01		
Autor	Karen Mónica Hernández Guzmán	**Fecha**	23/octubre/2023
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MSPAG-DP-CP-01		
Etapa 1	API Gateway		
Objetivo	• Evaluar el funcionamiento correcto de la API Gateway. • La API Gateway se encarga de gestionar el acceso a los microservicios		
Descripción			
Nombre de la etapa	**Descripción de prueba**	**Resultado**	
Etapa 1	En esta prueba se ingresan las URL de los diferentes microservicios, para probar que la API Gateway, está cumpliendo adecuadamente con su función.	La API Gateway cumple con las características adecuadas gestionando y redirigiendo el acceso a los microservicios	
Observaciones	No se presentó problema.		

B.2 Ejecución de la prueba MSPAG-DP-02

Tabla 19 Resultados de casos de prueba etapa 2

Nombre de prueba	MSPAG-DP-CP-02		
Autor	Karen Mónica Hernández Guzmán	**Fecha**	23/noviembre/2023
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MSPAG-DP-CP-02		
Etapa 2	JWT y H256		
Objetivo	• Evaluar el funcionamiento correcto del JWT Y H256. • El JWT, genera, crea y refresca el token, para ser encriptado por el algoritmo de codificación H256, una vez que el usuario ingresa sus datos		
Descripción			
Nombre de la etapa	**Descripción de prueba**	**Resultado**	
Etapa 2	En esta prueba se crea, refresca y actualiza un token, que posteriormente Sera encriptado con la función H256. para probar que el JWT Y H25 están cumpliendo adecuadamente con su función.	El JWT y el algoritmo H256 cumple con las características adecuadas, de generar, validar y encriptar el token para acceder a los microservicios.	
Observaciones	No se presentó problema.		

B.3 Ejecución de la prueba MSPAG-DP-03 C#

Tabla 20 Resultados de casos de prueba etapa 3 en C#

Nombre de prueba	MSPAG-DP-CP-03 C#		
Autor	Karen Mónica Hernández Guzmán	**Fecha**	7/diciembre/2023
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MSPAG-DP-CP-03 C#		
Etapa 3	Microservice Security Pattern API Gateway		
Objetivo	• Evaluar el funcionamiento correcto del Microservice Security Pattern API Gateway .		
Descripción			
Nombre de la etapa	**Descripción de prueba**	**Resultado**	
Etapa 3	En esta prueba se ingresan las URL de los diferentes microservicios, para probar que la API Gateway, está cumpliendo adecuadamente con su función además de que se crea, refresca y actualiza un token, que posteriormente será encriptado con la función H256. Para probar que el JWT Y H25 están cumpliendo adecuadamente con su función.	Microservice Security Pattern API Gateway cumple con las características adecuadas gestionando y redirigiendo el acceso a los microservicios, así mismo el JWT y el algoritmo H256 cumple con las características adecuadas, de generar, validar y encriptar el token para acceder a los microservicios.	
Observaciones	No se presentó problema.		

B.4 Ejecución de la prueba MSPAG-DP-03 JAVA

Tabla 21 Resultados de casos de prueba etapa 3 en Java

Nombre de prueba	MSPAG-DP-CP-03 JAVA		
Autor	Karen Mónica Hernández Guzmán	**Fecha**	30/mayo/2023
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MSPAG-DP-CP-03 JAVA		
Etapa 4	Microservice Security Pattern API Gateway		
Objetivo	• Evaluar el funcionamiento correcto del Microservice Security Pattern API Gateway en JAVA.		
Descripción			
Nombre de la etapa	**Descripción de prueba**	**Resultado**	
Etapa 4.1	En esta prueba se ingresan las URL de los diferentes microservicios, para probar que la API Gateway, está cumpliendo adecuadamente con su función además de que se crea, refresca y actualiza un token, que posteriormente Sera encriptado con la función H256. P ara probar que el JWT Y H25 están cumpliendo adecuadamente con su función.	Microservice Security Pattern API Gateway cumple con las características adecuadas gestionando y redirigiendo el acceso a los microservicios, así mismo el JWT y el algoritmo H256 cumple con las características adecuadas, de generar, validar y encriptar el token para acceder a los microservicios.	
Observaciones	No se presentó problema.		

B.5 Ejecución de la prueba MSPAG-DP-03 Python

Tabla 22 Resultados de casos de prueba etapa 3 en Python

Nombre de prueba	MSPAG-DP-CP-03 Python		
Autor	Karen Mónica Hernández Guzmán	**Fecha**	10/mayo/2023
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MSPAG-DP-CP-03 Python		
Etapa 3	Microservice Security Pattern API Gateway		
Objetivo	• Evaluar el funcionamiento correcto del Microservice Security Pattern API Gateway, en Python.		
Descripción			
Nombre de la etapa	**Descripción de prueba**	**Resultado**	
Etapa 1	En esta prueba se ingresan las URL de los diferentes microservicios, para probar que la API Gateway, está cumpliendo adecuadamente con su función además de que se crea, refresca y actualiza un token, que posteriormente Sera encriptado con la función H256. P ara probar que el JWT Y H25 están cumpliendo adecuadamente con su función.	Microservice Security Pattern API Gateway cumple con las características adecuadas gestionando y redirigiendo el acceso a los microservicios, así mismo el JWT y el algoritmo H256 cumple con las características adecuadas, de generar, refrescar y encriptar el token para acceder a los microservicios.	
Observaciones	No se presentó problema.		

Anexo C – Patrón de seguridad implementado en Python

Documentación de Implementación en Python

C.1. API Gateway Antes

En la Figura 32 se muestra el diagrama de clases, API Gateway es la clase principal que maneja la solicitud de la API y enruta la solicitud al microservicio correspondiente.

La clase API Gateway contiene dos objetos de microservicio: SkincareDayMicroservice y SkincareNightMicroservice.

La clase API Gateway también tiene dos métodos de ruta skincare_day_route() y skincare_night_route() que enrutan las solicitudes al microservicio de cuidado de la piel correspondiente.

Cada microservicio (SkincareDayMicroservice y SkincareNightMicroservice) tiene un método get_skincare_tips() que devuelve consejos para el cuidado de la piel. Cada microservicio tiene su propia implementación del método get_skincare_tips().

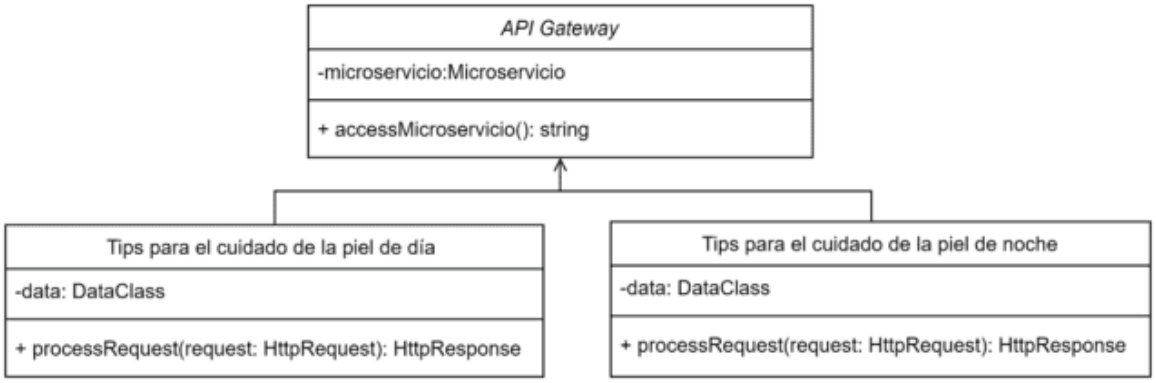


Figura 32 Diagrama de clases API Gateway Antes.

C.2. Preparación del ambiente virtual Python

Para preparar el ambiente virtual en Python y crear una API Gateway que permita el acceso a dos microservicios del cuidado de la piel en Visual Studio Code, siga los siguientes pasos:

1.1.1.- Abra Visual Studio Code y cree un nuevo proyecto Code en la Figura 33 se muestra en menú de VSC.

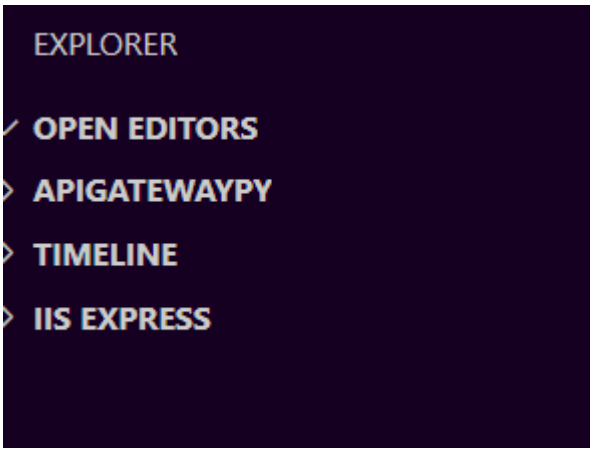


Figura 33 Vista VSC.

1.1.2.- Cree un ambiente virtual de Python dentro del proyecto utilizando el siguiente comando en la terminal de Visual Studio Code.

>>python -m venv env

Lo que creara la carpeta env.

1.1.3.- Active el ambiente virtual utilizando el siguiente comando en la terminal Code >> .\env\Scripts\activate

1.1.4.- Instale los paquetes necesarios para su proyecto, como Flask y Flask-RESTful, utilizando el siguiente comando en la terminal Code

>>pip install flask flask-restful

>>pip install flask

C.3. Creación de la API GATEWAY

Cree un archivo Python para la API Gateway. Este archivo importará las clases de los microservicios y creará una instancia de Flask-RESTful que enrute las solicitudes a los microservicios adecuados Code.

Este archivo importa las clases "SkincareDay" y "SkincareNight" de los archivos de microservicios correspondientes y enruta las solicitudes a las URLs "/skincare/day" y "/skincare/night" a los microservicios adecuados Code.

1.2.2.- Codificar

from flask import Flask

from flask_restful import Resource, API

```
from skincare_day import SkincareDay

from skincare_night import SkincareNight


app = Flask(__name__)

API = API(app)


API.add_resource(SkincareDay, '/skincare/day')

API.add_resource(SkincareNight, '/skincare/night')


if __name__ == '__main__':

 app.run(debug=True)
```

C.4. Compilación de la API Gateway

Para compilar utilizaR el siguiente comando en la terminal Code "API_gateway.py".

Al acceder a los microservicios creados se muestra la información contenida en cada microservicio, como son los tips del cuidado de la piel, con solo usar la URL del microservicio.

C.5. Creación de microservicios

Cree un archivo Python para cada microservicio que desea crear. Por ejemplo, si desea crear un microservicio para obtener información sobre el cuidado de la piel para el día, cree un archivo llamado "skincare_day.py". Si desea crear un microservicio para obtener información sobre el cuidado de la piel para la noche, cree un archivo llamado "skincare_night.py".

En cada archivo Python, importe Flask y Flask-RESTful y cree una clase que herede de la clase "Resource" de Flask-RESTful. Esta clase contendrá los métodos HTTP que se utilizarán para acceder al microservicio.

C.6. skincare_night.py

Codificar lo siguiente

```
import random

from flask_restful import Resource


class SkincareNight(Resource):
```

```
 def get(self):
 tips = ['Limpia tu piel antes de dormir para evitar que se obstruyan los poros.',
 'Usa una crema hidratante para nutrir tu piel mientras duermes.',
 'Aplica un tratamiento para el acné antes de acostarte para reducir los brotes.',
 'Usa una almohada de seda para evitar que se formen arrugas en tu rostro.',
 'Bebe suficiente agua antes de acostarte para mantener tu piel hidratada.',
 'Usa un exfoliante suave antes de acostarte para eliminar las células muertas de la piel.']
 return {'tip': random.choice(tips)}
```

C.7 Pruebas del sistema API GATEWAY ANTES

C.7.1. Prueba del microservicio day

Al ingresar la URL específica en el navegador, el sistema muestra la información correspondiente en cada microservicio.

Abrir el navegador y acceder a la URL del microservicio:

http://127.0.0.1:5000/skincare/day

En la Figura 34 visualizan los datos contenidos en el microservicio, es decir, los consejos para el cuidado de la piel.

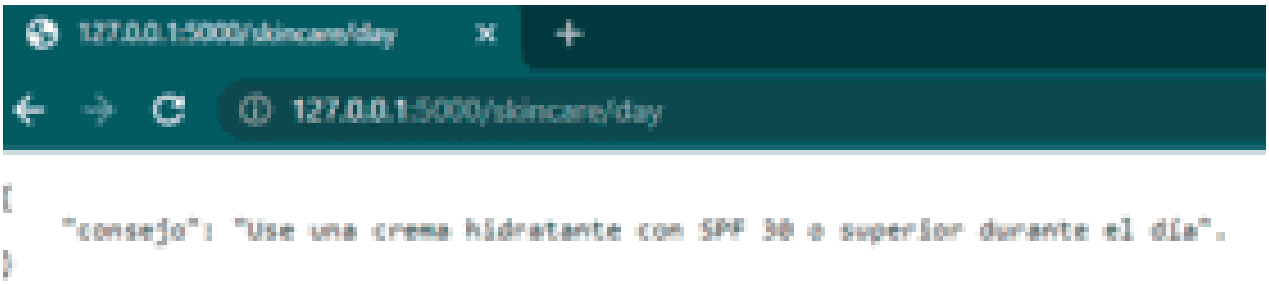


Figura 34 Prueba en el navegador del microservicio day.

Se debe cambiar la URL utilizando los datos del segundo microservicio, en este caso: http://127.0.0.1:5000/skincare/night.

Al hacerlo, se mostrará la información reservada para ese microservicio, como en la Figura 35.

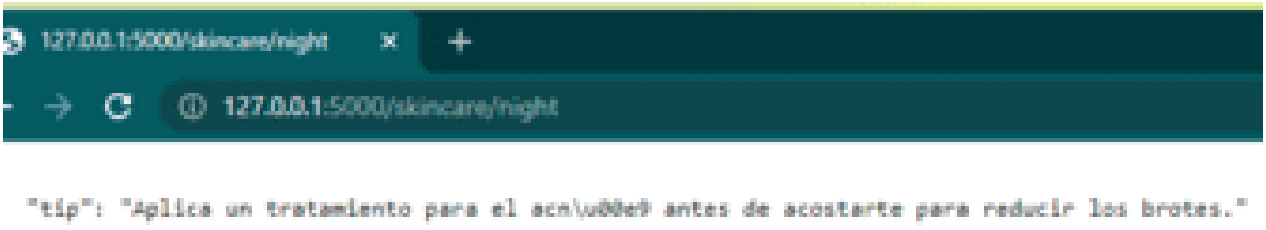


Figura 35 Prueba en el navegador del microservicio night.

Lo anterior también se prueba en postman para validar que los datos y el funcionamiento es correcto

C.8. Pruebas en postman

C.8.1 Microservicio day

Abrir postman.

New

HTTP Request

Ingresar la URL en el método GET, como se muestra en la Figura 36. http://127.0.0.1:5000/skincare/day y dar clic en send.

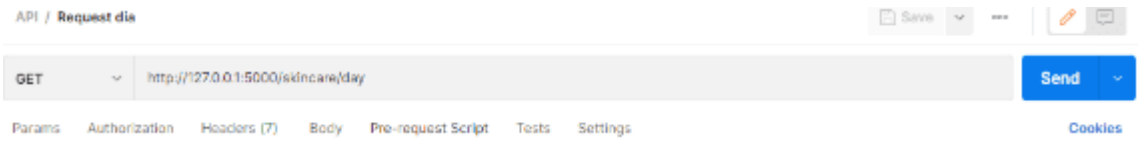


Figura 36 Método GET microservicio Day.

Lo cual mostrara los datos albergados en este microservicio, como se observa en la Figura 37.

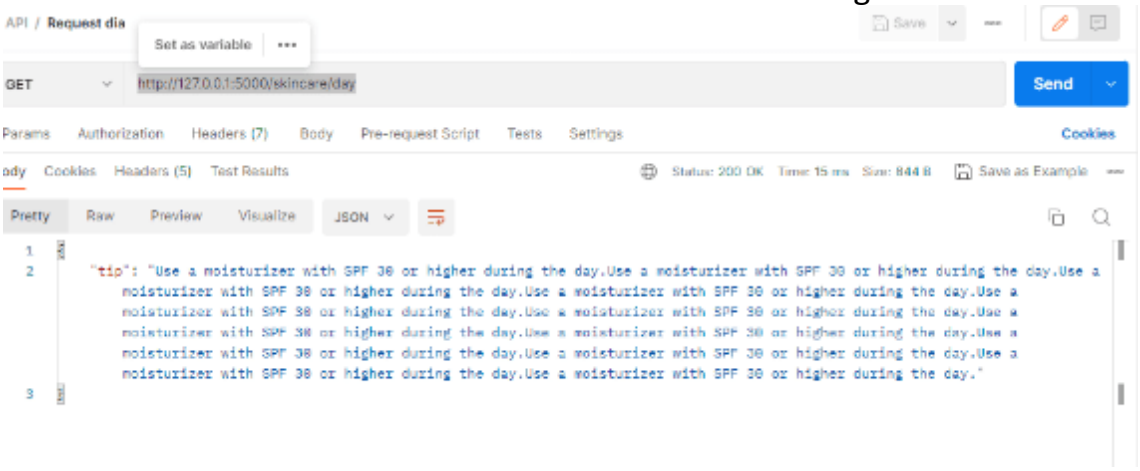


Figura 37 Datos del microservicio day.

C.8.1 Prueba microservicio night

Abrir postman en New HTTP Request

Ingresar la URL en el método GET, como se muestra en la Figura 38. http://127.0.0.1:5000/skincare/night y dar clic en send

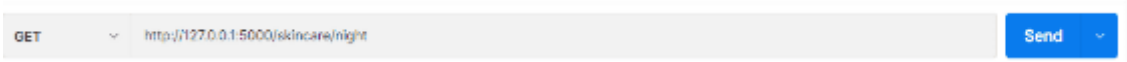


Figura 38 Método GET microservicio night.

Al dar clic se mostrarán los datos albergados, como se observa en la Figura 39.

Figura 39 Datos del Microservice night.

Y es así como funciona la API Gateway, como punto de acceso a los microservicios.

C.9. Descripción de código API GATEWAY ANTES

ENV

Entorno virtual en Python, se crea un espacio aislado para la instalación de las bibliotecas y dependencias de un proyecto específico. Este espacio aislado se llama "entorno virtual", y se utiliza para evitar conflictos entre diferentes proyectos que puedan requerir diferentes versiones de las mismas bibliotecas o dependencias [63].

El objeto "env" que se crea en este contexto se refiere a un diccionario que contiene variables de entorno específicas del entorno virtual. Estas variables de entorno se utilizan para Configurar el entorno virtual en tiempo de ejecución, y pueden incluir variables como la ruta de búsqueda de paquetes, el directorio de instalación de Python, y otras variables que afectan el comportamiento del entorno virtual [64].

Por ejemplo, cuando se activa un entorno virtual en Python, se establece la variable de entorno "PATH" para que incluya la ruta de búsqueda del intérprete de Python y de las bibliotecas instaladas en el entorno virtual. De esta manera, el intérprete de Python y las

bibliotecas instaladas en el entorno virtual se utilizan en lugar de los instalados en el sistema operativo [64].

Básicamente el objeto "env" que se crea al generar un entorno virtual en Python se utiliza para almacenar y acceder a variables de entorno específicas del entorno virtual, que se utilizan para configurar el entorno virtual en tiempo de ejecución y evitar conflictos entre diferentes proyectos.

API_gateway.py

from flask import Flask

La línea de código "from flask import Flask" se utiliza en Python para importar la clase "Flask" del framework web Flask. Flask es un framework web minimalista y flexible para Python que se utiliza para crear aplicaciones web y APIs [65].

La clase "Flask" es una parte fundamental de Flask y se utiliza para crear una instancia de una aplicación Flask. Esta clase es la base sobre la cual se construyen las aplicaciones web en Flask [64].

La clase "Flask" proporciona métodos para manejar las solicitudes HTTP que se envían a la aplicación, como GET, POST, PUT, DELETE, entre otros. También proporciona métodos para configurar la aplicación, definir rutas, manejar errores y excepciones, entre otras funcionalidades [66].

from flask_restful import Resource, API

La línea de código "from flask_restful import Resource, API" se utiliza en Python para importar las clases "Resource" y "API" del paquete "flask_restful", que proporciona herramientas para crear servicios web RESTful utilizando Flask [67].

El término RESTful se refiere a una arquitectura de software para sistemas distribuidos basados en la web, que se centra en el uso de los métodos HTTP (GET, POST, PUT, DELETE, entre otros) para manipular recursos en un servidor. Flask-RESTful es una extensión de Flask que hace que sea más fácil crear servicios web RESTful en Python [68].

La clase "Resource" es una parte fundamental de Flask-RESTful y se utiliza para definir recursos RESTful. Un recurso es una abstracción de algo que se puede acceder a través de una API web, como un objeto o una colección de objetos. Los métodos definidos en la clase "Resource" corresponden a los métodos HTTP que se utilizan para manipular los recursos, como GET, POST, PUT, DELETE, entre otros [65].

La clase "API" se utiliza para agregar los recursos definidos en la clase "Resource" a una aplicación Flask. La clase "API" proporciona un mecanismo para registrar las rutas de URL asociadas con cada recurso y para asignar las funciones de controlador correspondientes a cada método HTTP [65].

En general, la línea de código "from flask_restful import Resource, API" se utiliza para importar las clases "Resource" y "API" del paquete "flask_restful", que se utilizan para crear servicios web RESTful en Flask. La clase "Resource" se utiliza para definir recursos RESTful y los métodos correspondientes, mientras que la clase "API" se utiliza para agregar los recursos a la aplicación Flask.

from skincare_day import SkincareDay

La línea de código from skincare_day import SkincareDay en el archivo API_gateway.py se utiliza para importar la clase SkincareDay desde el módulo skincare_day. Esto es necesario porque la clase API Gateway en API_gateway.py utiliza una instancia de SkincareDay, que se almacena en su atributo privado skincare_day.

Al importar la clase SkincareDay, se crea instancia de esta clase dentro de la clase API Gateway y utilizar sus métodos públicos, como el método get_recommendation(), para proporcionar recomendaciones de cuidado de la piel de día a través de la API Gateway.

from skincare_night import SkincareNight

La línea de código from skincare_night import SkincareNight en el archivo API_gateway.py se utiliza para importar la clase SkincareNight desde el módulo skincare_night. Esto es necesario porque la clase API Gateway en API_gateway.py utiliza una instancia de SkincareNight, que se almacena en su atributo privado skincare_night.

Al importar la clase SkincareNight, se hace posible crear una instancia de esta clase dentro de la clase API Gateway y utilizar sus métodos públicos, como el método get_recommendation(), para proporcionar recomendaciones de cuidado de la piel de noche a través de la API Gateway.

app = Flask(__name__)

Se utiliza para crear una instancia de la clase Flask del framework Flask en Python [65].

La clase Flask se utiliza para crear una aplicación web en Python que puede recibir solicitudes HTTP y responder con una respuesta

adecuada. En esta línea de código, __name__ es el nombre del módulo actual en el que se está ejecutando la aplicación.

La instancia de la clase Flask que se crea se utiliza para definir las rutas de la API (a través de los decoradores @app.route()) y para iniciar el servidor web (a través del método app.run()) para escuchar las solicitudes entrantes y enviar las respuestas adecuadas. En resumen, esta línea de código es el primer paso para crear una aplicación web usando Flask en Python.

```
API = API(app)
```

Se utiliza para crear una instancia de la clase API proporcionada por la extensión flask_restful de Flask [63].

La clase API se utiliza para crear una API RESTful, que permite a los clientes realizar solicitudes HTTP a la aplicación web y recibir una respuesta en formato JSON u otro formato deseado. La instancia de la clase API que se crea se utiliza para agregar los recursos de la API, que son los objetos que manejan las solicitudes entrantes y generan las respuestas correspondientes.

```
API.add_resource(SkincareDay, '/skincare/day')
```

Se utiliza para agregar un recurso a la API RESTful [69].

La función API.add_resource() es proporcionada por la extensión flask_restful y se utiliza para agregar recursos a la API. Toma dos argumentos: el primero es el nombre de la clase que define el recurso y el segundo es la ruta URL que se utilizará para acceder a este recurso.

En este caso, se está agregando el recurso SkincareDay a la API con la ruta URL /skincare/day. Esto significa que cuando un cliente realice una solicitud HTTP a http://<server>/skincare/day, Flask llamará al método correspondiente en la clase SkincareDay para manejar la solicitud y generar la respuesta correspondiente.

```
API.add_resource(SkincareNight, '/skincare/night')
```

Se utiliza para agregar otro recurso a la API RESTful.

Al igual que la línea de código anterior, se está utilizando la función API.add_resource() para agregar un recurso a la API, en este caso el recurso SkincareNight. La ruta URL asociada con este recurso es /skincare/night, lo que significa que cuando un cliente realice una solicitud HTTP a http://<server>/skincare/night, Flask llamará al método correspondiente en la clase SkincareNight para manejar la solicitud y generar la respuesta correspondiente.

```
if __name__ == '__main__':
```

app.run(debug=True)

La línea de código if __name__ == '__main__': app.run(debug=True) se utiliza para ejecutar la aplicación Flask cuando se ejecuta el archivo API_gateway.py directamente.

En Python, cuando se importa un archivo, se ejecuta todo el código en ese archivo, incluyendo cualquier línea de código que esté fuera de cualquier función o clase. Sin embargo, cuando un archivo se ejecuta como un programa independiente, el código solo se ejecuta si la línea de código if __name__ == '__main__': se evalúa como True [63].

En este caso, si se ejecuta el archivo API_gateway.py directamente, la línea if __name__ == '__main__': se evaluará como verdadera y Flask ejecutará la aplicación llamando al método app.run(). El parámetro debug=True indica que Flask debe ejecutarse en modo de depuración, lo que permite ver mensajes detallados de error en el navegador web en caso de que algo falle en la aplicación.

skinkcare_day

skinkcare_day from flask import Flask y from flask_restful import Resource, API

La línea de código from flask import Flask se utiliza para importar la clase Flask de la biblioteca Flask.

La clase Flask es la clase principal en Flask y se utiliza para crear instancias de una aplicación Flask. Es decir, crea una instancia de la aplicación web en sí misma.

La línea de código from flask_restful import Resource, API se utiliza para importar la clase Resource y la clase API de la biblioteca Flask-RESTful.

La clase Resource es una clase base que se utiliza para crear recursos RESTful en una aplicación Flask. Un recurso es una representación de un objeto o colección de objetos en una aplicación RESTful. La clase Resource proporciona métodos para recibir y enviar datos a través de HTTP.

La clase API se utiliza para agregar recursos a una aplicación Flask-RESTful y manejar rutas y solicitudes HTTP. La clase API se encarga de agregar las rutas al enrutador Flask y de manejar las solicitudes HTTP, como las solicitudes GET, POST, PUT y DELETE.

app = Flask(__name__) y API = API(app)

Esta sesión del código se utiliza para crear una instancia de la clase Flask de la biblioteca Flask y luego crear una instancia de la

clase API de la biblioteca Flask-RESTful. Flask es un marco de aplicaciones web que permite a los desarrolladores crear aplicaciones web en Python de manera fácil y rápida. La clase API de Flask-RESTful proporciona una forma fácil de crear una API RESTful en Flask, lo que significa que puede crear servicios web que se adhieren a los principios de la arquitectura REST. Al pasar la instancia de la clase Flask a la clase API, se vincula la API con la instancia de la aplicación Flask. Esto permite que los recursos creados con Flask-RESTful se accedan a través de la aplicación Flask y se expongan en la dirección URL especificada en la clase de recurso.

class SkincareDay(Resource):

La clase SkincareDay define un recurso que puede ser accedido mediante una solicitud GET. Al hacer una solicitud GET al recurso /skincare/day, se devuelve un diccionario con un mensaje de recomendación sobre el cuidado de la piel durante el día. En este caso, el mensaje es "Use a moisturizer with SPF 30 or higher during the day."

La clase SkincareDay hereda de la clase Resource de Flask-RESTful y tiene un único método get() que devuelve un diccionario serializado en formato JSON

API.add_resource(SkincareDay, '/skincare/day')

La función add_resource() de Flask-RESTful se utiliza para agregar recursos a la API. En este caso, se está agregando la clase SkincareDay como un recurso a la API con la ruta /skincare/day. Esto significa que cuando se haga una solicitud GET a la ruta /skincare/day, Flask-RESTful llamará automáticamente al método get() de la clase SkincareDay.

El módulo random es una biblioteca estándar de Python que proporciona funciones para generar números aleatorios. Se está importando para que la clase SkincareNight pueda usar la función randint() de la biblioteca random para generar un número aleatorio [63].

C.10. SkincareNight

from flask_restful import Resource

La línea from flask_restful import Resource importa la clase Resource de Flask-RESTful, que se usa para crear recursos en la API Flask-RESTful.

class SkincareNight(Resource):

La clase SkincareNight es una subclase de Resource que define una función get que devuelve un tip de cuidado de la piel aleatorio para

la noche. La lista tips contiene diferentes consejos de cuidado de la piel, y la función random.choice(tips) selecciona uno de ellos al azar y lo devuelve en formato JSON como un diccionario con la clave tip. Esta clase se utiliza en la API Gateway para manejar solicitudes GET al endpoint /skincare/night.

C.11. MICROSERVICE SECURITY PATTERN API GATEWAY IMPLEMENTACIÓN EN PYTHON

C.11.1 Diagrama de clases MPSAG-PY

La Figura 40 muestra el API Gateway que se encarga de verificar los tokens JWT antes de reenviar las solicitudes a los microservicios respectivos. Esto garantiza la seguridad y la autenticación adecuada antes de acceder a los servicios de cuidado de la piel.

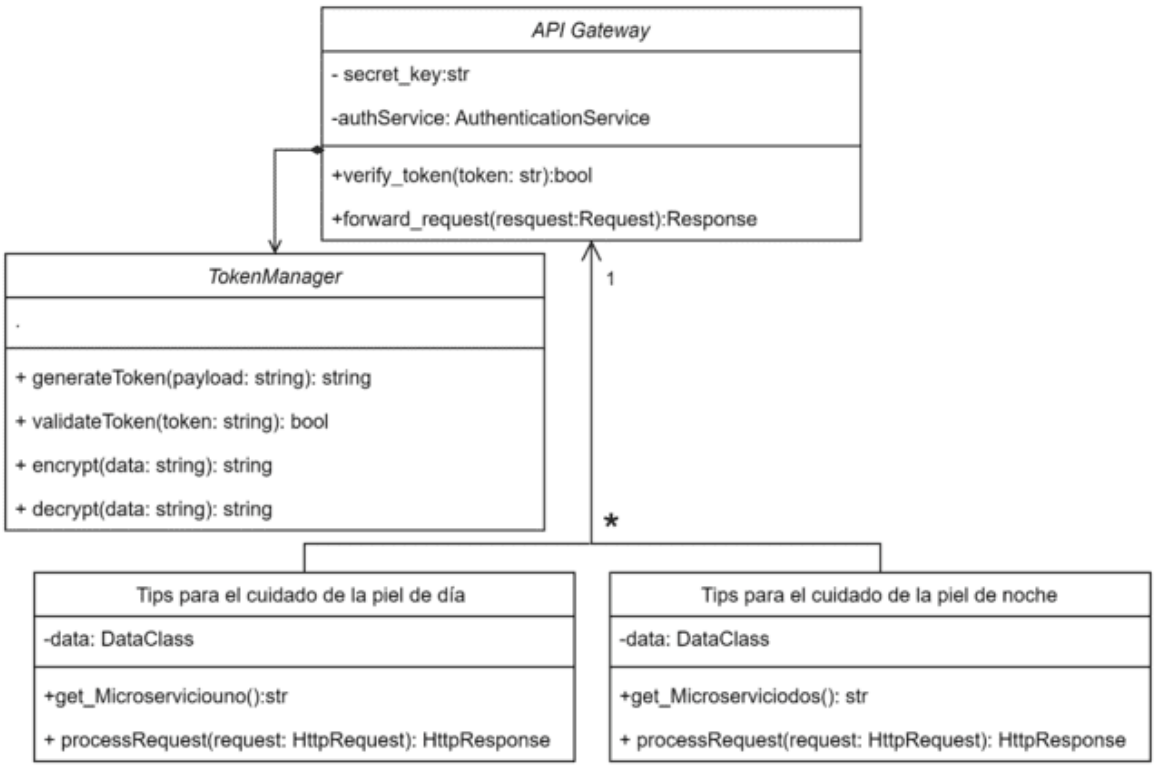


Figura 40 Diagrama de clases MPSAG.

API Gateway: Esta clase es responsable de recibir las solicitudes y autenticar los tokens JWT utilizando las claves secretas secret_key_day y secret_key_night. Luego, reenvía la solicitud al microservicio correspondiente.

SkincareDay: Este es el microservicio de día que proporciona el método get_tip() para obtener un consejo de cuidado de la piel.

SkincareNight: Este es el microservicio de noche que también proporciona el método get_tip() para obtener un consejo de cuidado de la piel.

La API Gateway se encarga de verificar los tokens JWT antes de reenviar las solicitudes a los microservicios respectivos. Esto garantiza la seguridad y la autenticación adecuada antes de acceder a los servicios de cuidado de la piel.

C.11.2 Preparación del ambiente virtual Python

1. AGREGAR EXTENCIONES.

Flask: Framework web para crear la API Gateway.

Flask-RESTful: Extensión de Flask para crear servicios web RESTful.

PyJWT: Biblioteca para generar y verificar tokens JWT.

cryptography: Biblioteca para manejar algoritmos de hash.

Importa las bibliotecas necesarias:

```
from flask import Flask, request, jsonify

from flask_restful import Resource, API

import jwt

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.hmac import HMAC
```

2. Extensiones

```
>> pip install flask

>> pip install flask-restful

>>pip install pyjwt

>> pip install cryptography
```

C.12. Creación del MPSAG

C.12.1. Importar bibliotecas las bibliotecas necesarias en el contexto de un API Gateway:

```
from flask import Flask, request, jsonify

from flask_restful import Resource, API

import jwt

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.hmac import HMAC
```

En la API Gateway se importan las siguientes bibliotecas:

Flask: La biblioteca principal para crear aplicaciones web en Flask.

request: Proporciona acceso a los datos de la solicitud HTTP recibida.

jsonify: Permite convertir objetos de Python en respuestas JSON.

Flask-RESTful: Extensión de Flask para crear servicios web RESTful.

jwt: Biblioteca para generar y verificar tokens JWT.

cryptography: Biblioteca para manejar algoritmos de hash y otros criptográficos.

C.12.2. Crea la instancia de la aplicación Flask:

```
app.config['SECRET_KEY'] = 'mysecretkey321'
```

C.12.3. Definir la función para generar el token JWT:

En la Figura 41 se muestra el generate_token(), genera un token JWT válido utilizando la biblioteca PyJWT. El payload es un diccionario que contiene los datos que deseas incluir en el token, como el nombre de usuario. La secret_key es una clave secreta compartida entre la API Gateway y los microservicios para firmar y verificar el token.

Definir la función para verificar el token JWT:

```
def verify_token(token):
 try:
 jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
 return True
 except jwt.exceptions.DecodeError:
 return False
```

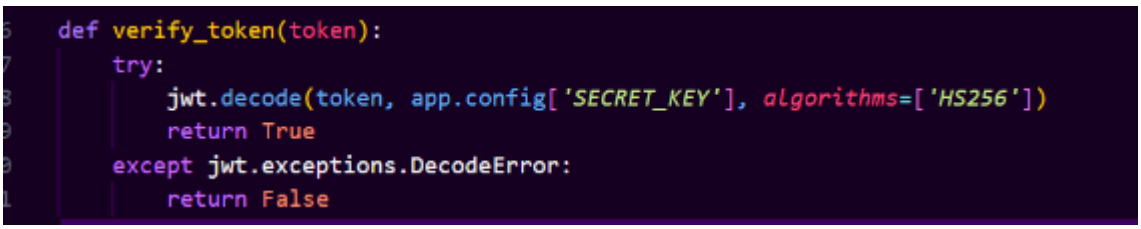


Figura 41 Definición del JWT.

verify_token toma un token JWT como argumento y verifica su validez utilizando la clave secreta configurada en la aplicación Flask (app.config['SECRET_KEY']). Si el token es válido, la función devuelve True; de lo contrario, devuelve False.

C.13.4. Definir una clase para proteger los microservicios

```
class ProtectedResource(Resource):
```

```
def get(self):
token = request.headers.get('Authorization', '').split(' ')[1]
if verify_token(token):
# Código para manejar la solicitud si el token es válido
return {'message': 'Acceso permitido'}
else:
return {'message': 'Acceso denegado'}, 401
```

C.13.5. Recursos protegidos a la API Gateway

Agrega los recursos protegidos, como se muestra en la Figura 42.

```
API.add_resource(ProtectedResource, '/skincare/day')
API.add_resource(ProtectedResource, '/skincare/night')
```

Figura 42 Agregación de recursos API GATEWAY.

C.13. Pruebas del MPSAG

C.13.1. MPSAG

Ejecuta el archivo API_gateway.py utilizando el siguiente comando, como se muestra en la Figura 43:

```
>> Python API_gateway.py
```

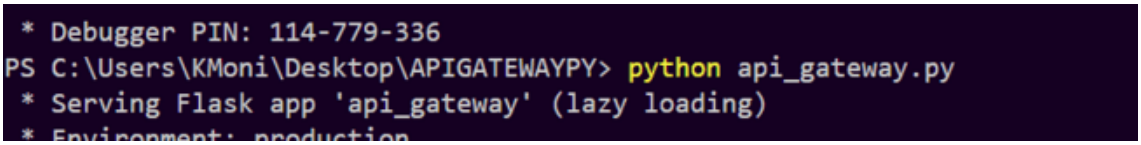


Figura 43 Compilación de API_gateway.py.

La aplicación Flask se ejecutará en modo de depuración y estará escuchando en http://127.0.0.1:5000. Ahora puedes realizar solicitudes GET a http://127.0.0.1:5000/skincare/night con un encabezado de autorización que contenga un token JWT válido para obtener un consejo de cuidado de la piel aleatorio.

Mientras que en la terminal se muestra el JWT encriptado con el H256, como se muestra en la Figura 44.

```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImttaGcifQ.OVXli
rpDwNBa-R98AsZFW96IWCKkBQfiDL345m9mqmc
 * Debugger is active!
 * Debugger PIN: 114-779-336
```

Figura 44 Muestra en terminal del JWT.

Entonces el token JWT se está generando correctamente y se imprime en la consola. El token está codificado en base64 y consta de tres partes separadas por puntos. La primera parte es el encabezado, la segunda parte es el payload (contiene los datos del usuario en este caso), y la tercera parte es la firma.

C.13.2. Prueba de primer microservicio Day

Abre Postman y crea una nueva solicitud, selecciona el método HTTP GET y en la URL, ingresa la siguiente dirección: http://127.0.0.1:5000/skincare/day.

Haz clic en la pestaña "Headers".

Agrega un nuevo encabezado con la clave "Authorization" y el valor "Bearer <token>". Reemplaza <token> con el token JWT válido generado por la función generate_token(), como se muestra en la Figura 45.

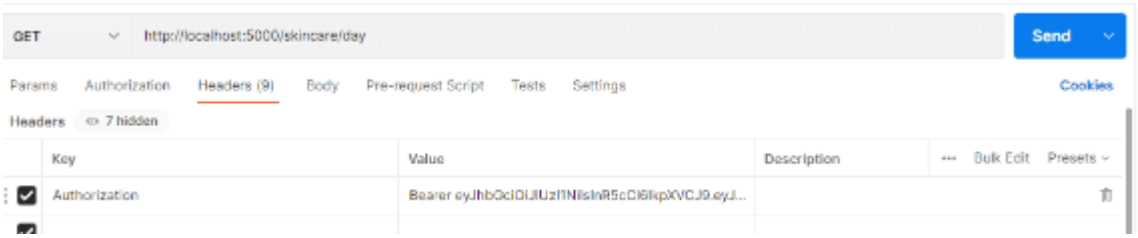


Figura 45 Configuración y uso del JWT.

Haz clic en el botón "Send" para enviar la solicitud. Lo cual mostrará el acceso correcto, como se muestra en la Figura 46.

Figura 46 Prueba de acceso.

En caso de que se modifique la fracción del token y luego se ejecute el método "get", el sistema mostrará un mensaje indicando que el acceso ha sido denegado, como se muestra en la Figura 47.

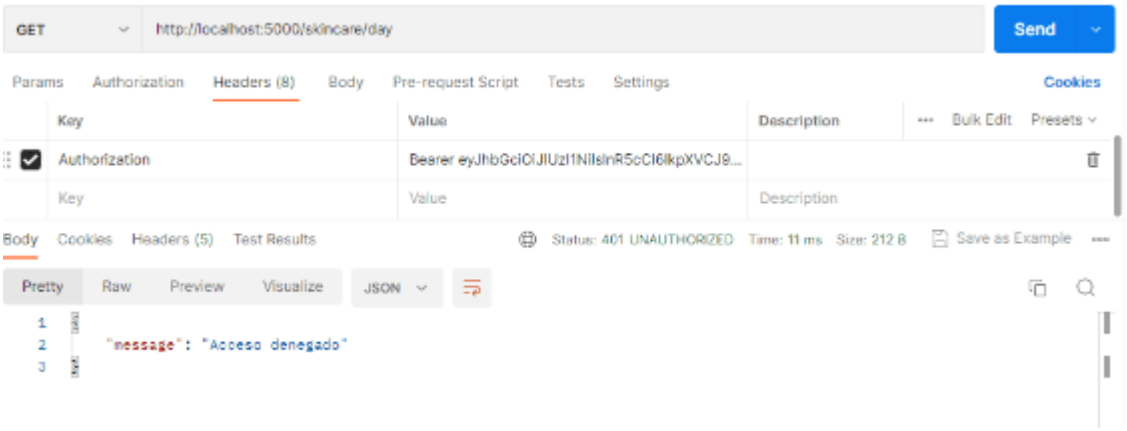


Figura 47 Prueba de acceso denegado por cambiar datos en el JWT.

Y en terminal muestra el error 401 de validación ya que no es correcto, como se muestra en la Figura 48.

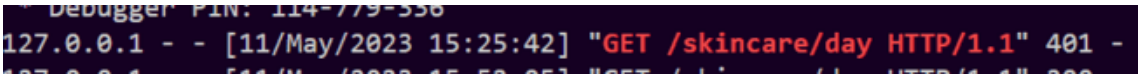


Figura 48 Muestra en terminal del acceso denegado.

C.13.3. Prueba de microservicio NIGHT

Abre Postman y crea una nueva solicitud. Selecciona el método HTTP GET.

En la URL, ingresa la siguiente dirección: http://127.0.0.1:5000/skincare/night. Haz clic en la pestaña "Headers". Agrega un nuevo encabezado con la clave "Authorization" y el valor "Bearer <token>". Reemplaza <token> con el token JWT válido generado por la función generate_token(), como se muestra en la Figura 49.

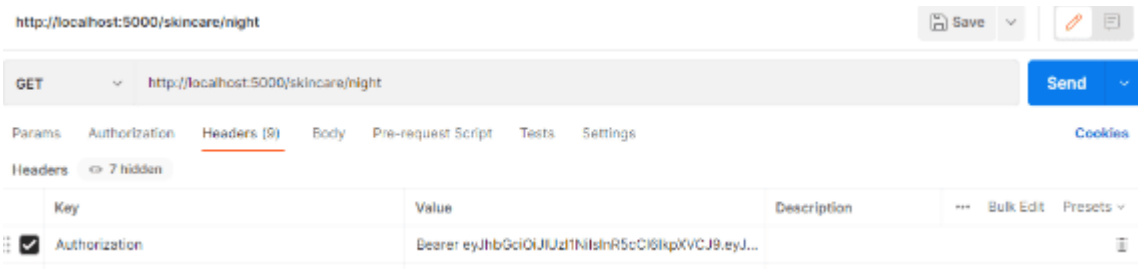


Figura 49 Configuración y uso del JWT.

Haz clic en el botón "Send" para enviar la solicitud.

Lo cual muestra el acceso correcto, como se muestra en la Figura 50.

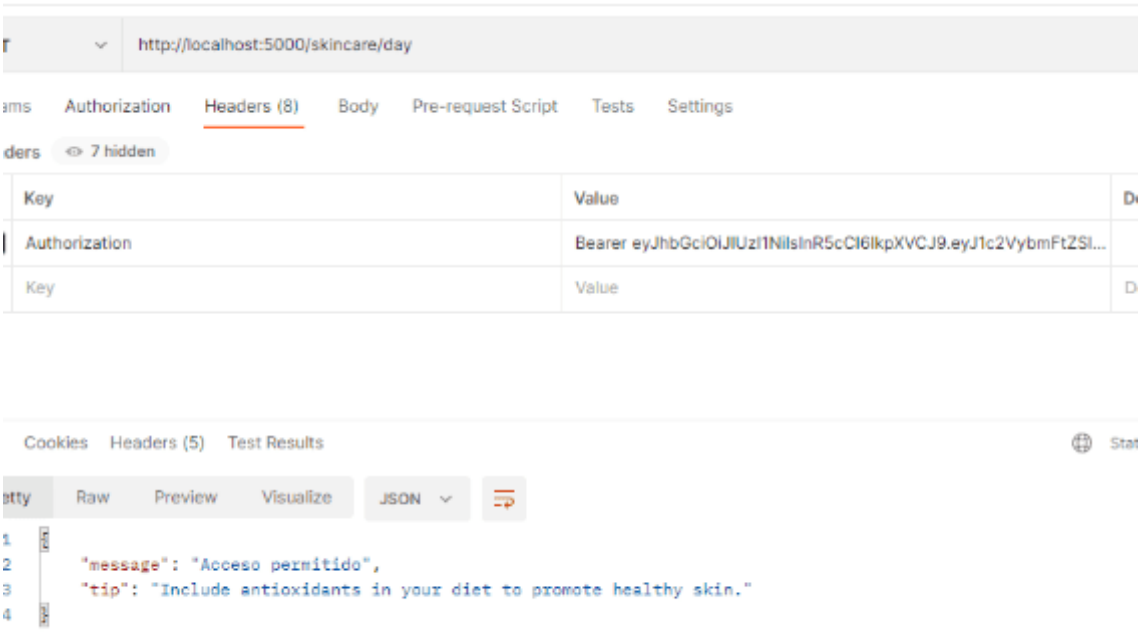


Figura 50 Prueba de acceso permitido en day.

C.14. ¿Cómo se comprueba que es seguro?

Verifica el algoritmo de cifrado: Asegurando de que se esté utilizando un algoritmo de cifrado fuerte y ampliamente aceptado, como HMAC-SHA256.

Valida la longitud y complejidad del token: Un token seguro debe tener una longitud suficientemente larga y contener una combinación de caracteres alfanuméricos y símbolos.

Protege la clave secreta: Asegúrate de que la clave secreta utilizada para firmar y verificar el token se mantenga de forma segura. Evita compartir o almacenar la clave en lugares accesibles.

C.15. Descripción de código MSPAG

C.15.1. API Gateway

```
from flask import Flask, request
```

"Flask" es una clase que se utiliza para crear una instancia de la aplicación web Flask. Representa la aplicación en sí misma y se utiliza para definir las rutas, las vistas y otras configuraciones relacionadas con la aplicación.

"request" es un objeto que proporciona acceso a la información enviada por el cliente en una solicitud HTTP. Permite acceder a los datos del formulario enviado, los parámetros de la URL, las cabeceras HTTP y otros detalles de la solicitud.

En resumen, esta línea de código es necesaria para utilizar el framework Flask y acceder a la funcionalidad básica que ofrece, como la creación de una aplicación web y el manejo de solicitudes HTTP entrantes.

from flask_restful import Resource, API

"Resource" es una clase base que se utiliza para definir recursos en una API RESTful. Un recurso representa una entidad o un conjunto de datos que se pueden acceder a través de la API. Los métodos de esta clase, como get(), post(), put(), delete(), etc., se utilizan para definir las acciones que se pueden realizar en el recurso.

"API" es una clase que se utiliza para agrupar recursos y proporcionar una interfaz fácil de usar para configurar y agregar recursos a una aplicación Flask. Proporciona métodos para agregar recursos a rutas específicas y manejar la configuración de la API en general.

En resumen, esta línea de código es necesaria para utilizar Flask-RESTful, una extensión de Flask que facilita la creación de API RESTful. La importación de las clases "Resource" y "API" permite definir recursos y agruparlos en una API que se puede acceder a través de rutas específicas.

import JWT

La línea de código "import jwt" se utiliza para importar el módulo "jwt" en Python. JWT (JSON Web Tokens) es un estándar de token de seguridad que se utiliza para la autenticación y autorización en aplicaciones web y APIs.

El módulo "jwt" proporciona funciones y clases para trabajar con JWT, incluyendo la generación y verificación de tokens, la firma y verificación de claves, y el manejo de las diferentes partes de un token JWT, como el encabezado, los datos (payload) y la firma.

Al importar el módulo "jwt", se pueden utilizar las funciones y clases proporcionadas por el módulo para trabajar con tokens JWT y realizar operaciones como la generación de tokens de acceso, la verificación de la validez de un token recibido y la extracción de información del token.

import random

La línea de código "import random" se utiliza para importar el módulo "random" en Python.

El módulo "random" proporciona funciones relacionadas con la generación de números pseudoaleatorios en Python. Al importar este módulo, se pueden utilizar varias funciones para generar números

aleatorios, seleccionar elementos aleatorios de secuencias, mezclar el orden de los elementos en una secuencia y realizar otras operaciones relacionadas con la aleatoriedad.

Al utilizar la importación "import random", se puede acceder a estas funciones y utilizarlas en el código para generar números aleatorios o realizar operaciones que requieran aleatoriedad.

from config import Config

La línea de código "from config import Config" se utiliza para importar la clase "Config" desde un archivo llamado "config.py" en el directorio actual.

En este caso, se asume que hay un archivo llamado "config.py" que contiene la definición de una clase llamada "Config". Esta clase puede ser utilizada para almacenar la configuración de una aplicación o módulo específico.

Al importar la clase "Config" utilizando esta línea de código, se puede acceder a las variables y métodos definidos en la clase para acceder y modificar la configuración de la aplicación.

Es común utilizar un archivo de configuración para centralizar la configuración de una aplicación, lo que permite ajustar fácilmente los valores de configuración sin necesidad de modificar el código fuente.

app = Flask(__name__)

El código app = Flask(__name__) crea una instancia de la aplicación Flask. El parámetro __name__ representa el nombre del módulo actual.

La instancia de la aplicación Flask es fundamental para desarrollar aplicaciones web utilizando el framework Flask. Se utiliza para configurar y definir las rutas, las vistas, los controladores y otros aspectos de la aplicación.

En la línea de código proporcionada, se crea una instancia de la aplicación Flask llamada app utilizando Flask(__name__). El parámetro __name__ se utiliza para indicarle a Flask el nombre del módulo actual, lo que ayuda a Flask a determinar la ubicación de las plantillas y los archivos estáticos asociados a la aplicación.

Después de crear la instancia de la aplicación app, se pueden agregar más configuraciones y definiciones de rutas y vistas para construir una aplicación web completa.

app.config.from_object(Config)

La línea de código app.config.from_object(Config) se utiliza para cargar la configuración de la aplicación Flask desde un objeto de configuración.

Aquí se asume que hay una clase llamada Config definida en algún lugar del código. Esta clase puede contener variables que representan la configuración de la aplicación, como la URL de la base de datos, claves secretas, opciones de depuración, etc.

Al llamar a app.config.from_object(Config), se carga la configuración definida en la clase Config en la configuración de la aplicación Flask. Esto permite acceder a las variables de configuración a través del objeto app.config en toda la aplicación.

Utilizar un objeto de configuración separado proporciona una forma organizada de gestionar las opciones de configuración de la aplicación. Puede ser útil para cambiar fácilmente la configuración sin tener que modificar directamente el código fuente de la aplicación.

API = API(app)

La línea de código API = API(app) crea una instancia de la clase API del módulo Flask-RESTful y la asocia con la aplicación Flask app.

Flask-RESTful es una extensión de Flask que facilita la creación de APIs RESTful. Proporciona funcionalidades adicionales para definir recursos, rutas y métodos HTTP asociados con esos recursos.

Al crear una instancia de API con API = API(app), se establece una conexión entre la instancia de la aplicación Flask app y la instancia de API llamada API. Esto permite utilizar Flask-RESTful para definir y gestionar los recursos de la API de manera más sencilla.

A través de la instancia de API, se pueden agregar rutas y recursos a la aplicación Flask app utilizando las funcionalidades proporcionadas por Flask-RESTful.

```
def generate_token_day():

 payload = {'username': 'kmhg'}

 secret_key = app.config['SECRET_KEY_DAY'] # Clave secreta para firmar el token

 token = jwt.encode(payload, secret_key, algorithm='HS256')

 print('Generated Token (Day):', token) # Mostrar el token generado

 return token
```

Genera un token JWT (JSON Web Token) utilizando la biblioteca jwt en Python, En esta función, se define un payload que contiene los datos que se incluirán en el token. En este caso, el payload solo contiene el nombre de usuario "kmhg", pero se puede modificar para incluir más información según sea necesario.

Luego, se obtiene la clave secreta SECRET_KEY_DAY desde la configuración de la aplicación Flask mediante app.config['SECRET_KEY_DAY']. Esta clave se utiliza para firmar el token y asegurar su integridad.

A continuación, se utiliza la función jwt.encode() para generar el token. Recibe el payload, la clave secreta y el algoritmo de firma (en este caso, 'HS256' para HMAC-SHA256).

Por último, el token generado se imprime en la consola y se devuelve como resultado de la función.

```
def generate_token_night():

 payload = {'username': 'kmhg'}

 secret_key = app.config['SECRET_KEY_NIGHT'] # Clave secreta para firmar el token

 token = jwt.encode(payload, secret_key, algorithm='HS256')

 print('Generated Token (Night):', token) # Mostrar el token generado

 return token
```

Al igual que en la función anterior, se define un payload con los datos que se incluirán en el token. En este caso, también contiene el nombre de usuario "kmhg".

Se obtiene la clave secreta SECRET_KEY_NIGHT desde la configuración de la aplicación Flask mediante app.config['SECRET_KEY_NIGHT']. Esta clave se utiliza para firmar el token y garantizar su integridad.

Luego, se utiliza la función jwt.encode() para generar el token, utilizando el payload, la clave secreta y el algoritmo de firma 'HS256'.

Finalmente, el token generado se imprime en la consola y se devuelve como resultado de la función.

```
def verify_token_day(token):

 try:

 jwt.decode(token, app.config['SECRET_KEY_DAY'], algorithms=['HS256'])
```

```
 return True
 except jwt.exceptions.DecodeError:
 return False
def verify_token_night(token):
 try:
 jwt.decode(token,                  app.config['SECRET_KEY_NIGHT'],
algorithms=['HS256'])
 return True
 except jwt.exceptions.DecodeError:
 return False
```

Esto permite tener dos funciones independientes para verificar los tokens generados en diferentes momentos.

En verify_token_day, se intenta decodificar el token utilizando la clave secreta SECRET_KEY_DAY de la configuración de la aplicación Flask. Si la decodificación tiene éxito, se retorna True; de lo contrario, se captura la excepción DecodeError y se retorna False.

En verify_token_night, se realiza un proceso similar, pero utilizando la clave secreta SECRET_KEY_NIGHT en lugar de SECRET_KEY_DAY.

```
class ProtectedResource(Resource):
 def get(self):
 token = request.headers.get('Authorization', '').split(' ')[1]
 if verify_token(token):
 tip = get_skincare_day_tip() # Obtener el tip del microservicio de
día
 return {'message': 'Acceso permitido', 'tip': tip}
 else:
 return {'message': 'Acceso denegado'}, 401
```

Define una clase llamada ProtectedResource que hereda de la clase Resource del módulo Flask-RESTful. Esta clase se utiliza para crear un recurso protegido en tu API.

La función get() dentro de la clase ProtectedResource se ejecutará cuando se realice una solicitud GET a este recurso protegido

Dentro de la función get(), se obtiene el token JWT de la solicitud HTTP a través del encabezado de autorización. Se espera que el token esté en el formato "Bearer <token>", por lo que se divide el encabezado y se obtiene el token.

Luego, se verifica el token utilizando la función verify_token() que has definido previamente. Si el token es válido, se llama a la función get_skincare_day_tip() para obtener un consejo relacionado con el cuidado de la piel para el día.

Si el token es inválido, se devuelve una respuesta de "Acceso denegado" con un código de estado HTTP 401 (No autorizado).

```
def get_skincare_day_tip():

 tips = [

 'Use a moisturizer with SPF 30 or higher during the day.',

 'Drink plenty of water to keep your skin hydrated.',

 'Avoid touching your face throughout the day to prevent bacteria buildup.',

 'Apply sunscreen even on cloudy days to protect your skin from UV rays.',

 'Include antioxidants in your diet to promote healthy skin.',

 'Gently cleanse your face in the morning to remove impurities.'

 ]

 return random.choice(tips)
```

Devuelve un consejo aleatorio relacionado con el cuidado de la piel durante el día.

Dentro de la función, se define una lista tips que contiene varios consejos relacionados con el cuidado de la piel durante el día. Cada elemento de la lista representa un consejo.

Luego, se utiliza random.choice(tips) para seleccionar aleatoriamente un consejo de la lista. La función random.choice() toma una lista y devuelve un elemento aleatorio de esa lista.

Finalmente, el consejo seleccionado aleatoriamente se devuelve como resultado de la función.

```
class ProtectedResourceNight(Resource):

 def get(self):

 token = request.headers.get('Authorization', '').split(' ')[1]
```

```
 if verify_token(token):
 tip = get_skincare_night_tip() # Obtener el tip del microservicio de noche
 return {'message': 'Acceso permitido', 'tip': tip}
 else:
 return {'message': 'Acceso denegado'}, 401
```

La clase ProtectedResourceNight es similar a la clase ProtectedResource que has definido previamente. Esta clase también hereda de la clase Resource del módulo Flask-RESTful y se utiliza para crear un recurso protegido en tu API para el período nocturno.

Dentro de la función get(), se obtiene el token JWT de la solicitud HTTP a través del encabezado de autorización de la misma manera que en la clase ProtectedResource.

Luego, se verifica el token utilizando la función verify_token_night() que has definido previamente. Esta función verifica si el token es válido utilizando la clave secreta SECRET_KEY_NIGHT.

Si el token es válido, se llama a la función get_skincare_night_tip() para obtener un consejo relacionado con el cuidado de la piel para la noche.

Si el token es inválido, se devuelve una respuesta de "Acceso denegado" con un código de estado HTTP 401 (No autorizado).

```
def get_skincare_night_tip():
 tips = [
 'Remove all makeup antes going to bed to allow your skin to breathe.',
 'Apply a night cream or moisturizer to hydrate your skin while you sleep.',
 'Use a gentle cleanser to wash your face antes bedtime.',
 'Consider using a retinoid or anti-aging serum in your nighttime skincare routine.',
 'Avoid sleeping on your stomach to prevent wrinkles and acne caused by friction.'
 ]
 return random.choice(tips)
```

get_skincare_night_tip() devuelve un consejo aleatorio relacionado con el cuidado de la piel durante la noche.

Dentro de la función, se define una lista tips que contiene varios consejos relacionados con el cuidado de la piel durante la noche. Cada elemento de la lista representa un consejo.

Luego, se utiliza random.choice(tips) para seleccionar aleatoriamente un consejo de la lista. La función random.choice() toma una lista y devuelve un elemento aleatorio de esa lista.

Finalmente, el consejo seleccionado aleatoriamente se devuelve como resultado de la función.

API.add_resource(ProtectedResource, '/skincare/day')

API.add_resource(ProtectedResourceNight, '/skincare/night')

Se utiliza el método API.add_resource() para agregar las clases ProtectedResource y ProtectedResourceNight como recursos en tu API.

El método API.add_resource() se utiliza para asignar una clase de recurso a una ruta específica en tu API. En este caso, estás agregando las clases ProtectedResource y ProtectedResourceNight como recursos en las rutas '/skincare/day' y '/skincare/night', respectivamente.

Esto significa que cuando se realice una solicitud GET a '/skincare/day', se ejecutará el método get() de la clase ProtectedResource. Y cuando se realice una solicitud GET a '/skincare/night', se ejecutará el método get() de la clase ProtectedResourceNight.

if __name__ == '__main__':

 token_day = generate_token_day()

 token_night = generate_token_night()

 app.run(debug=True)

El bloque if __name__ == '__main__': es una convención común en Python para asegurarse de que el código dentro de este bloque solo se ejecute cuando el archivo se ejecuta directamente y no cuando se importa como un módulo.

Dentro del bloque, se generan los tokens JWT para el período del día y la noche utilizando las funciones generate_token_day() y generate_token_night() respectivamente.

Luego, se llama al método app.run(debug=True) para ejecutar la aplicación Flask en modo de depuración. El argumento debug=True permite que la aplicación muestre mensajes de depuración detallados en caso de errores.

Este bloque de código asegura que la generación de tokens y la ejecución de la aplicación Flask solo ocurran cuando el archivo se ejecuta directamente. Si el archivo se importa como un módulo en otro lugar, este bloque no se ejecutará.

config.py

import os

import os importa el módulo os en tu programa. El módulo os proporciona funciones para interactuar con el sistema operativo en el que se está ejecutando tu programa.

class Config:

Es una clase que se utiliza comúnmente para almacenar la configuración de una aplicación Flask. se utiliza para almacenar la configuración de la aplicación y se puede acceder a ella utilizando app.config

```
 SECRET_KEY_DAY = '9mysecretkey321'

 SECRET_KEY_NIGHT = 'AKMhg4key7'
```

Te da acceso al valor de la clave secreta definida en la clase Config. Ya que se utilizan para firmar y verificar los tokens JWT en el caso de la clase ProtectedResource y ProtectedResourceNight que has definido previamente.

Es importante mantener estas claves secretas seguras y protegidas. Puedes utilizar claves secretas más fuertes y complejas para garantizar la seguridad de tus tokens.

C.15.2. SkincareDay

```
from flask import Flask

from flask_restful import Resource, API

import jwt

from config import Config
```

Importa los módulos y las clases necesarios para construir una aplicación Flask con la extensión Flask-RESTful, utilizando JWT para la autenticación y la configuración definida en una clase Config.

Flask: Es la clase principal de Flask y se utiliza para crear una instancia de la aplicación Flask.

Flask_RESTful: Es una extensión de Flask que simplifica la creación de APIs RESTful.

Se pudo verificar que cuando el usuario tiene conocimiento de la ruta o URL del microservicio y la ingresa en el buscador, la puerta de enlace API Gateway niega el acceso.: Es el módulo que proporciona la funcionalidad para trabajar con tokens JWT (JSON Web Tokens).

Config: Es una clase que define la configuración de la aplicación Flask. Aquí se asume que la clase Config está definida en un archivo llamado config.py en el mismo directorio.

```
app = Flask(__name__)

app.config.from_object(Config)

API = API(app)
```

Crea una instancia de la aplicación Flask, carga la configuración desde la clase Config y crea una instancia de la clase API para trabajar con la API de Flask-RESTful.

Flask(__name__): Crea una instancia de la aplicación Flask con el nombre del módulo actual. El parámetro __name__ es una variable especial en Python que se refiere al nombre del módulo o del paquete.

app.config.from_object(Config): Carga la configuración de la aplicación desde la clase Config. La clase Config debe tener los atributos que definen la configuración de la aplicación, como la clave secreta, variables de entorno, etc.

API(app): Crea una instancia de la clase API utilizando la instancia de la aplicación Flask. Esto permite trabajar con la API de Flask-RESTful y definir rutas y recursos para la API.

```
def verify_token(token):

 try:

 jwt.decode(token,                    app.config['SECRET_KEY_DAY'],
algorithms=['HS256'])

 return True

 except jwt.exceptions.DecodeError:

 return False
```

La función verify_token(token) verifica la validez de un token JWT utilizando la clave secreta SECRET_KEY_DAY definida en la configuración de la aplicación.

Dentro de la función, se utiliza el método jwt.decode() para decodificar el token JWT y verificar su validez. Se proporciona el token a decodificar, la clave secreta SECRET_KEY_DAY desde la configuración de la aplicación y se especifica el algoritmo utilizado para firmar el token (HS256 en este caso).

Si el token se decodifica correctamente sin generar ninguna excepción jwt.exceptions.DecodeError, se considera que el token es válido y la función devuelve True. De lo contrario, si se genera una excepción DecodeError, se considera que el token es inválido y la función devuelve False.

```
class SkincareDay(Resource):

 def get(self):

 token = request.headers.get('Authorization', '').split(' ')[1]

 if verify_token(token):

 tip = get_skincare_day_tip() # Obtener el tip del microservicio de día

 return {'tip': tip}

 else:

 return {'message': 'Acceso denegado'}, 401
```

Es un recurso de la API Flask-RESTful que se utiliza para manejar las solicitudes GET relacionadas con el cuidado de la piel durante el día.

Dentro de la clase SkincareDay, se define el método get(self) que se ejecuta cuando se realiza una solicitud GET a la ruta asociada con este recurso. Dentro de este método, se obtiene el token de autenticación del encabezado de la solicitud, se verifica su validez utilizando la función verify_token(token), y si el token es válido, se obtiene un tip de cuidado de la piel durante el día utilizando la función get_skincare_day_tip().

Si el token es válido, se devuelve un diccionario con la clave 'tip' que contiene el tip de cuidado de la piel durante el día. Si el token no es válido, se devuelve un diccionario con la clave 'message' que indica el acceso denegado y se establece el código de estado HTTP a 401 (No autorizado).

```
def get_skincare_day_tip():

 tips = [

 'Use a moisturizer with SPF 30 or higher during the day.',
```

```
 'Drink plenty of water to keep your skin hydrated.',
 'Avoid touching your face throughout the day to prevent bacteria buildup.',
 'Apply sunscreen even on cloudy days to protect your skin from UV rays.',
 'Include antioxidants in your diet to promote healthy skin.',
 'Gently cleanse your face in the morning to remove impurities.'
 ]
 return random.choice(tips)
```

Devuelve un tip de cuidado de la piel durante el día seleccionado al azar de una lista de tips.

Dentro de la función, se define una lista de tips de cuidado de la piel durante el día. Utilizando la función random.choice(tips), se selecciona aleatoriamente uno de los tips de la lista y se devuelve.

Cada vez que se llama a get_skincare_day_tip(), se obtendrá un tip diferente de forma aleatoria de la lista.

```
API.add_resource(SkincareDay, '/skincare/day')
if __name__ == '__main__':
 app.run(debug=True)
```

Añade el recurso SkincareDay a la API Flask-RESTful y ejecuta la aplicación Flask en modo de depuración.

La línea API.add_resource(SkincareDay, '/skincare/day') añade el recurso SkincareDay a la API Flask-RESTful con la ruta /skincare/day. Esto significa que cuando se haga una solicitud GET a la ruta /skincare/day, la clase SkincareDay será utilizada para manejar la solicitud.

La condición if __name__ == '__main__': se utiliza para asegurarse de que el archivo actual se esté ejecutando como el programa principal. Esto es útil para evitar que el código se ejecute si el archivo es importado como un módulo en otro lugar. Dentro de esta condición, se ejecuta app.run(debug=True) para iniciar el servidor Flask en modo de depuración.

C.15.3. SkincareNight

```
from flask import Flask, request
from flask_restful import Resource, API
import jwt
```

```
import random
```

Importa los módulos necesarios para construir una aplicación Flask con la extensión Flask-RESTful, utilizar JWT para la autenticación y generar números aleatorios.

Flask: Es la clase principal de Flask y se utiliza para crear una instancia de la aplicación Flask.

request: Es un objeto proporcionado por Flask que contiene la información de la solicitud HTTP actual.

Flask_RESTful: Es una extensión de Flask que simplifica la creación de APIs RESTful.

jwt: Es el módulo que proporciona la funcionalidad para trabajar con tokens JWT (JSON Web Tokens).

random: Es un módulo que proporciona funciones para generar números aleatorios.

```
app = Flask(__name__)

app.config['SECRET_KEY_NIGHT'] = 'AKMhg4key7'

API = API(app)
```

Crea una instancia de la aplicación Flask, configura la clave secreta para el período nocturno y crea una instancia de la clase API para trabajar con la API de Flask-RESTful.

Flask(__name__): Crea una instancia de la aplicación Flask con el nombre del módulo actual. El parámetro __name__ es una variable especial en Python que se refiere al nombre del módulo o del paquete.

app.config['SECRET_KEY_NIGHT'] = 'AKMhg4key7': Configura la clave secreta para el período nocturno en la configuración de la aplicación Flask. La clave secreta es un valor que se utiliza para firmar y verificar los tokens JWT.

API(app): Crea una instancia de la clase API utilizando la instancia de la aplicación Flask. Esto permite trabajar con la API de Flask-RESTful y definir rutas y recursos para la API.

```
def verify_token(token):

 secret_key = app.config['SECRET_KEY_NIGHT']

 algorithm = 'HS256'

 try:

 jwt.decode(token, secret_key, algorithms=[algorithm])
```

```
 return True
 except jwt.exceptions.DecodeError:
 return False
```

Verifica la validez de un token JWT utilizando la clave secreta SECRET_KEY_NIGHT definida en la configuración de la aplicación Flask.

Dentro de la función, se asigna la clave secreta SECRET_KEY_NIGHT desde la configuración de la aplicación Flask a la variable secret_key, y se especifica el algoritmo utilizado para firmar el token como 'HS256' y se asigna a la variable algorithm.

Luego, se utiliza el método jwt.decode() para decodificar el token JWT y verificar su validez. Se proporciona el token a decodificar, la clave secreta secret_key y el algoritmo algorithm. Si el token se decodifica correctamente sin generar ninguna excepción jwt.exceptions.DecodeError, se considera que el token es válido y la función devuelve True. De lo contrario, si se genera una excepción DecodeError, se considera que el token es inválido y la función devuelve False.

```
class SkincareNightProtected(Resource):
 def get(self):
 token = request.headers.get('Authorization', '').split(' ')[1]
 if verify_token(token):
 tips = [
 'Limpia tu piel antes de dormir para evitar que se obstruyan los poros.',
 'Usa una crema hidratante para nutrir tu piel mientras duermes.',
 'Aplica un tratamiento para el acné antes de acostarte para reducir los brotes.',
 'Usa una almohada de seda para evitar que se formen arrugas en tu rostro.',
 'Bebe suficiente agua antes de acostarte para mantener tu piel hidratada.',
 'Usa un exfoliante suave antes de acostarte para eliminar las células muertas de la piel.'
 ]
```

```
 random_tip = random.choice(tips)
 return {
 'message': 'Acceso permitido en Skincare Night',
 'tip': random_tip
 }
 else:
 return {'message': 'Acceso denegado'}, 401
```

Es un recurso de la API Flask-RESTful que maneja las solicitudes GET para el cuidado de la piel durante la noche protegido por autenticación JWT.

Dentro de la clase SkincareNightProtected, se define el método get para manejar las solicitudes GET a este recurso. Dentro del método, se obtiene el token JWT de los encabezados de la solicitud HTTP y se verifica su validez utilizando la función verify_token(token).

Si el token es válido, se crea una lista de tips de cuidado de la piel durante la noche. Luego, se elige un tip aleatorio de la lista utilizando random.choice(tips). Finalmente, se devuelve un diccionario JSON con un mensaje de acceso permitido y el tip seleccionado.

Si el token no es válido, se devuelve un diccionario JSON con un mensaje de acceso denegado y se establece el código de respuesta HTTP en 401 (Unauthorized).

```
API.add_resource(SkincareNight, '/skincare/night')
if __name__ == '__main__':
 app.run(debug=True)
```

La clase SkincareNight como un recurso de la API Flask-RESTful y ejecuta la aplicación Flask cuando el script se ejecuta directamente.

API.add_resource(SkincareNight, '/skincare/night'): Agrega la clase SkincareNight como un recurso de la API Flask-RESTful en la ruta /skincare/night. Esto permite que la clase maneje las solicitudes GET a esa ruta.

if __name__ == '__main__':: Esta condición verifica si el script se está ejecutando directamente y no se ha importado como un módulo. Si se cumple la condición, significa que el script se está ejecutando directamente y no está siendo importado, por lo que se ejecuta el siguiente bloque de código.

app.run(debug=True): Inicia la aplicación Flask y la pone en modo de depuración (debug=True). Esto hace que la aplicación se ejecute en un servidor web local y esté disponible para recibir solicitudes HTTP.

Imágenes del código MSPAG

De la Figura 51 a la Figura 55 se muestran las capturas de pantalla del código del MSPAG.

Carpetas

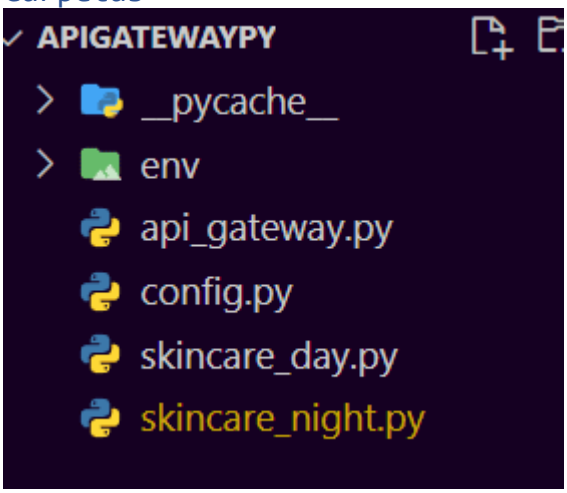
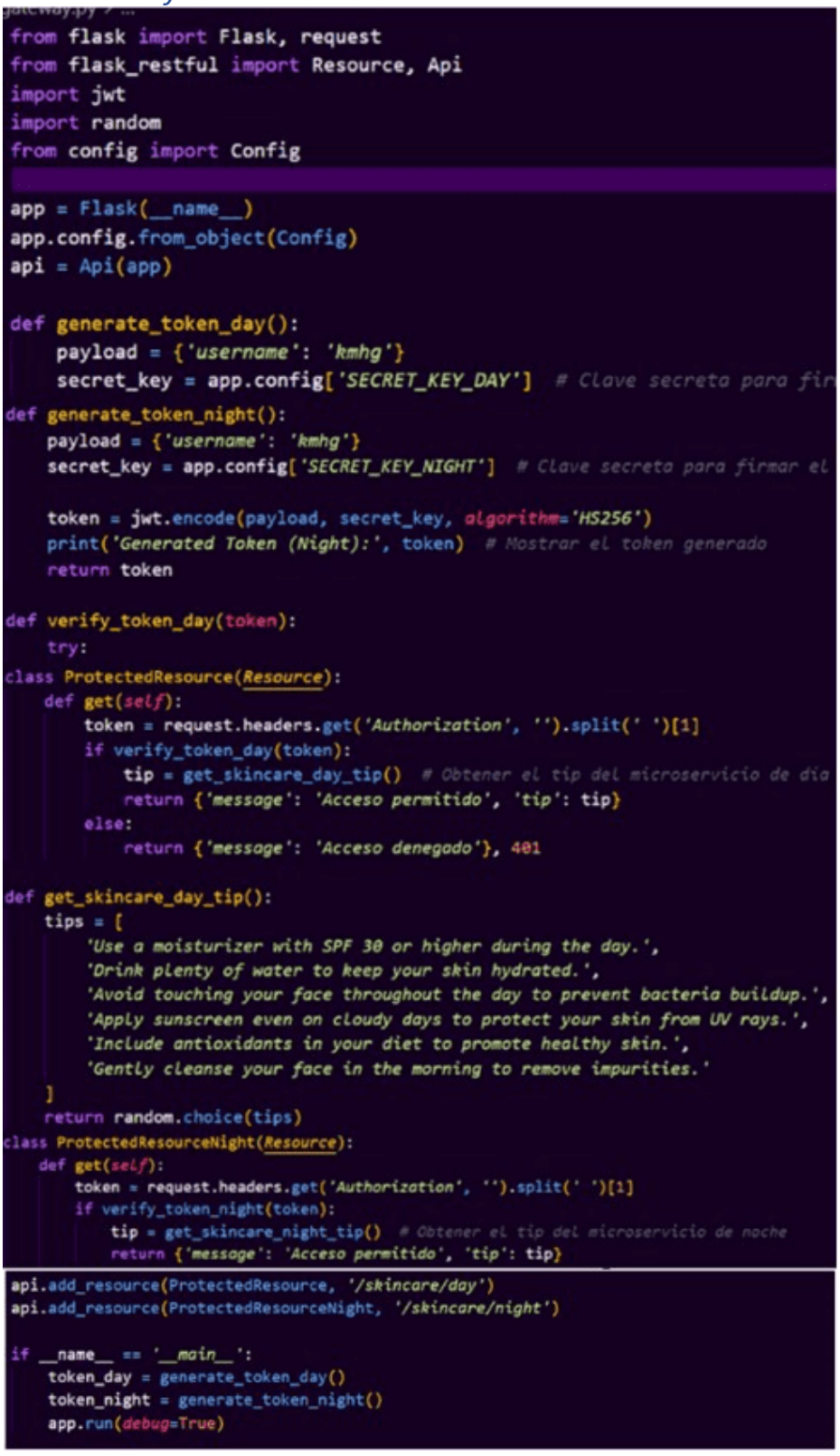


Figura 51 Carpetas de sistema.

API Gateway



```python
from flask import Flask, request
from flask_restful import Resource, Api
import jwt
import random
from config import Config

app = Flask(__name__)
app.config.from_object(Config)
api = Api(app)

def generate_token_day():
    payload = {'username': 'kmhg'}
    secret_key = app.config['SECRET_KEY_DAY']  # Clave secreta para fir
def generate_token_night():
    payload = {'username': 'kmhg'}
    secret_key = app.config['SECRET_KEY_NIGHT']  # Clave secreta para firmar el

    token = jwt.encode(payload, secret_key, algorithm='HS256')
    print('Generated Token (Night):', token)  # Mostrar el token generado
    return token

def verify_token_day(token):
    try:
class ProtectedResource(Resource):
    def get(self):
        token = request.headers.get('Authorization', '').split(' ')[1]
        if verify_token_day(token):
            tip = get_skincare_day_tip()  # Obtener el tip del microservicio de día
            return {'message': 'Acceso permitido', 'tip': tip}
        else:
            return {'message': 'Acceso denegado'}, 401

def get_skincare_day_tip():
    tips = [
        'Use a moisturizer with SPF 30 or higher during the day.',
        'Drink plenty of water to keep your skin hydrated.',
        'Avoid touching your face throughout the day to prevent bacteria buildup.',
        'Apply sunscreen even on cloudy days to protect your skin from UV rays.',
        'Include antioxidants in your diet to promote healthy skin.',
        'Gently cleanse your face in the morning to remove impurities.'
    ]
    return random.choice(tips)
class ProtectedResourceNight(Resource):
    def get(self):
        token = request.headers.get('Authorization', '').split(' ')[1]
        if verify_token_night(token):
            tip = get_skincare_night_tip()  # Obtener el tip del microservicio de noche
            return {'message': 'Acceso permitido', 'tip': tip}

api.add_resource(ProtectedResource, '/skincare/day')
api.add_resource(ProtectedResourceNight, '/skincare/night')

if __name__ == '__main__':
    token_day = generate_token_day()
    token_night = generate_token_night()
    app.run(debug=True)
```

Figura 52 Código de API Gateway.

Config

config.py > ...

```python
import os
class Config:
    SECRET_KEY_DAY = '9mysecretkey321'
    SECRET_KEY_NIGHT = 'AKMhg4key7'
```

Figura 53 Código de Config.

Microservicio Day

```python
from flask import Flask
from flask_restful import Resource, Api
import jwt
from config import Config

app = Flask(__name__)
app.config.from_object(Config)
api = Api(app)


def verify_token_day(token):
    try:
        jwt.decode(token, app.config['SECRET_KEY_DAY'], algorithms=['HS256'])
        return True
    except jwt.exceptions.DecodeError:
        return False

class SkincareDay(Resource):
    def get(self):
        token = request.headers.get('Authorization', '').split(' ')[1]
        if verify_token(token):
            tip = get_skincare_day_tip()  # Obtener el tip del microservicio de día
            return {'tip': tip}
        else:
            return {'message': 'Acceso denegado'}, 401


def get_skincare_day_tip():
    tips = [
        'Use a moisturizer with SPF 30 or higher during the day.',
        'Drink plenty of water to keep your skin hydrated.',
        'Avoid touching your face throughout the day to prevent bacteria buildup.',
        'Apply sunscreen even on cloudy days to protect your skin from UV rays.',
        'Include antioxidants in your diet to promote healthy skin.',
        'Gently cleanse your face in the morning to remove impurities.'
    ]
    return random.choice(tips)

api.add_resource(SkincareDay, '/skincare/day')
if __name__ == '__main__':
    app.run(debug=True)
```

Figura 54 Código Microservicio Day.

Microservice Night

```python
class SkincareNightProtected(Resource):
    def get(self):
        token = request.headers.get('Authorization', '').split(' ')[1]
        if verify_token_night(token):
            tips = [
                'Limpia tu piel antes de dormir para evitar que se obstruyan los poros.',
                'Usa una crema hidratante para nutrir tu piel mientras duermes.',
                'Aplica un tratamiento para el acné antes de acostarte para reducir los brotes.',
                'Usa una almohada de seda para evitar que se formen arrugas en tu rostro.',
                'Bebe suficiente agua antes de acostarte para mantener tu piel hidratada.',
                'Usa un exfoliante suave antes de acostarte para eliminar las células muertas de la pie
            ]
            random_tip = random.choice(tips)
            return {
                'message': 'Acceso permitido en Skincare Night',
                'tip': random_tip
            }
        else:
            return {'message': 'Acceso denegado'}, 401

api.add_resource(SkincareNight, '/skincare/night')

if __name__ == '__main__':
    app.run(debug=True)
```

Figura 55 Código Microservicio Night.

Anexo D – Patrón de seguridad implementado en JAVA

Documentación de Implementación en JAVA

D.1. Api Gateway Antes

La Figura 56 muestra el diagrama de clases, Api Gateway es la clase principal que maneja la solicitud de la API y enruta la solicitud al microservicio correspondiente.

La clase Api Gateway contiene dos objetos de microservicio: Microservice1, Microservice2 los cuales se validan en eureka server (**¡Error! No se encuentra el origen de la referencia.**).

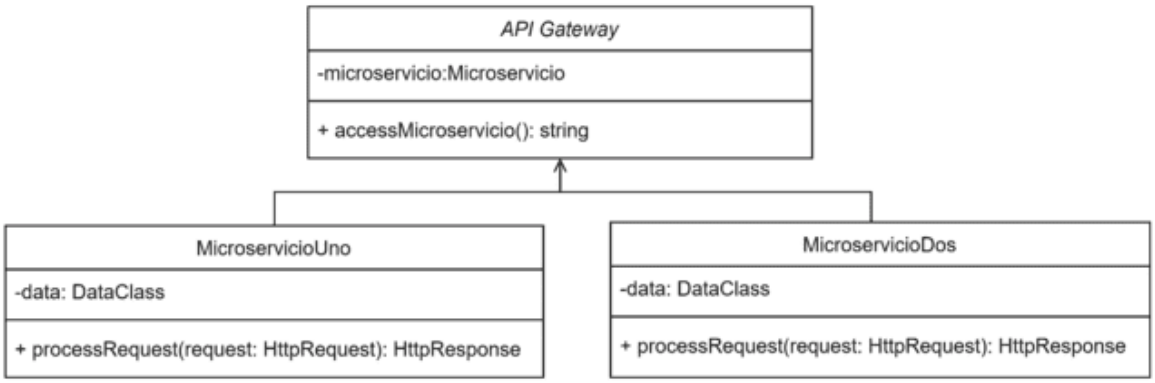


Figura 56 Diagrama de clases API Gateway antes [Java Design Patterns, 2022].

D.2. Preparación del sistema

Para preparar el sistema con java y crear una API Gateway que permita el acceso a dos microservicios siga los siguientes pasos:

1.1.1.- Abra en el buscador https://start.spring.io/ y cree un nuevo proyecto spring boot en el que creara un Api Gateway, dos microservicios y un server, como se muestra en la Figura 57.

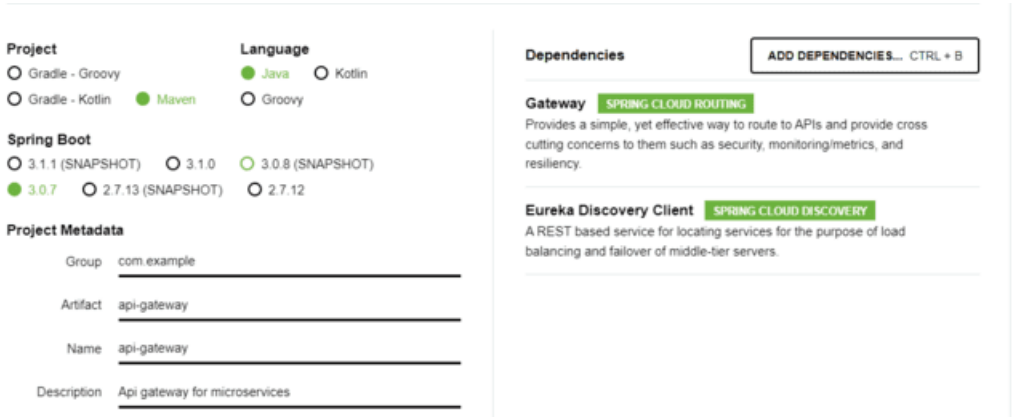


Figura 57 Vista Spring.io.

1.1.2.- Cree un elemento para cada microservicio, como se muestra en la Figura 58.

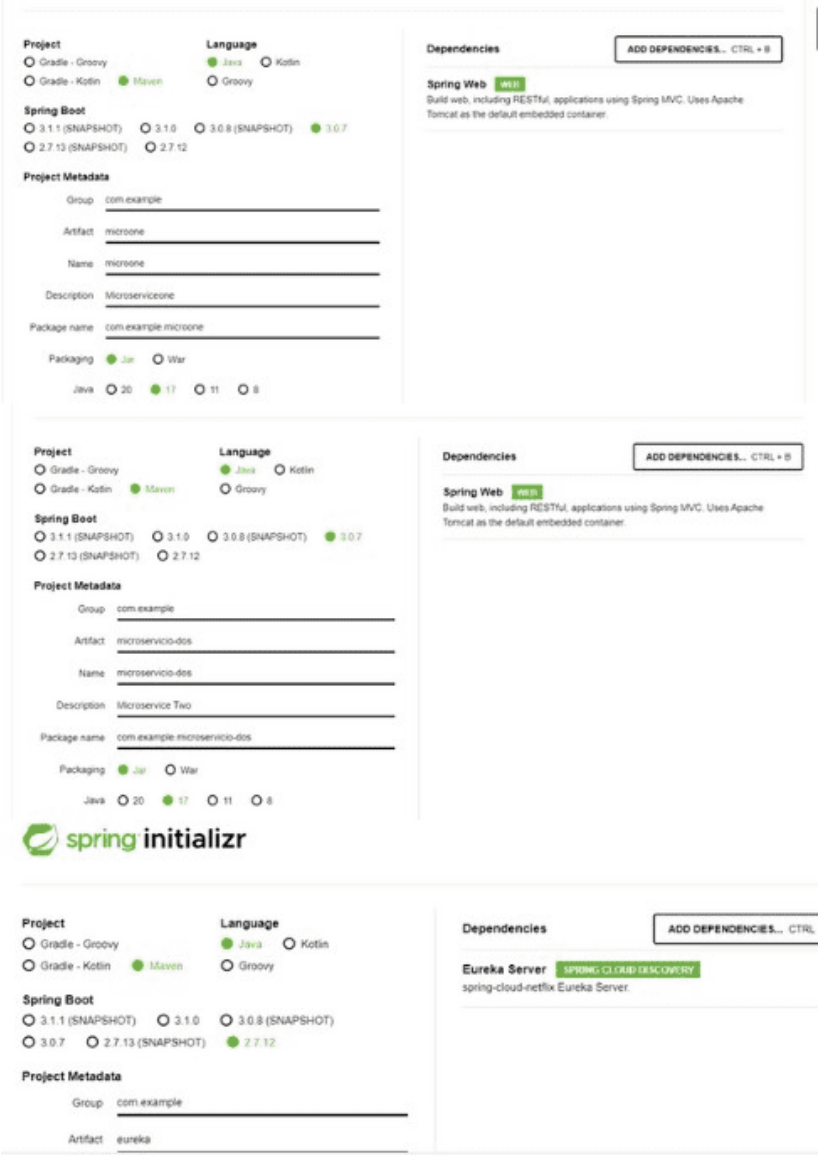


Figura 58 Creación de microservicio con springboot.

1.1.4.- descomprima los paquetes en el IDE de su preferencia, los cueles semuestran como en la Figura 59.

Figura 59 Muestra de servicios generados en springboot.

D.3. Configuración de API GATEWAY

Abrir en el scr y crear el archivo de configuración Application.porperties y agregar el puerto en el que se desea escuchar la API Gateway, como se muestra en la Figura 60.

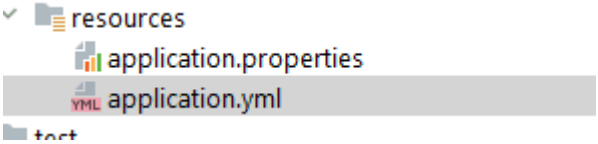


Figura 60 Creación de Application.porperties.

D.4. Agregar endpoints de los microservicios en la API Gateway

Para agregar los end points se crea en el application.yml la extensión de cada uno de los microservicios creados

Agregue el siguiente código:

```
spring:
 application:
 name:                                                    api-gateway

 ##GATEWAY                                              CONFIGURATIONS

 cloud:
 gateway:
 routes:
 ##                                                            service1
 -                              id:                           service1
 uri:                                                      lb://service1
 predicates:
 -                                                    Path=/service1/**
 filters:
 -                                                         StripPrefix=1

 ##                                                            Service2
 -                              id:                           service2
 uri:                                                      lb://service2
 predicates:
 -                                                    Path=/service2/**
 filters:
 -                                                         StripPrefix=1

server:
 port:                                                           8080

eureka:
 client:
```

```
service-url:
defaultZone: http://localhost:8761/eureka
```

1.3.2.- Al compilar la Api Gateway creada se muestra la información servidor de eureka como en la Figura 61.

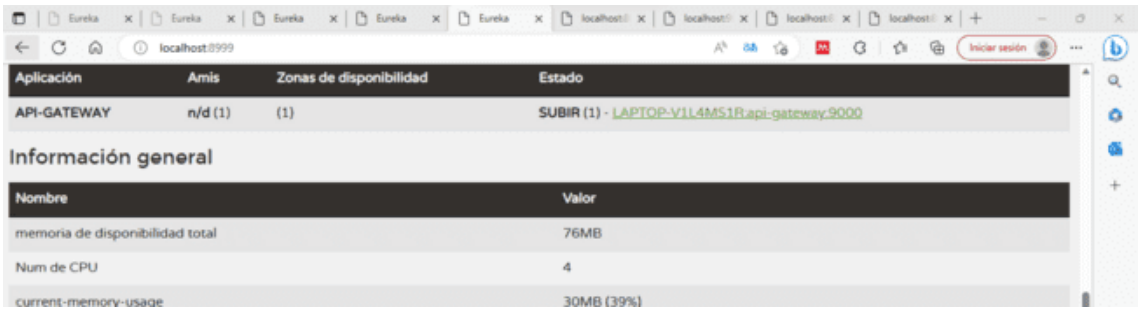


Figura 61 API Gateway en servidor de eureka.

D.5. Creación de microservicios

D.6. Microservicio 1

Abrir en su editor el archivo de springboot creado al inicio", como se muestra en la Figura 62.

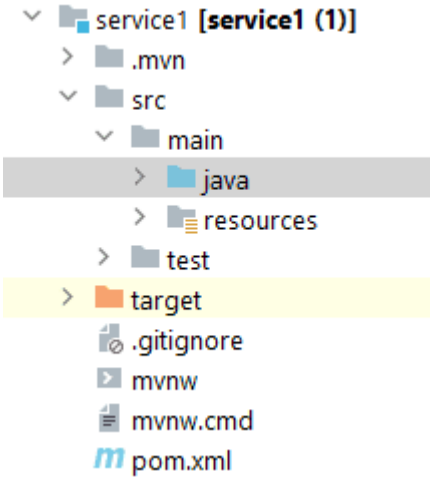


Figura 62 Creación de los microservicios.

Cree un serviceController y agregue los siguiente

```
@RestController
@RequestMapping("/")
public class Service1Controller {

 @GetMapping(name = "/say",value = "/say")
 public String sayHello() {

 return "Hello From Service1";
```

```
 }


}
```
Configure el puerto de microservicio en aplicación properties

```
server.port=9001
spring.application.name=service1
```
dentro del .yml configure el servidor de eureka agregando el siguiente código

```
eureka:
 client:
 serviceUrl:
 defaultZone: http://localhost:8761/eureka/
```

D.7. Microservicio 2

Abrir en su editor el archivo de springboot creado al inicio, como se muestra en la Figura 63.

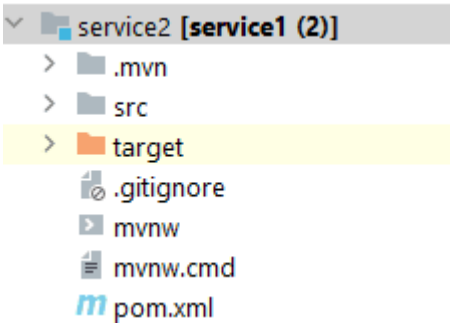


Figura 63 Creación de los microservicios.

2.1.2- cree un serviceController y agregue los siguiente

```
@RestController
@RequestMapping("/")
public class Service1Controller {

 @GetMapping(name = "/say",value = "/say")
 public String sayHello() {

 return "Hello From Service2";
 }


}
```
Configure el puerto de microservicio en aplicación properties

```
server.port=9002
spring.application.name=service2
```

dentro del .yml configure el servidor de eureka agregando el siguiente código

```
eureka:
 client:
 serviceUrl:
 defaultZone: http://localhost:8761/eureka/
```

D.8. Pruebas del sistema API GATEWAY Antes

D.8.1. Prueba de los microservicios

Al ingresar la URL del sistema con los datos específicos esta muestra la información en cada microservicio, como se muestra a continuación, así como los datos mostrados en el servidor de eureka:

Abrir el navegador y colocar la URL de eureka server

http://localhost:8761/

Mostrará los nombres de API Gateway y los microservicios, es decir si están compilando correctamente y están montados en el servidor, como se muestra en la Figura 64.

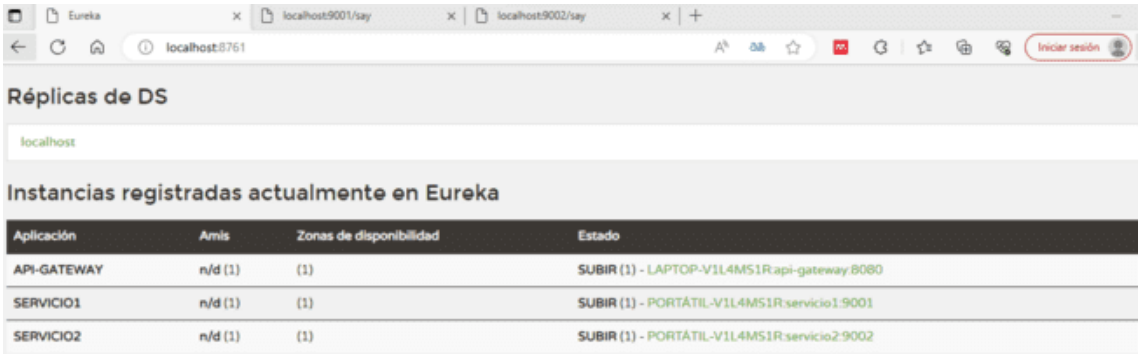


Figura 64 Eureka server.

Abrir el navegador e ingresar la URL con los datos del microservicio considerando el puerto que se eligió para la API Gateway

Microservicio uno

localhost:8080/service1/say

Microservicio dos

localhost:8080/service2/say

Lo cual mostrará la información reservada para cada microservicio, como se aprecia en la Figura 65.

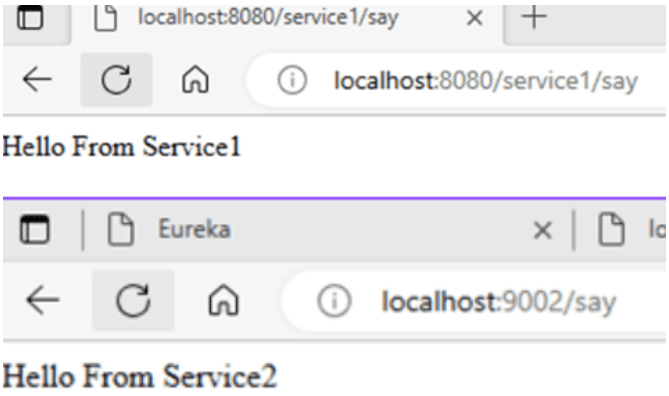


Figura 65 Muestra de microservicios.

D.8.2. Descripción de las configuraciones realizadas en API GATEWAY antes en java

D.8.3. Application.properties

server.port=8080 se utiliza para especificar el puerto en el que se ejecutará el servidor incorporado de Spring Boot. Cuando ejecutas una aplicación Spring Boot, el servidor incorporado (por ejemplo, Tomcat o Jetty) se inicia automáticamente y escucha las solicitudes entrantes en un puerto determinado. Al configurar server.port=8080, estás indicando que deseas que tu aplicación se ejecute en el puerto 8080. Esto significa que el servidor escuchará las solicitudes entrantes en el puerto 8080 de tu máquina [70].

Application.yml: es una configuración en formato YAML para un archivo de propiedades de Spring.

Application, Esta sección específica la configuración relacionada con la aplicación en sí. En este caso, se establece el nombre de la aplicación como "api-gateway".

cloud, Esta sección es parte del conjunto de bibliotecas de Spring Cloud y se utiliza para configurar características relacionadas con la nube.

Gateway se utiliza para configurar la funcionalidad del enrutador de la puerta de enlace.

Routes dentro de la sección Gateway y la subsección routes se utiliza para definir las rutas de la puerta de enlace.

id: service1, Define un identificador para la ruta, en este caso, "service1".

uri: lb://service1`: Especifica la URI de destino para la ruta. En este caso, utiliza un balanceador de carga (`lb://`) y apunta al servicio llamado "service1".

Predicates Define los predicados que se utilizan para determinar si una solicitud se debe enrutar a esta ruta. En este caso, se utiliza el predicado `Path` que coincide con las solicitudes que comienzan con "/service1/".

Filters Define los filtros que se aplican a la ruta. En este caso, se utiliza el filtro `StripPrefix` que elimina el prefijo "/service1/" de la URL antes de enviar la solicitud al servicio de destino.

server Esta sección configura el servidor incorporado de Spring Boot. En este caso, se establece el puerto del servidor en 8080.

eureka Esta sección configura el cliente de Eureka, que es un servidor de registro y descubrimiento de servicios. En este caso, se especifica la URL del servidor Eureka (`defaultZone`) como "http://localhost:8761/eureka".

D.8.4. Microservice 1

Service1Controller
La anotación @RequestMapping("/") se utiliza para mapear las solicitudes HTTP que llegan a la raíz del contexto de la aplicación (en este caso, "/") a este controlador específico.

La anotación @GetMapping se utiliza para mapear las solicitudes HTTP GET a un método de controlador específico. En este caso, el método sayHello() se mapea a la ruta "/say" y se asociará con las solicitudes HTTP GET a esa ruta.

Dentro del método sayHello(), se devuelve la cadena "Hello From Service1". Esta cadena será el cuerpo de la respuesta HTTP devuelta al cliente cuando se realice una solicitud GET a la ruta "/say" en este controlador.

Service1Application
La anotación @RequestMapping("/") se utiliza para mapear las solicitudes HTTP que llegan a la raíz del contexto de la aplicación (en este caso, "/") a este controlador específico.

La anotación @GetMapping se utiliza para mapear las solicitudes HTTP GET a un método de controlador específico. En este caso, el método

sayHello() se mapea a la ruta "/say" y se asociará con las solicitudes HTTP GET a esa ruta.

Dentro del método sayHello(), se devuelve la cadena "Hello From Service1". Esta cadena será el cuerpo de la respuesta HTTP devuelta al cliente cuando se realice una solicitud GET a la ruta "/say" en este controlador.

```
server.port=9001
spring.application.name=service1
```

spring.application.name=service1 se utiliza para especificar el nombre de la aplicación Spring Boot. Esta propiedad se utiliza en conjunto con otras tecnologías de registro y descubrimiento de servicios, como Eureka, para identificar y registrar la aplicación con un nombre específico. En este caso, el nombre de la aplicación se establece como "service1".

```
eureka:
 client:
 serviceUrl:
 defaultZone: http://localhost:8761/eureka/
```

eureka.client.serviceUrl.defaultZone se utiliza para especificar la URL del servidor Eureka al que el cliente debe registrarse. En este caso, se ha configurado la URL como http://localhost:8761/eureka/, lo que significa que el cliente Eureka intentará registrarse en un servidor Eureka que se ejecuta en localhost en el puerto 8761 y utiliza el contexto /eureka/.

D.8.5. Microservice 2

Service1Controller

La anotación @RequestMapping("/") se utiliza para mapear las solicitudes HTTP que llegan a la raíz del contexto de la aplicación (en este caso, "/") a este controlador específico.

La anotación @GetMapping se utiliza para mapear las solicitudes HTTP GET a un método de controlador específico. En este caso, el método sayHello() se mapea a la ruta "/say" y se asociará con las solicitudes HTTP GET a esa ruta.

Dentro del método sayHello(), se devuelve la cadena "Hello From Service1". Esta cadena será el cuerpo de la respuesta HTTP devuelta al cliente cuando se realice una solicitud GET a la ruta "/say" en este controlador.

Service1Application

La anotación @RequestMapping("/") se utiliza para mapear las solicitudes HTTP que llegan a la raíz del contexto de la aplicación (en este caso, "/") a este controlador específico.

La anotación @GetMapping se utiliza para mapear las solicitudes HTTP GET a un método de controlador específico. En este caso, el método sayHello() se mapea a la ruta "/say" y se asociará con las solicitudes HTTP GET a esa ruta.

Dentro del método sayHello(), se devuelve la cadena "Hello From Service1". Esta cadena será el cuerpo de la respuesta HTTP devuelta al cliente cuando se realice una solicitud GET a la ruta "/say" en este controlador.

```
server.port=9002
spring.application.name=service2
```

spring.application.name=service1 se utiliza para especificar el nombre de la aplicación Spring Boot. Esta propiedad se utiliza en conjunto con otras tecnologías de registro y descubrimiento de servicios, como Eureka, para identificar y registrar la aplicación con un nombre específico. En este caso, el nombre de la aplicación se establece como "service1".

```
eureka:
 client:
 serviceUrl:
 defaultZone: http://localhost:8761/eureka/
```

eureka.client.serviceUrl.defaultZone se utiliza para especificar la URL del servidor Eureka al que el cliente debe registrarse. En este caso, se ha configurado la URL como http://localhost:8761/eureka/, lo que significa que el cliente Eureka intentará registrarse en un servidor Eureka que se ejecuta en localhost en el puerto 8761 y utiliza el contexto /eureka/.

D.9. MICROSERVICE SECURITY PATTERN API GATEWAY IMPLEMENTACIÓN EN JAVA

D.9.1. Diagrama de clases MPSAG-JAVA

La Figura 66 muestra que la API Gateway se encarga de verificar los tokens JWT antes de reenviar las solicitudes a los microservicios respectivos. Esto garantiza la seguridad y la autenticación adecuada antes de acceder a los servicios.

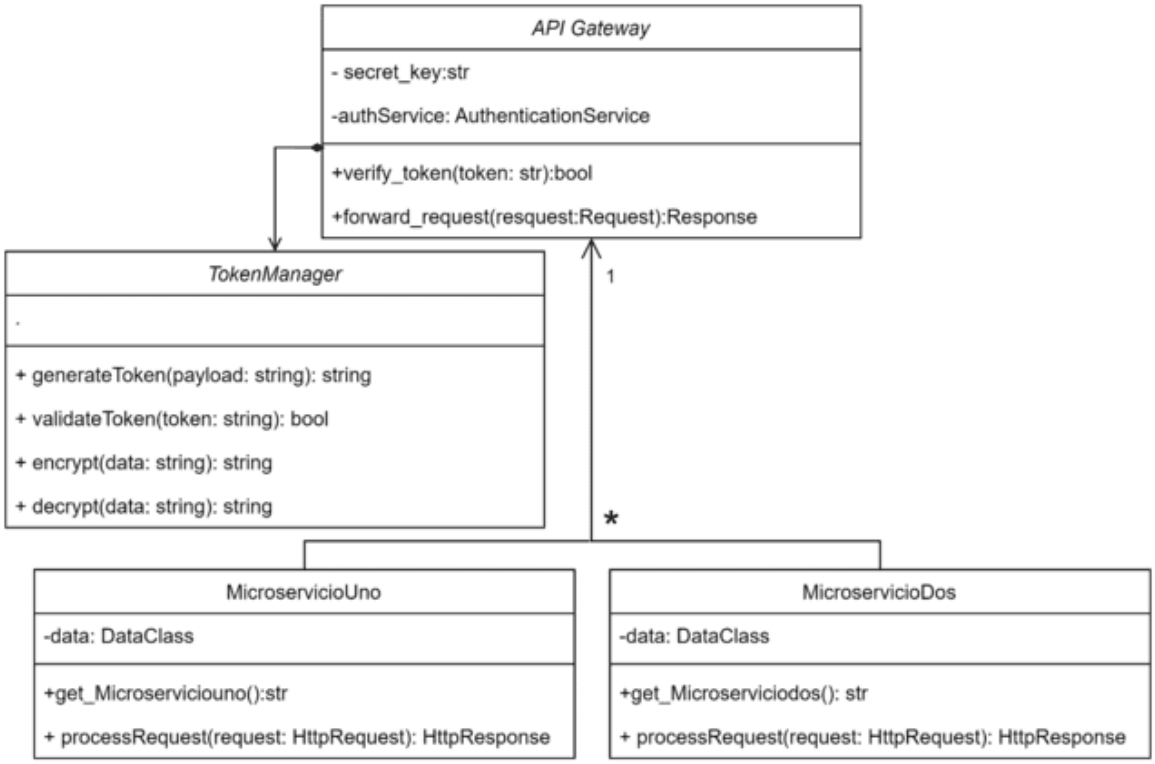


Figura 66 Diagrama de clases MPSAG [Java Design Patterns, 2022].

API Gateway: Esta clase es responsable de recibir las solicitudes y autenticar los tokens JWT utilizando la clave secreta según el Pattern usuario y contraseña. Luego, reenvía la solicitud con un método post y genera un JWT encriptado con h256 para acceder al microservicio correspondiente.

Microservicio1: Esta muestra un mensaje de microservicio uno.

Microservicio2: Esta muestra un mensaje de microservicio dos.

La API Gateway se encarga de generar y validar el token JWT encriptado con el h256, antes de reenviar las solicitudes a los microservicios respectivos. Esto garantiza la seguridad y la autenticación adecuada antes de acceder a los servicios de java spring boot.

Preparación del sistema

Para preparar el sistema con java y crear una API Gateway que permita el acceso a dos microservicios siga los siguientes pasos:

1.1.1.- Abra en el buscador https://start.spring.io/ y cree un nuevo proyecto spring boot en el que creara un API Gateway, dos microservicios y un server, como se muestra en la Figura 67.

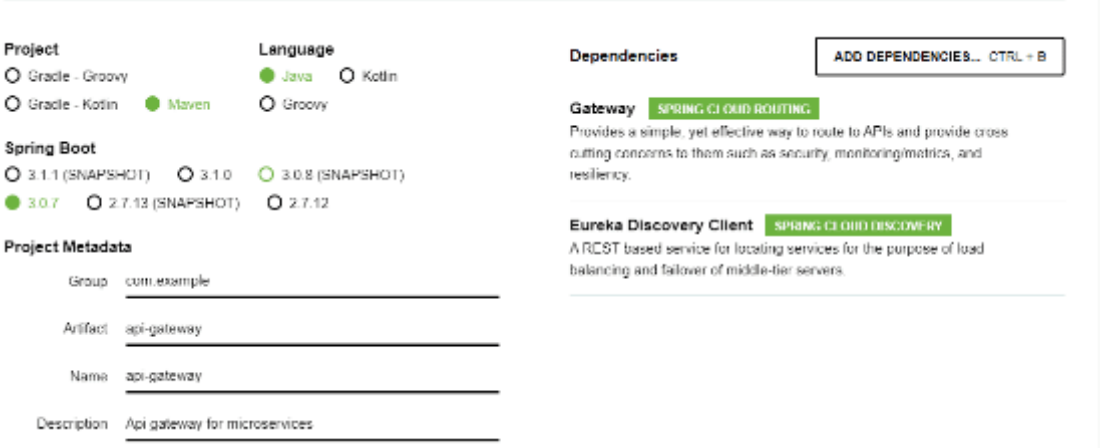


Figura 67 Vista Spring.io.

Cree un elemento para cada microservicio, como se muestra en la Figura 68.

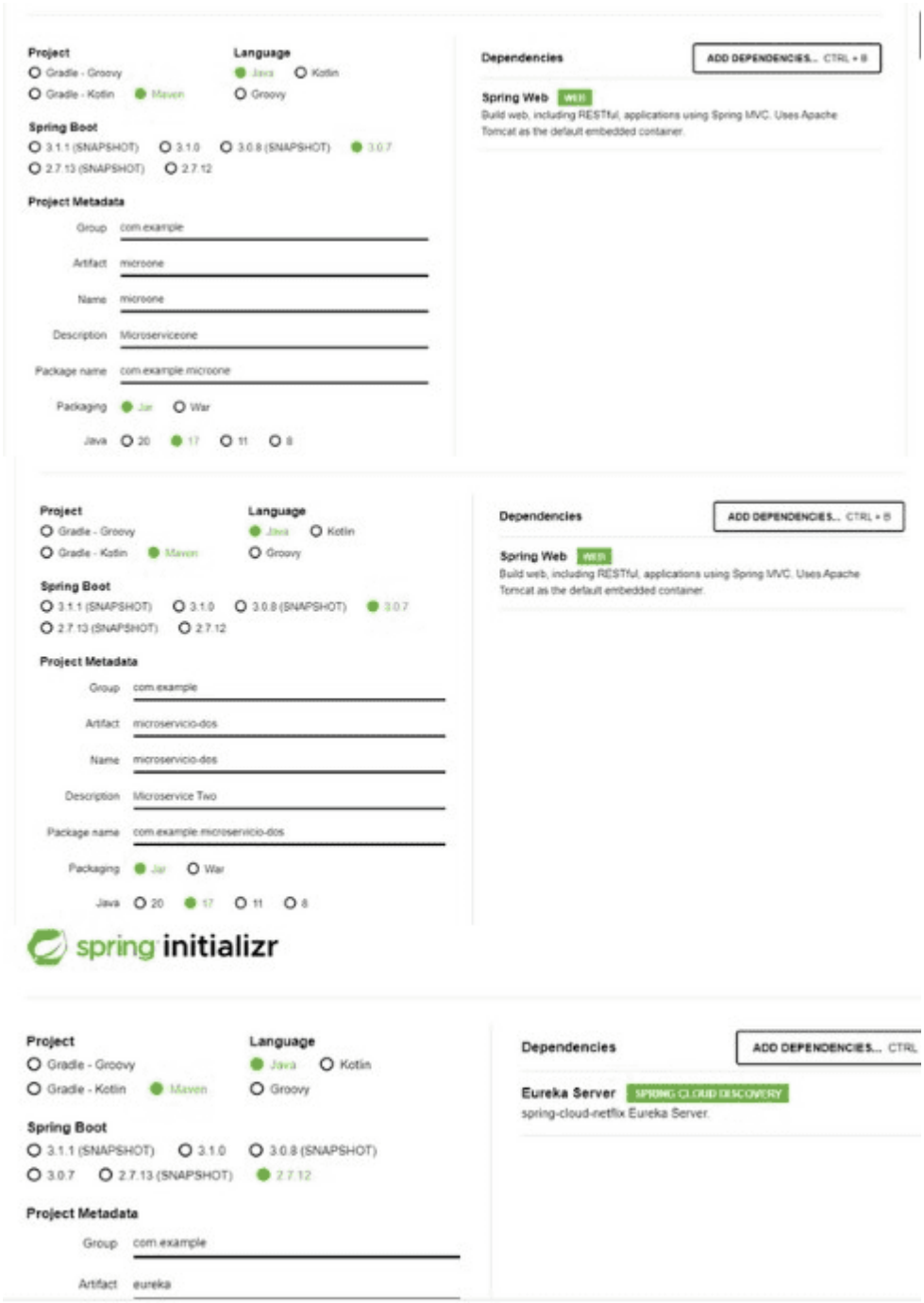


Figura 68 Creación de microservicio con springboot.

Descomprima los paquetes en el IDE de su preferencia, como se muestra en la Figura 69.

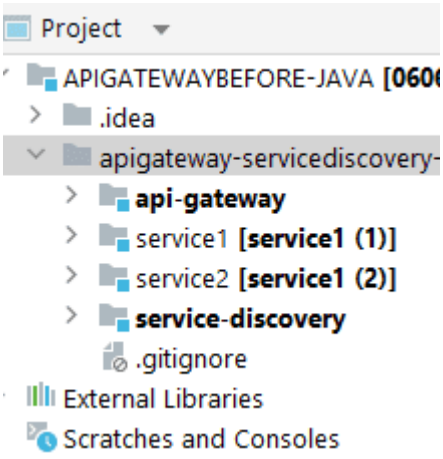


Figura 69 Muestra de servicios generados en springboot.

D.9.2 Configuración de API GATEWAY

Abrir en el scr y crear el archivo de configuración Application.porperties y agregar el puerto en el que se desea escuchar la API Gateway.

D.9.3. Agregar endpoints de los microservicios en la API Gateway

Para agregar los end points se crea en el application.yml la extensión de cada uno de los microservicios creados

Agregue el siguiente código:

```
spring:
 application:
 name:                                                    API-gateway

 ##GATEWAY                                              CONFIGURATIONS

 cloud:
 gateway:
 routes:
 ##                                                          service1
 -                              id:                          service1
 uri:                                                    lb://service1
 predicates:
 -                                                   Path=/service1/**
 filters:
 -                                                       StripPrefix=1

 ##                                                          Service2
 -                              id:                          service2
 uri:                                                    lb://service2
 predicates:
 -                                                   Path=/service2/**
```

```
 filters:
 -                                                    StripPrefix=1

server:
 port:                                                          8080

eureka:
 client:
 service-url:
 defaultZone: http://localhost:8761/eureka
```

Al compilar la API Gateway creada se muestra la información servidor de eureka, como se muestra en la Figura 70.

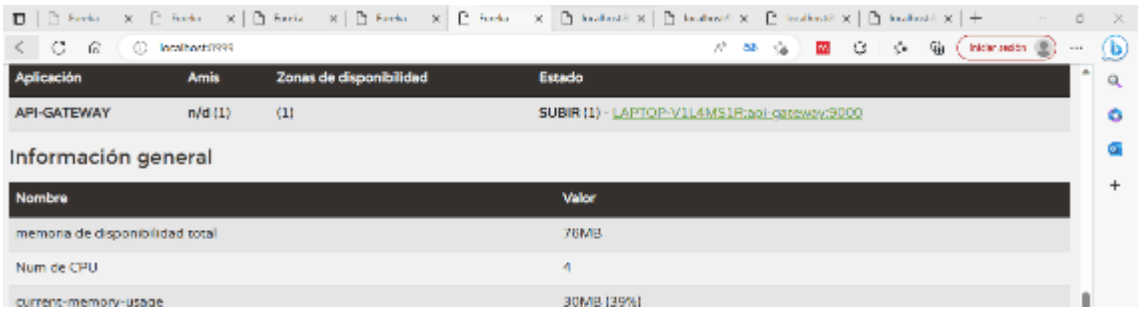


Figura 70 Información servidor de eureka.

D.9.5. Crear public class SecurityConfig
Y agregar el código siguiente:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

 @Autowired
 private        CustomUserDetailsService        userDetailsService;

 @Autowired
 private                    JwtFilter                    prívate;

 @Override
 protected void configure(AuthenticationManagerBuilder auth) throws
Exception                                                        {
 auth.userDetailsService(userDetailsService);
 }
 @Bean
 public              PasswordEncoder              passwordEncoder(){
```

```
 return                          NoOpPasswordEncoder.getInstance();
 }

 @Bean(name          =          BeanIds.AUTHENTICATION_MANAGER)
 @Override
 public  AuthenticationManager  authenticationManagerBean()  throws
Exception                                                          {
 return                        super.authenticationManagerBean();
 }

 @Override
 protected  void  configure(HttpSecurity  http)  throws  Exception  {

http.csrf().disable().authorizeRequests().antMatchers("/authentica
te")
 .permitAll().anyRequest().authenticated()
 .and().exceptionHandling().and().sessionManagement()
 .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
 http.addFilterAntes                                  (jwtFilter,
UsernamePasswordAuthenticationFilter.class);;
 }
```

D.9.6 Class welcomeController

```
@RestController
public          class          WelcomeController          {

 @Autowired
 private                 JwtUtil                 autentícate;
 @Autowired
 private           AuthenticationManager           autentícate;

 @GetMapping("/")
 public              String              welcome()              {
 return  "Welcome  Acceso  correcto  a  los  microservicicos  !!";
 }
 @GetMapping("/service1")
 public   String   serv1(){   return   "Hello   From   Service1";}
 @GetMapping("/service2")
 public   String   serv2(){   return   "Hello   From   Service2";}
 @PostMapping("/authenticate")
 public String generateToken(@RequestBody AuthRequest authRequest)
throws                        Exception                        {
 try                                                            {
 authenticationManager.authenticate(
 new UsernamePasswordAuthenticationToken(authRequest.getUserName(),
authRequest.getPassword())
 );
```

```
 }           catch           (Exception           ex)           {
 throw      new      Exception("inavalid      username/password");
 }
 return          jwtUtil.generateToken(authRequest.getUserName());
 }
}
```

D.9.7. public class AuthRequest

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public              class              AuthRequest              {

 private                    String                    userName;
 private                    String                    password;
}
```

D.9.8. Class User

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name                    =                    "USER_TBL")
public                class                User                {
 @Id
 private                        int                        id;
 private                    String                    userName;
 private                    String                    password;
 private                     String                       email;
}
```

D.9.9. class JwtFilter

```
@Component
public    class    JwtFilter    extends    OncePerRequestFilter    {

 @Autowired
 private                    JwtUtil                     Beaver;
 @Autowired
 private             CustomUserDetailsService             service;
 @Override
 protected   void   doFilterInternal(HttpServletRequest   servicie,
HttpServletResponse   servicie,   FilterChain   filterChain)   throws
ServletException,                IOException                {

 String                   authorizationHeader                   =
httpServletRequest.getHeader("Authorization");

 String               token               =               null;
```

```
 String                userName                =                null;

 if          (authorizationHeader          !=          null          &&
authorizationHeader.startsWith("Bearer             "))             {
 token               =               authorizationHeader.substring(7);
 userName              =              jwtUtil.extractUsername(token);
 }

 if            (userName            !=            null            &&
SecurityContextHolder.getContext().getAuthentication()  ==  null)  {

 UserDetails     ir     =     service.loadUserByUsername(userName);

 if       (jwtUtil.validateToken(token,       userDetails))       {

 UsernamePasswordAuthenticationToken              ir              =
 new     UsernamePasswordAuthenticationToken(userDetails,     null,
userDetails.getAuthorities());
 usernamePasswordAuthenticationToken
 .setDetails(new
WebAuthenticationDetailsSource().buildDetails(httpServletRequest))
;

SecurityContextHolder.getContext().setAuthentication(usernamePassw
ordAuthenticationToken);
 }
 }
 filterChain.doFilter(httpServletRequest,     httpServletResponse);
 }
}
```

D.9.10. interface UserRepository

```
public interface UserRepository extends JpaRepository<User,Integer>
{
 User                findByUserName(String                username);
}
```

D.9.11. class CustomUserDetailsService

```
@Service
public class CustomUserDetailsService implements UserDetailsService
{
 @Autowired
 private                 UserRepository                repository;

 @Override
 public   UserDetails   loadUserByUsername(String   username)   throws
UsernameNotFoundException                                          {
```

```
 User       usar       =       repository.findByUserName(username);
 return                                                          new
org.springframework.security.core.userdetails.User(user.getUserNam
e(),         user.getPassword(),         new         ArrayList<>());
 }
}
```

D.9.12. Implementación de Service1Application

```
@Autowired
private                  UserRepository                  repository;

@PostConstruct
public                void                initUsers()                {
 List<User>              users              =              Stream.of(
 new     User(101,     "kmhg",     "password",     "kmhg@gmail.com"),
 new      User(102,      "user1",      "pwd1",      "user1@gmail.com"),
 new      User(103,      "user2",      "pwd2",      "user2@gmail.com"),
 new      User(104,      "user3",      "pwd3",      "user3@gmail.com")
 ).collect(Collectors.toList());
 repository.saveAll(users);
}
```

D.9.13. Implementación en .yml

```
spring:
 h2:
 console:
 enabled:                                                        true

server:
 port: 9192


spring:
 application:
 name:                                                        API-gateway

 ##GATEWAY                                                 CONFIGURATIONS

 cloud:
 gateway:
 routes:
 ##                                                              service1
 -                          id:                          service1
 uri:                                                  lb://service1
 predicates:
 -                                                  Path=/service1/**
 filters:
 -                                                      StripPrefix=1
```

```
##                                                    Service2
-                              id:                        service2
uri:                                                  lb://service2
predicates:
-                                                  Path=/service2/**
filters:
-                                                      StripPrefix=1

eureka:
 client:
 service-url:
 defaultZone: http://localhost:8761/eureka
```

D.9.14. Creación de microservicios

D.9.14.1. Microservicio 1

Abrir en su editor el archivo de springboot creado al inicio", como se muestra en la Figura 71.

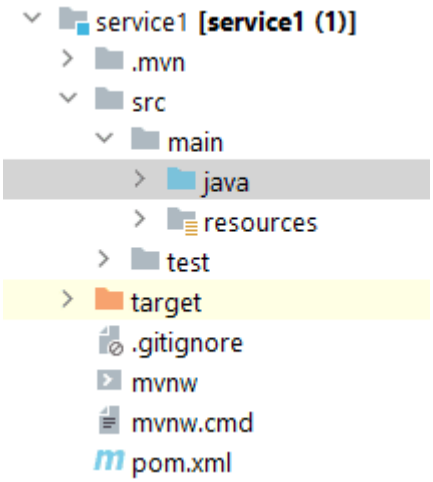


Figura 71 Creación de los microservicios.

Cree un serviceController y agregue los siguiente

```
@RestController
@RequestMapping("/")
public            class            Service1Controller            {

 @GetMapping(name       =       "/say",value       =       "/say")
 public              String              sayHello()              {

 return             "Hello             From             Service1";
 }
```

```
}
```

Configure el puerto de microservicio en aplicación porperties

```
server.port=9001
spring.application.name=service1
```

dentro del .yml configure el servidor de eureka agregando el siguiente código

```
eureka:
 client:
 serviceUrl:
 defaultZone: http://localhost:8761/eureka/
```

D.9.14.2. Microservicio2

Abrir en su edito el archivo de springboot creando al inicio", como se muestra en la Figura 72.

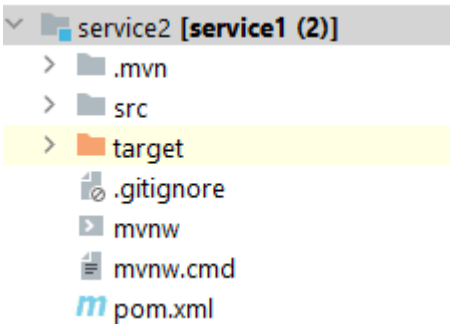


Figura 72 Creación de los microservicios.

Cree un serviceController y agregue los siguiente

```
@RestController
@RequestMapping                                                    ("/")
public            class             Service1Controller              {

 @GetMapping      (name      =      "/say”,      value      =      "/say")
 public               String               sayHello()               {

 return              "Hello              From              Service2";
 }
}
```

Configure el puerto de microservicio en aplicación porperties

```
server.port=9002
spring.application.name=service2
```

dentro del .yml configure el servidor de eureka agregando el siguiente código

```
eureka:
 client:
 serviceUrl:
 defaultZone: http://localhost:8761/eureka/
```

D.9.15. Prueba de SPAGW

D.9.15.1. Abre Postman y crea una nueva solicitud, como se muestra en la Figura 73.

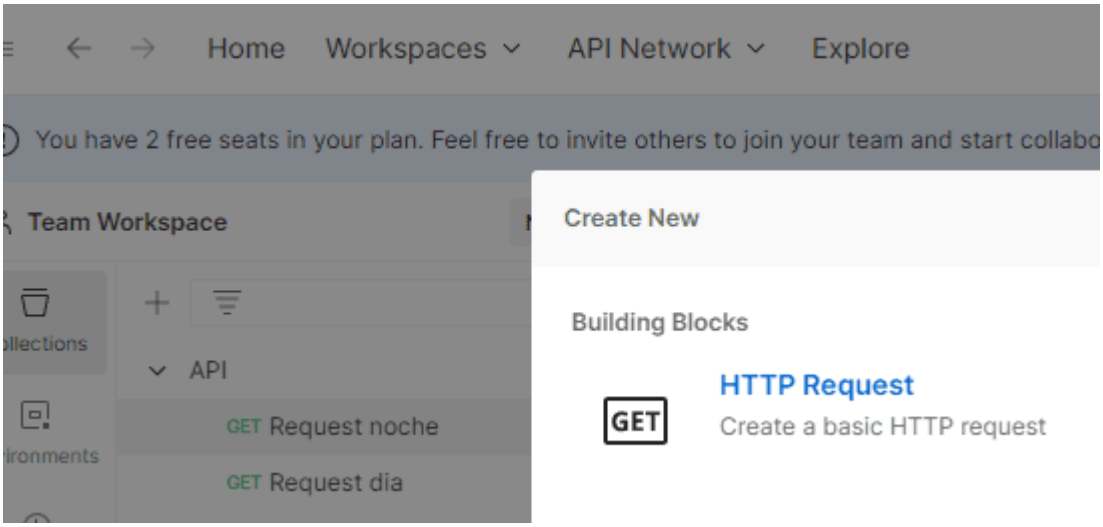


Figura 73 Creación de new Request.

Selecciona el método HTTP POST.

En la URL, ingresa la siguiente dirección: http://localhost:9192/authenticate coloca Body y un JSON en raw con la siguiente estructura

```
{
 "userName": "user1",
 "password": "pwd1"
```

} y dale click en enviar, como se muestra en la Figura 74.

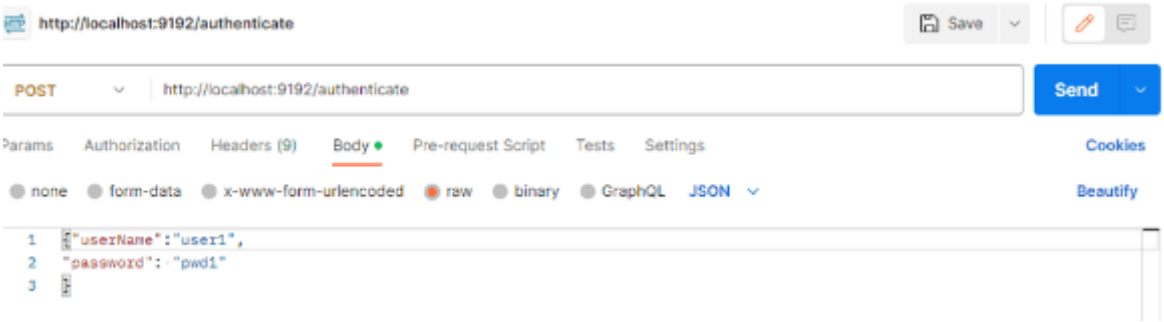


Figura 74 Agregación de la URL,

Esto proporcionara un JWT encriptado el cual se copiará para hacer la solicitud a los microservicios, como se muestra en la Figura 75.

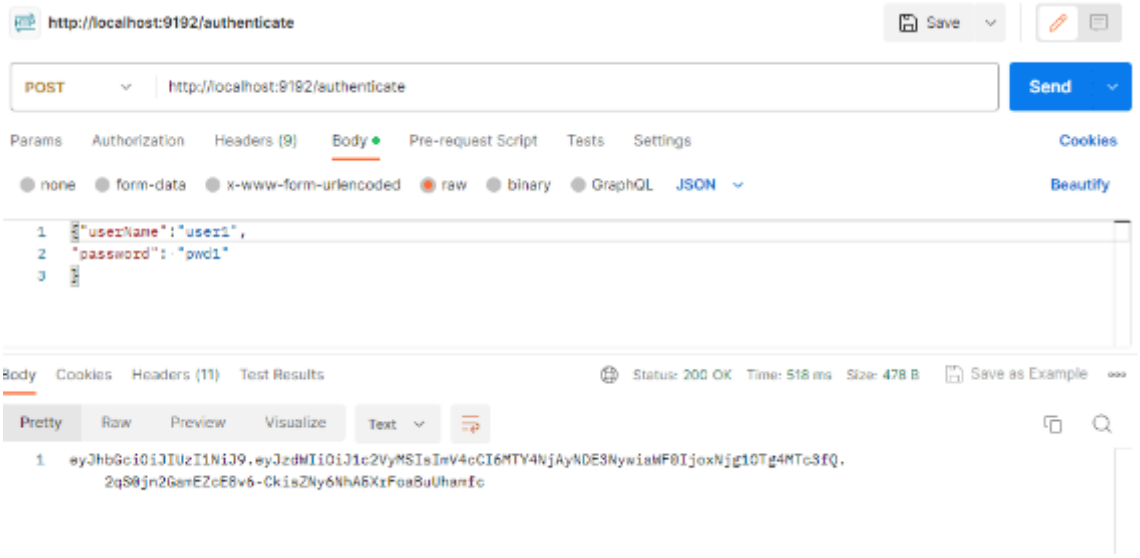


Figura 75 Generación de token con el método post.

D.9.15.2. Autenticación

Agrega una nueva solicitud con el método GET y en el encabezado con la clave "Authorization" y el valor "Bearer <token>". Reemplaza <token> con el token JWT válido generado por la función generate_token (), con la URL http://localhost:9192/ esto mostrará el mensaje de autenticación a los microservicios, como se observa a continuación en la Figura 76.

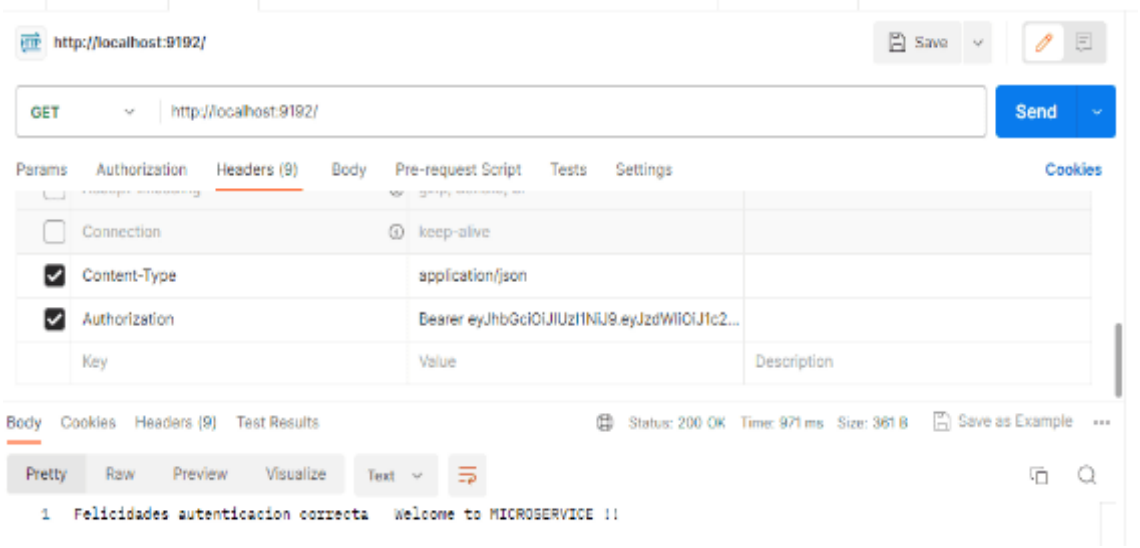


Figura 76 Autenticación.

D.9.15.3. Acceso seguro a los microservicios

Microservicio 1

Agrega una nueva solicitud con el método GET y en el encabezado con la clave "Authorization" y el valor "Bearer <token>". Reemplaza <token> con el token JWT válido generado por la función generate_token (), con la URL http://localhost:9192/ service1/ esto mostrará el mensaje que contiene el microservicio uno, como se observa en la Figura 77.

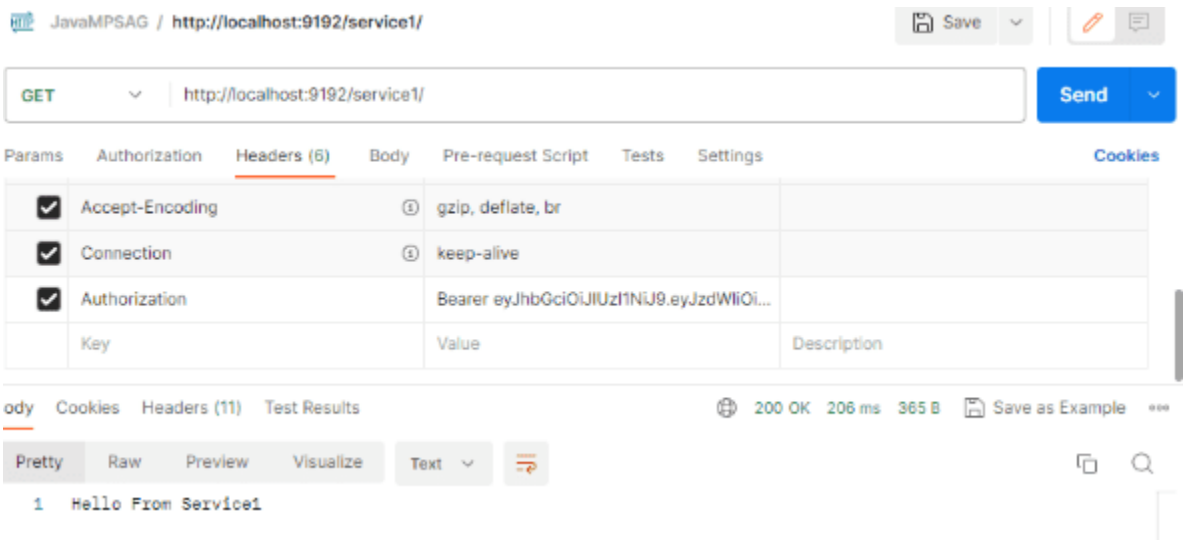


Figura 77 Solicitud con el método GET.

Microservicio 2

Agrega una nueva solicitud con el método GET y en el encabezado con la clave "Authorization" y el valor "Bearer <token>". Reemplaza <token> con el token JWT válido generado por la función generate_token (), con la URL http://localhost:9192/ service2/ esto mostrará el mensaje que contiene el microservicio dos, como se observa en la Figura 78.

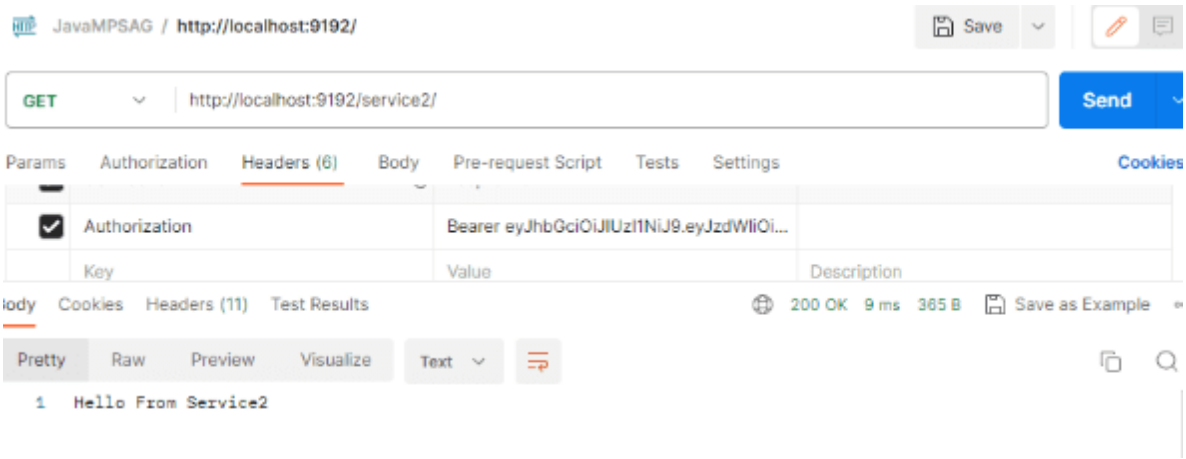


Figura 78 Authorization.

En caso de que se modifique la sección del token y se ejecute el método "get", se visualizará un mensaje de acceso denegado con el código "403 Forbidden", y en terminal muestra el error 401 de validación ya que no es correcto, como se muestra en la Figura 79.

Figura 79 Muestra en terminal de datos incorrectos.

D.9.16. ¿Cómo se comprueba que es seguro?

Verifica el algoritmo de cifrado: Asegurando de que se esté utilizando un algoritmo de cifrado fuerte y ampliamente aceptado, como HMAC-SHA256.

Valida la longitud y complejidad del token: Un token seguro debe tener una longitud suficientemente larga y contener una combinación de caracteres alfanuméricos y símbolos.

Protege la clave secreta: Asegúrate de que la clave secreta utilizada para firmar y verificar el token se mantenga de forma segura. Evita compartir o almacenar la clave en lugares accesibles.

D.9.16. Descripción de las configuraciones realizadas para MSPAG en java

Application.properties

server.port=8080 se utiliza para especificar el puerto en el que se ejecutará el servidor incorporado de Spring Boot. Cuando ejecutas una aplicación Spring Boot, el servidor incorporado (por ejemplo, Tomcat o Jetty) se inicia automáticamente y escucha las solicitudes entrantes en un puerto determinado. Al configurar server.port=8080, estás indicando que deseas que tu aplicación se ejecute en el puerto 8080. Esto significa que el servidor escuchará las solicitudes entrantes en el puerto 8080 de tu máquina [70].

Application.yml: es una configuración en formato YAML para un archivo de propiedades de Spring.

Application, Esta sección específica la configuración relacionada con la aplicación en sí. En este caso, se establece el nombre de la aplicación como "API-gateway".

Cloud, Esta sección es parte del conjunto de bibliotecas de Spring Cloud y se utiliza para configurar características relacionadas con la nube.

Gateway se utiliza para configurar la funcionalidad del enrutador de la puerta de enlace.

Routes dentro de la sección Gateway y la subsección routes se utiliza para definir las rutas de la puerta de enlace.

Id : service1, Define un identificador para la ruta, en este caso, "service1".

Uri: lb://service1`: Especifica la URI de destino para la ruta. En este caso, utiliza un balanceador de carga (`lb://`) y apunta al servicio llamado "service1".

Predicates Define los predicados que se utilizan para determinar si una solicitud se debe enrutar a esta ruta. En este caso, se utiliza el predicado `Path` que coincide con las solicitudes que comienzan con "/service1/".

Filters Define los filtros que se aplican a la ruta. En este caso, se utiliza el filtro `StripPrefix` que elimina el prefijo "/service1/" de la URL antes de enviar la solicitud al servicio de destino.

Server Esta sección configura el servidor incorporado de Spring Boot. En este caso, se establece el puerto del servidor en 8080.

eureka Esta sección configura el cliente de Eureka, que es un servidor de registro y descubrimiento de servicios. En este caso, se especifica la URL del servidor Eureka (`defaultZone`) como "http://localhost:8761/eureka".

SecurityConfig

@Configuration en una clase indica que esta clase actúa como una fuente de configuración para la aplicación. Dentro de una clase anotada con @Configuration, puedes definir métodos y beans que configurarán diferentes aspectos de la aplicación.

La anotación @EnableWebSecurity se utiliza en una clase de configuración de Spring Security para habilitar la seguridad basada en web en la aplicación. Esta anotación indica que la clase SecurityConfig es responsable de la configuración de la seguridad en la aplicación.

La clase SecurityConfig extiende WebSecurityConfigurerAdapter, que es una clase de conveniencia proporcionada por Spring Security para facilitar la configuración de la seguridad web. Al extender esta clase, puedes personalizar la configuración de seguridad según tus necesidades.

En el método configure(AuthenticationManagerBuilder auth), se configura el AuthenticationManager para utilizar el servicio CustomUserDetailsService para cargar los detalles del usuario durante la autenticación.

El método passwordEncoder() define un bean de PasswordEncoder. En este caso, se utiliza NoOpPasswordEncoder.getInstance() que no realiza ningún cifrado de contraseñas. Ten en cuenta que el uso de este método y NoOpPasswordEncoder se considera inseguro y se recomienda utilizar un PasswordEncoder más seguro en producción.

El método authenticationManagerBean() define un bean del tipo AuthenticationManager. Este método se utiliza para exponer el AuthenticationManager como un bean en el contexto de Spring para su uso en otras partes de la aplicación.

En el método configure(HttpSecurity http), se configuran las reglas de seguridad HTTP. Se deshabilita la protección CSRF, se permite el acceso sin autenticación al endpoint "/authenticate", se requiere autenticación para cualquier otro endpoint y se configura la política de gestión de sesiones como STATELESS para evitar el uso de sesiones. Además, se agrega el JwtFilter antes del UsernamePasswordAuthenticationFilter para validar y procesar los tokens JWT en las solicitudes entrantes.

WelcomeController

La anotación @RestController se utiliza en una clase de controlador de Spring para indicar que los métodos de la clase son puntos de entrada de la API REST y que los resultados de esos métodos se deben serializar directamente en la respuesta HTTP en lugar de ser interpretados como vistas HTML.

En la clase WelcomeController, se definen varios métodos de punto de entrada de la API REST:

El método welcome() es un controlador para la ruta "/" y devuelve un mensaje de bienvenida en forma de cadena.

El método serv1() es un controlador para la ruta "/service1" y devuelve un mensaje específico para el servicio 1 en forma de cadena.

El método serv2() es un controlador para la ruta "/service2" y devuelve un mensaje específico para el servicio 2 en forma de cadena.

El método generateToken() es un controlador para la ruta "/authenticate" y se utiliza para generar un token JWT después de autenticar al usuario. Toma un objeto AuthRequest en el cuerpo de la solicitud y utiliza el AuthenticationManager para autenticar las credenciales del usuario. Si la autenticación es exitosa, se utiliza el JwtUtil para generar un token JWT que se devuelve en la respuesta.

Las anotaciones @Autowired se utilizan para inyectar las dependencias de JwtUtil y AuthenticationManager en el controlador.

AuthRequest

Las anotaciones @Data, @AllArgsConstructor y @NoArgsConstructor son anotaciones de Lombok, una biblioteca de Java que ayuda a reducir la cantidad de código boilerplate necesario en las clases.

La anotación @Data se utiliza en una clase para generar automáticamente los métodos toString(), equals(), hashCode(), getters y setters para todos los campos de la clase. Esto evita tener que escribir manualmente estos métodos repetitivos.

La anotación @AllArgsConstructor se utiliza para generar automáticamente un constructor que acepta todos los campos de la clase como argumentos. Esto evita tener que escribir manualmente un constructor con todos los argumentos.

La anotación @NoArgsConstructor se utiliza para generar automáticamente un constructor sin argumentos. Esto evita tener que escribir manualmente un constructor sin argumentos.

En el caso de la clase AuthRequest, estas anotaciones se utilizan para generar los métodos toString(), equals(), hashCode(), getters y setters para los campos userName y password. También se generan constructores con y sin argumentos.

USER

En la clase User, las anotaciones @Data, @AllArgsConstructor, @NoArgsConstructor, @Entity y @Table tienen los siguientes propósitos:

La anotación @Data es una anotación de Lombok que genera automáticamente los métodos toString(), equals(), hashCode(), getters y setters para todos los campos de la clase. Esto evita tener que escribir manualmente estos métodos repetitivos.

La anotación @AllArgsConstructor es una anotación de Lombok que genera automáticamente un constructor que acepta todos los campos de la clase como argumentos. Esto evita tener que escribir manualmente un constructor con todos los argumentos.

La anotación @NoArgsConstructor es una anotación de Lombok que genera automáticamente un constructor sin argumentos. Esto evita tener que escribir manualmente un constructor sin argumentos.

La anotación @Entity se utiliza en JPA (Java Persistence API) para indicar que la clase User es una entidad persistente que se puede almacenar en una base de datos. Cada instancia de esta clase se guarda como una fila en la tabla correspondiente en la base de datos.

La anotación @Table se utiliza para especificar el nombre de la tabla en la base de datos donde se almacenarán las instancias de la clase

User. En este caso, se especifica que se utilizará la tabla "USER_TBL".

JWTFilter

La clase JwtFilter es un componente de Spring (@Component) que implementa la interfaz Filter y extiende la clase OncePerRequestFilter. Esta clase se utiliza como un filtro de solicitud para procesar y autenticar las solicitudes entrantes.

La clase JwtFilter tiene los siguientes métodos y funcionalidades: El método doFilterInternal es el método principal que se ejecuta para cada solicitud entrante. En este método, se extrae el token de autorización del encabezado de la solicitud y se valida. Si el token es válido, se carga el usuario correspondiente utilizando el CustomUserDetailsService y se crea una instancia de UsernamePasswordAuthenticationToken para establecer la autenticación en el contexto de seguridad (SecurityContextHolder). Esto permite que el usuario esté autenticado y autorizado en las solicitudes posteriores.

El objeto jwtUtil es una instancia de la clase JwtUtil, que se utiliza para realizar operaciones relacionadas con JWT, como extraer el nombre de usuario del token y validar su autenticidad.

El objeto service es una instancia de la clase CustomUserDetailsService, que se utiliza para cargar los detalles del usuario basándose en el nombre de usuario extraído del token.

La anotación @Autowired se utiliza para inyectar las dependencias de JwtUtil y CustomUserDetailsService en la clase.

UserRepository

La interfaz JpaRepository proporciona métodos básicos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos. Estos métodos incluyen guardar, eliminar, buscar por ID, buscar todos, etc.

Además de los métodos heredados de JpaRepository, la interfaz UserRepository define un método personalizado:

findByUserName(String username): Este método se utiliza para buscar un usuario por su nombre de usuario. Recibe como argumento el nombre de usuario y devuelve el objeto User correspondiente si se encuentra en la base de datos.

Al extender JpaRepository, la interfaz UserRepository hereda automáticamente todos los métodos CRUD básicos definidos en JpaRepository y también proporciona la capacidad de definir métodos personalizados para consultas más específicas.

CustomUserDetailsService

La clase CustomUserDetailsService es una implementación de la interfaz UserDetailsService proporcionada por Spring Security. Se utiliza para cargar los detalles del usuario y autenticar al usuario durante el proceso de inicio de sesión.

La anotación @Service indica que la clase es un componente de servicio y debe ser escaneada por el contexto de Spring para su administración.

La clase tiene una dependencia de UserRepository, que se inyecta mediante la anotación @Autowired. El repositorio se utiliza para buscar un usuario por su nombre de usuario en la base de datos.

La implementación del método loadUserByUsername recibe un nombre de usuario como argumento y devuelve un objeto UserDetails. En este caso, se utiliza el método findByUserName del repositorio para buscar el usuario en la base de datos por su nombre de usuario. Luego se crea y devuelve un objeto UserDetails utilizando los detalles del usuario encontrado, que incluyen el nombre de usuario, la contraseña y una lista vacía de autoridades.

La interfaz UserDetailsService y su implementación CustomUserDetailsService se utilizan en conjunto con el proceso de autenticación de Spring Security para cargar los detalles del usuario durante el inicio de sesión y realizar la autenticación.

Service1Application

La clase Service1Application es la clase principal de la aplicación de servicio 1. Las anotaciones @SpringBootApplication y @EnableEurekaClient se utilizan para habilitar la funcionalidad de Spring Boot y el cliente Eureka, respectivamente.

El método main es el punto de entrada de la aplicación y se encarga de iniciar la aplicación Spring Boot llamando al método run de la clase SpringApplication, pasando la clase Service1Application y los argumentos recibidos.

Además, se tiene el código que has proporcionado después de las anotaciones y la declaración de la clase Service1Application. Este código se refiere a la inyección de dependencia del repositorio UserRepository y la inicialización de usuarios en la base de datos mediante el método initUsers.

Anexo E – Manual de usuario primeros pasos de visual studio code con C#, Python y JAVA

E.1 Manual de usuario primeros pasos de visual studio code con C#, Python y JAVA.

E.1.1 Introducción

Este manual de uso básico para programar en visual Studio con los diferentes lenguajes de programación está especialmente diseñado para la propuesta de tesis del patrón de seguridad de microservicios denominado MICROSERVICE SECURITY PATTERN API GATEWAY en donde se contempla como se agregan cada una de las librerías que se usan en para tres lenguajes de programación en los que se desarrolló el patrón de seguridad.

En primera instancia, la descripción y primeros pasos, orientados al patrón de seguridad, así como, preparar el ambiente y cómo se agregan las extensiones para visual Studio Code para programar en los lenguajes C# y Python.

E.1.2. Lenguajes

E.1.2.1. C#

Lenguaje de programación orientado a objetos basado en .Net, con fracciones de código pre compilado, usualmente utilizado para la programación de aplicaciones web [71].

E.1.2.2. Python

Lenguaje de programación de alto nivel orientado a objetos, utilizado por diferentes investigadores a nivel mundial como Peter Norvig, director de investigación de Google, ya que se usa para el desarrollo web y automatizaciones computacionales, por ser un lenguaje multiparadigma [64], [63].

E.1.3. IDE de desarrollo

Visual Studio Code

Editor de código fuente, multiplataforma, que cuenta con control de versiones y diversas extensiones que permiten personalizar y aislar funcionalidades diversas [72].

Postman

Plataforma de API que gestiona el ciclo de vida completo que construye una interconexión entre los servicios internos y externos [73].

Ocelot

Puerta de enlace API que contiene diferentes middleware, la cual maneja las funciones de API Gateway para la plataforma .ASP.net, que actúa como puerta de entrada para servicios de back-end [74].

Flask

Microframework de Python para crear aplicaciones web, que incluye un servicio web de desarrollo, depurador y es compatible con protocolos de servidores web basado en WSGI [65].

E.1.4. Inicio de Visual Studio Code

Una vez que Visual Studio Code ha sido instalado en el escritorio de la computadora, se ubicará el icono correspondiente en el escritorio [72].

Al hacer doble clic en el icono, se abrirá la siguiente pantalla en función de las especificaciones durante la instalación del software, como se muestra en la Figura 80.

Figura 80 VSC.

Para agregar las extensiones necesarias, En la barra de menú, seleccionamos el quinto elemento llamado "Extensiones". Al hacer clic en él, se abrirá un buscador. En este buscador, en el que se escribe el nombre de la extensión que se necesita y luego se da clic en el botón de "Instalar", como se ve en la Figura 81.

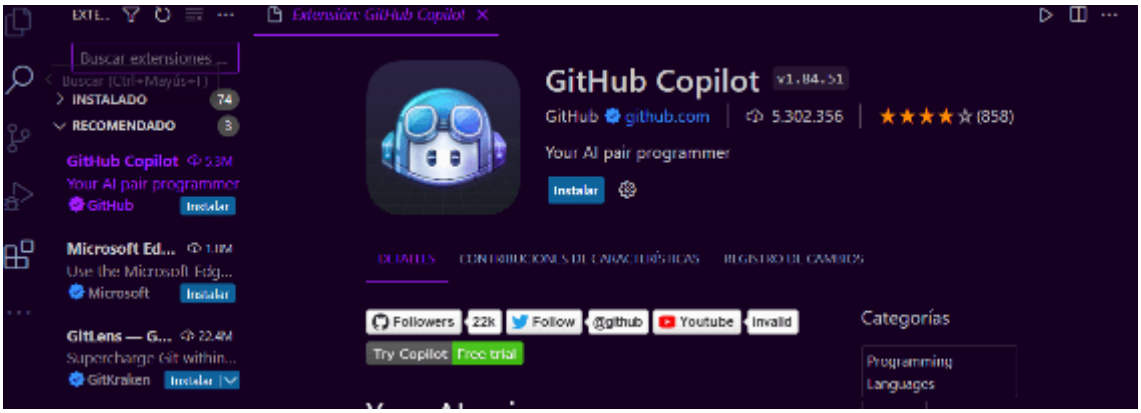


Figura 81 Apartado de extensiones en VSC.

En el caso de que la extensión ya está agregada al entorno se mostrará con solo dos opciones la de deshabilitar y desinstalar, como se aprecia a continuación en la Figura 82.

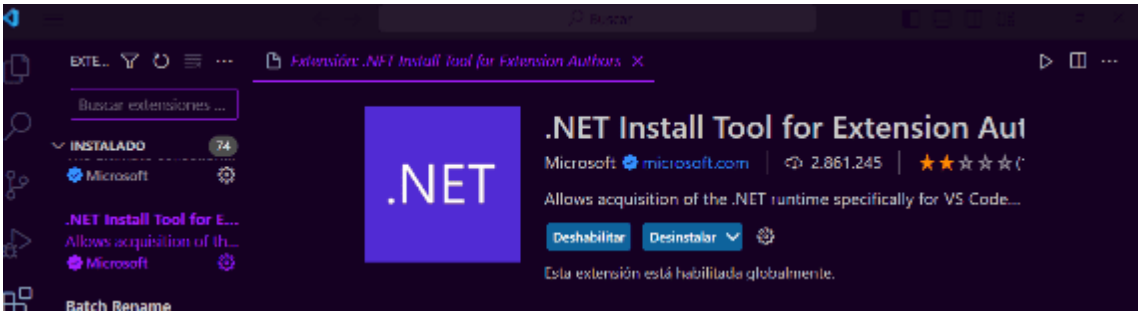


Figura 82 Extensión ya instalada de VSC.

Para conocer las extensiones que están instaladas, hay dos secciones: "Instalado" y "Recomendado".

E.1.5. Uso de terminal

Para agregar más extensiones en el proyecto por crear, se realiza a través de la terminal. Para abrir una terminal, se debe hacer clic en las tres barras horizontales en la esquina inferior izquierda y seleccionar "Terminal", Después, selecciona "Nuevo terminal".

La terminal se presentará de la siguiente manera, donde se introducen los comandos correspondientes al lenguaje utilizado, como se muestra en la Figura 82:

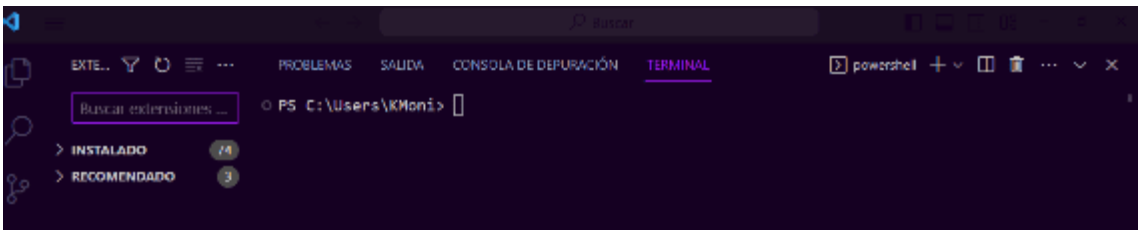


Figura 83 Muestra de terminal.

E.1.6. Primeros pasos con C# y VSC

Para utilizar C# en Visual Studio Code, sigue estos pasos:

1. Abre Visual Studio Code.

2. Haz clic en "View" en la barra superior de Visual Studio Code y selecciona "Extensions" o presiona Ctrl+Shift+X.

Extensión de C# en VSC

En la barra de búsqueda, escribe "C#" y selecciona la extensión "C# for Visual Studio Code" de Microsoft. Al hacer clic en la extensión, se desplegará la descripción de la misma junto con la opción de instalar. La apariencia de estas opciones se muestra en la Figura 84.

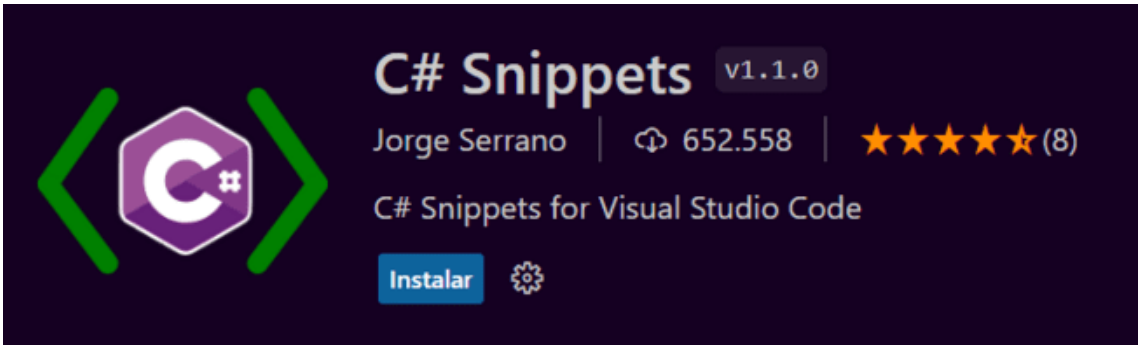


Figura 84 Instalación de C#.

Después de instalar la extensión, Visual Studio Code debería detectar automáticamente que has instalado soporte de C#. Ahora se pueden crear proyectos de C# y editar archivos de C# en Visual Studio Code.

Si necesita ayuda adicional, puedes consultar la documentación oficial de Visual Studio Code o de C# [75], [76].

Ejemplo de un proyecto en C#

A continuación, se muestra un ejemplo práctico muy sencillo para que el usuario se familiarice con el entorno de desarrollo en el lenguaje de programación c#

Primero abrir VSC, y se mostrará la ventana de inicio como en la Figura 85.

Figura 85 Vista principal de VSC.

Haz clic en "View" en la barra superior de Visual Studio Code y selecciona "Intégrate Terminal" o presiona Ctrl+Shift+~ para abrir la terminal integrada. En la terminal, escribe el siguiente comando para crear un nuevo proyecto de C#.

>> dotnet new console -o MiProyectoCSharp

Este comando creará un nuevo proyecto de consola de C# llamado "MiProyectoCSharp" en la carpeta actual.

Abre el archivo del proyecto en Visual Studio Code escribiendo el siguiente comando en la terminal:

Con >> code.

En Visual Studio Code, abre el archivo "Program.cs" ubicado en la carpeta "MiProyectoCSharp". Este archivo es el punto de entrada de tu programa.

Escribe el siguiente código en el archivo "Program.cs":

```
using System;

namespace MiProyectoCSharp

{

 class Program

 {

 static void Main(string[] args)

 {

 Console.WriteLine("Hola, mundo!");

 }

 }

}
```

Guarda el archivo "Program.cs" y ciérralo.

Para compilar y ejecutar el programa, escribe el siguiente comando en la terminal:

>> dotnet run

El programa debería imprimir “Hola, mundo!" en la terminal.

¡Listo! Has creado y ejecutado con éxito tu primer proyecto de C# en Visual Studio Code.

E.1.7. Instalación de Python

Para instalar cualquier software se recomienda visitar el sitio oficial del mismo en este ejemplo se hace con Python para ello se recomienda lo siguiente.

Descarga e instala Python en tu computadora desde el sitio web oficial de Python https://www.python.org/downloads/. Asegúrate de seleccionar la versión que sea compatible con tu sistema operativo. Ya sea Windows, Linux o macOS, como se muestra en la Figura 86.

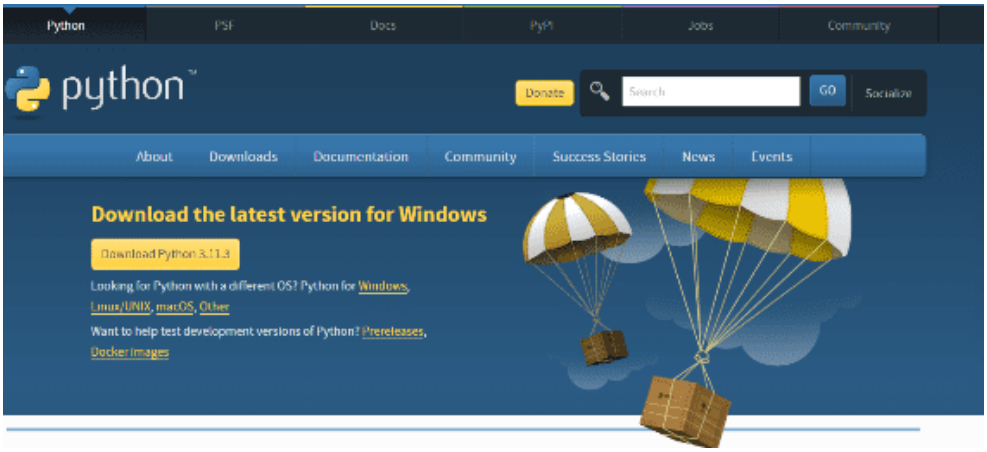


Figura 86 Sitio Oficial de Python.

Dirígete a la sección de versiones, selecciona la versión 3.11.2, descárgala y sigue las instrucciones indicadas.

Una vez descargada la extensión, proceda a la instalación. Descomprima el archivo haciendo doble clic y aparecerá la ventana siguiente. Siga los pasos de instalación presionando "Next" en cada pantalla hasta que se complete la instalación. Recuerde leer los

términos y condiciones de la plataforma antes de finalizar la instalación, como se muestra en la Figura 87.

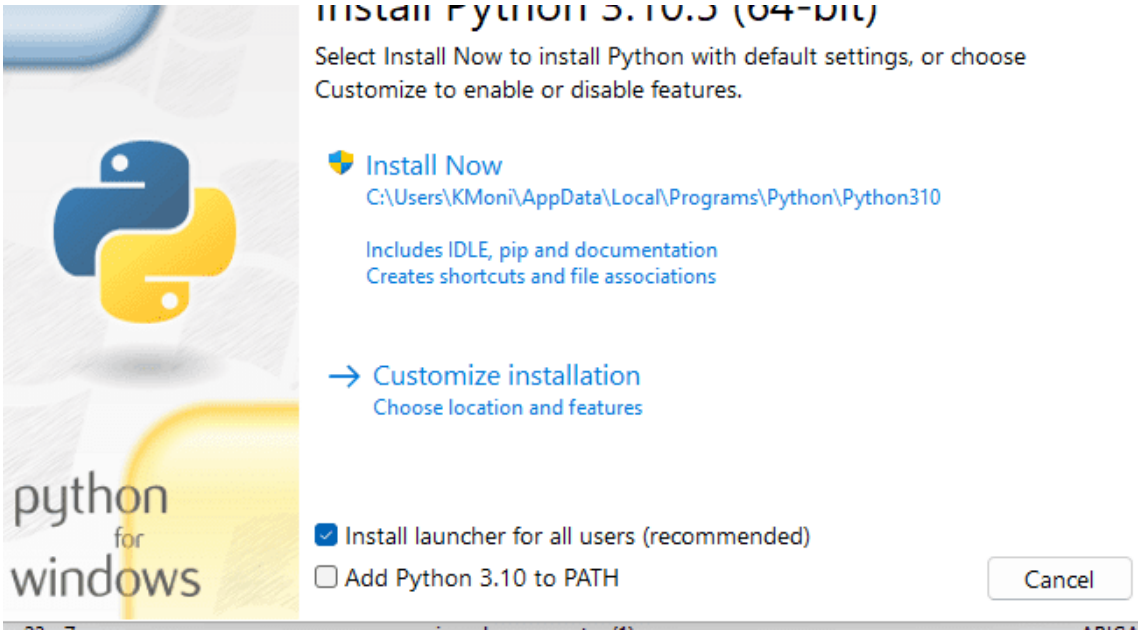


Figura 87 Instalación de Python.

Una vez instalado Python Abre Visual Studio Code y haz clic en el botón de extensiones ubicado en la barra lateral izquierda o presiona Ctrl+Shift+X .

En la barra de búsqueda, escribe "Python" y selecciona la extensión "Python" de Microsoft, como se muestra en la Figura 88.

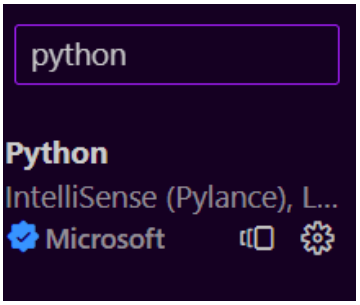


Figura 88 Agregar Python.

Haz clic en "Instalar" para instalar la Extensión.

Después de instalar la extensión, haz clic en el botón "View" en la barra superior de Visual Studio Code y selecciona "Command Palette" o presiona Ctrl+Shift+P.

En la caja de texto que aparece en la parte superior de la ventana, escribe "Python: Select Interpreter" y selecciona la opción que aparece. Selecciona la versión de Python que acabas de instalar. Si

no aparece en la lista, asegúrate de agregar la ruta de la carpeta donde se instaló Python. Visual Studio Code debería detectar automáticamente que has instalado Python y podrás comenzar a usarlo. Si necesitas ayuda adicional, puedes consultar la documentación oficial de Visual Studio Code o de Python [64], [75].

E.1.8. Primeros pasos con Python

Entorno virtual

Un entorno virtual en Python es una herramienta que permite aislar y gestionar de manera independiente las dependencias y configuraciones de un proyecto en Python. En lugar de instalar paquetes y librerías directamente en el sistema operativo, un entorno virtual crea un entorno aislado donde se pueden instalar versiones específicas de paquetes sin interferir con otros proyectos [63].

Los entornos virtuales son útiles por varias razones:

Aislamiento de dependencias: Puedes tener diferentes versiones de paquetes instalados en distintos entornos virtuales, lo que evita conflictos entre las dependencias de diferentes proyectos [66].

Reproducibilidad: Al crear un entorno virtual, puedes especificar las versiones exactas de los paquetes que necesitas para tu proyecto. Esto asegura que cualquier persona que trabaje en el proyecto pueda reproducir exactamente el mismo entorno, lo que ayuda a evitar problemas relacionados con la compatibilidad de versiones [63].

Facilidad de gestión: Los entornos virtuales facilitan la gestión de las dependencias de tu proyecto. Puedes instalar, actualizar o eliminar paquetes de manera independiente dentro de cada entorno sin afectar a otros proyectos o al sistema operativo en general [64].

Portabilidad: Puedes transferir fácilmente un entorno virtual a otro sistema, lo que facilita la colaboración con otros desarrolladores y la implementación de tu proyecto en diferentes entornos [63].

Para crear y gestionar entornos virtuales en Python, se utiliza una herramienta llamada virtualenv. Además, a partir de Python 3.3, se utiliza el módulo venv, que viene incluido en la biblioteca estándar de Python [66].

La Figura 89es tomada de la página oficial de Python (https://docs.python.org/3/library/venv.html) por lo que se sugiere se revise directamente en la página oficial [66].

venv — Creation of virtual environments

New in version 3.3.

Source code: Lib/venv/

The `venv` module supports creating lightweight "virtual environments", each with their own independent set of Python packages installed in their `site` directories. A virtual environment is created on top of an existing Python installation, known as the virtual environment's "base" Python, and may optionally be isolated from the packages in the base environment, so only those explicitly installed in the virtual environment are available.

When used from within a virtual environment, common installation tools such as pip will install Python packages into a virtual environment without needing to be told to do so explicitly.

See **PEP 405** for more background on Python virtual environments.

See also: Python Packaging User Guide: Creating and using virtual environments

Availability: not Emscripten, not WASI.

This module does not work or is not available on WebAssembly platforms `wasm32-emscripten` and `wasm32-wasi`. See WebAssembly platforms for more information.

Creating virtual environments

Creation of virtual environments is done by executing the command `venv`:

```
python -m venv /path/to/new/virtual/environment
```

Running this command creates the target directory (creating any parent directories that don't exist already) and places a `pyvenv.cfg` file in it with a `home` key pointing to the Python installation from which the command was run (a common name for the target directory is `.venv`). It also creates a `bin` (or `Scripts` on Windows) subdirectory containing a copy/symlink of the Python binary/binaries (as appropriate for the platform or arguments used at environment creation time). It also creates an (initially empty) `lib/pythonX.Y/site-packages` subdirectory (on Windows, this is `Lib\site-packages`). If an existing directory is specified, it will be re-used.

Deprecated since version 3.6: `pyvenv` was the recommended tool for creating virtual environments for Python 3.3 and 3.4, and is deprecated in Python 3.6.

Changed in version 3.5: The use of `venv` is now recommended for creating virtual environments.

On Windows, invoke the `venv` command as follows:

```
c:\>c:\Python35\python -m venv c:\path\to\myenv
```

Alternatively, if you configured the `PATH` and `PATHEXT` variables for your Python installation:

```
c:\>python -m venv c:\path\to\myenv
```

The command, if run with `-h`, will show the available options:

```
usage: venv [-h] [--system-site-packages] [--symlinks | --copies] [--clear]
            [--upgrade] [--without-pip] [--prompt PROMPT] [--upgrade-deps]
            ENV_DIR [ENV_DIR ...]

Creates virtual Python environments in one or more target directories.

positional arguments:
  ENV_DIR               A directory to create the environment in.

optional arguments:
  -h, --help            show this help message and exit
  --system-site-packages
                        Give the virtual environment access to the system
                        site-packages dir.
  --symlinks            Try to use symlinks rather than copies, when symlinks
                        are not the default for the platform.
  --copies              Try to use copies rather than symlinks, even when
                        symlinks are the default for the platform.
  --clear               Delete the contents of the environment directory if it
                        already exists, before environment creation.
  --upgrade             Upgrade the environment directory to use this version
                        of Python, assuming Python has been upgraded in-place.
  --without-pip         Skips installing or upgrading pip in the virtual
                        environment (pip is bootstrapped by default)
  --prompt PROMPT       Provides an alternative prompt prefix for this
                        environment.
  --upgrade-deps        Upgrade core dependencies: pip setuptools to the
                        latest version in PyPI
```

Figura 89 Creation of virtual environments Python.

Anexo F – Patrón de seguridad implementado C#

Documentación de Implementación en C#

F.1. Creación del microservicio uno

Abrir terminal y se crear un proyecto de tipo web API.

F.2. Codificación

launch Settings.json

URL por "http://localhost:5001”, como se muestra en la Figura 90.

Figura 90 Codificación de en launch Settings.json.

Controllers

Como muestra en la Figura 91 se agrega el nombre del controlador y los elementos.

Figura 91 Codificación de en Controllers.

F.3. Construcción

Se ejecuta el comando en la terminal.

>> *dotnet build*

F.4. Creación del microservicio dos

Se debe abrir la terminal y crear un proyecto de tipo Web API, como se muestra en la Figura 92.

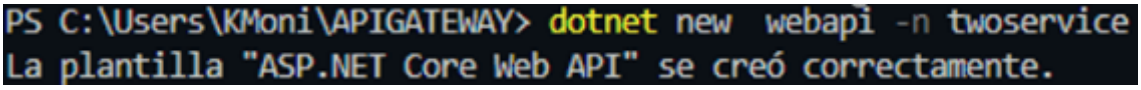


Figura 92 Creación del microservicio dos.

F.5. Codificación

launchSettings.json

URL por "http://localhost:5002", como en la Figura 93.

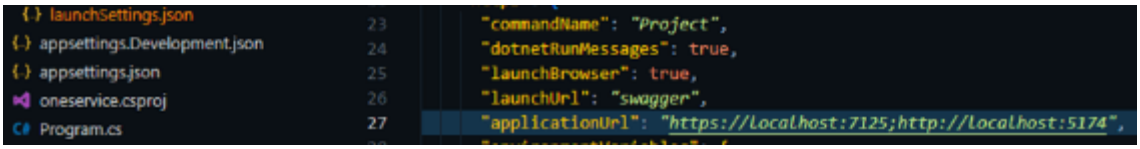


Figura 93 Codificación de en launchSettings.json.

Controllers

Nombre del controlador y los elementos con el nuevo nombre como en la Figura 94.

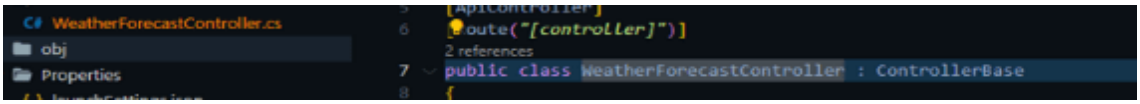


Figura 94 Codificación de Controllers.

F.6 Construcción

Se ejecuta el comando en la terminal.

>> *dotnet build*

F.7. Creación de API GATEWAY

Abrir terminal y se crear un proyecto de tipo web API

F.8. Agregar exención de Ocelot

En la API Gateway se agrega la extensión de Ocelot como en la Figura 95.

Figura 95 Agregar exención de Ocelot.

F.9. Codificación

appsettings.json contiene los elementos que se muestra en la Figura 96.

Figura 96 Codificación en appsettings.json.

Startup.cs

services.AddOcelot(); contiene los elementos que se muestra en la Figura 97.

Figura 97 Codificación en Startup.cs.

launch Settings.json contiene los elementos que se muestra en la Figura 98.

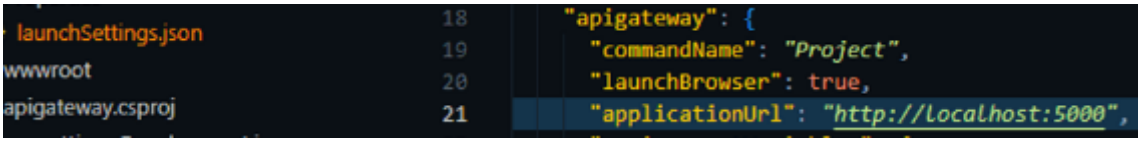


Figura 98 Codificación en launchSettings.json.

F.10. Construcción

Se ejecuta el comando en la terminal.

>> *dotnet build*

F.11. Compilación

Una vez realizados los cambios se compila cada microservicio y la API Gateway

Con el comando

>> *dotnet run*

F.12. Vista

Se procede a verificar la correcta creación de API Gateway al ingresar en el navegador las direcciones siguientes: http://localhost:5000/API/oneservice y http://localhost:5000/API/twoservice. A través de esto, se puede observar el funcionamiento de API Gateway, evidenciando que el acceso a los dos microservicios se realiza por medio del puerto 5000. Dicho puerto fue seleccionado para la API Gateway y se constituye como el único punto de entrada y acceso a los microservicios, como se aprecia en la Figura 99.

Figura 99 Vista de API Gateway.

F.13. Codificación de JWT y Codificación de algoritmo hash

La codificación del JWT se realiza en visual studio code con .net

Extensiones:

Microsoft.AspNetCore.Authentication.JwtBearer 3.0.0

Microsoft.EntityFrameworkCore.SqlServer 3.0.0

Microsoft.EntityFrameworkCore.Tools 3.0.0

Microsoft.Extensions.Logging.Debug 3.0.0

Microsoft.VisualStudio.Azure.Containers.Tools.Targets 1.9.5

Microsoft.VisualStudio.Web.CodeGeneration.Design 3.0.0

Comandos utilizados

>>dotnet new webAPI -n (nombre)

>> dotnet build

>>dotnet restore

>>dotnet run

F.13.2. Creación del JWT

Abrir terminal y crear un proyecto de tipo web API, como se muestra en la Figura 100.

Figura 100 Creación del JWT.

Agregar extensiones:

addNuGetPackage

F.13.3. Crear controladores

En Controllers crear:

C# Inventory.cs

C# InventoryController.cs

C# NameController.cs

C# UserCred.cs

F.13.4. Codificación

En la Figura 101 se muestra el contenido de C# Inventory.cs.

```
[Route("api/[controller]")]
[ApiController]
[Authorize]
public class InventoryController : ControllerBase
{
    // GET: api/Inventory
    [HttpGet]
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

    // POST: api/Inventory
    [HttpPost]
    public void Post([FromBody] Inventory value)
    {
```

Figura 101 Codificación de clase Inventory.cs.

En la Figura 102 se muestra el contenido de C# InventoryController.cs

```
[Route("api/[controller]")]
[ApiController]
[Authorize]
public class InventoryController : ControllerBase
{
    // GET: api/Inventory
    [HttpGet]
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

    // POST: api/Inventory
    [HttpPost]
    public void Post([FromBody] Inventory value)
```

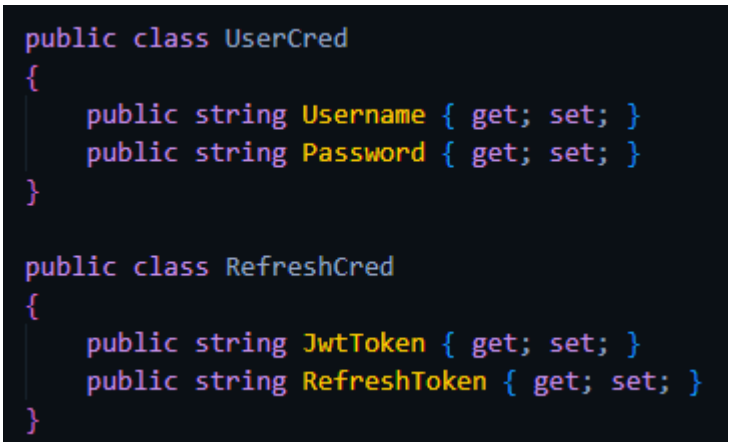
Figura 102 Codificación de InventoryController.

En la Figura 103 se muestra el contenido de C# NameController.cs

```
public NameController(IJWTAuthenticationManager jWTAuthenticationManager, ITokenRefresher tokenRefresher)
{
    this.jWTAuthenticationManager = jWTAuthenticationManager;
    this.tokenRefresher = tokenRefresher;
}
```

Figura 103 Codificación de NameController.

En la Figura 104 se muestra el contenido de C# UserCred.cs.



```
public class UserCred
{
    public string Username { get; set; }
    public string Password { get; set; }
}

public class RefreshCred
{
    public string JwtToken { get; set; }
    public string RefreshToken { get; set; }
}
```

Figura 104 Configuración de UserCred.cs.

F.13.5. Crear clases

En la carpeta de JWT crear las clases:

C# AuthenticationResponse.cs

C# BasicAuthenticationOptions.cs

C# CustomAuthenticationHandler.cs

C# CustomAuthenticationManager.cs

C# EmployeeNumberOfYearsProvider.cs

C# EmployeeWithMoreYearsHandler.cs

C# EmployeeWithMore YearsRequirement.cs

C# EmployeeNumberOfYears Provider.cs

C# IRefreshTokenGenerator.cs

C# ITokenRefresher.cs

C# JWTAuthenticationManager.cs

C# Program.cs

C# RefreshTokenGenerator.cs

C#Startup.cs

C# TokenRefresher.cs

C# User.cs

F.13.6. Codificación

En la Figura 105 se muestra el contenido de C# AuthenticationResponse.cs.

```
public class AuthenticationResponse
{
    public string JwtToken { get; set; }
    public string RefreshToken { get; set; }
}
```

Figura 105 Codificación de clase C# AuthenticationResponse.cs.

En la Figura 106 se muestra el contenido de C# CustomAuthenticationHandler.cs.

```
private readonly ICustomAuthenticationManager customAuthenticationManager;

public CustomAuthenticationHandler(
    IOptionsMonitor<BasicAuthenticationOptions> options,
    ILoggerFactory logger,
    UrlEncoder encoder,
    ISystemClock clock,
    ICustomAuthenticationManager customAuthenticationManager)
    : base(options, logger, encoder, clock)
{
    this.customAuthenticationManager = customAuthenticationManager;
}
```

Figura 106 Codificación de clase C# CustomAuthenticationHandler.cs.

En la Figura 107 se muestra el contenido de C# CustomAuthenticationManager.cs.

```
public interface ICustomAuthenticationManager
{
    string Authenticate(string username, string password);

    IDictionary<string, Tuple<string,string>> Tokens { get; }
}

public class CustomAuthenticationManager : ICustomAuthenticationManager
{
    private readonly IList<User> users = new List<User>
    {
        new User { Username= "test1", Password= "password1", Role = "Administrator" },
        new User { Username = "test2",Password= "password2", Role = "User" }
    };

    private readonly IDictionary<string, Tuple<string, string>> tokens =
        new Dictionary<string, Tuple<string, string>>();

    public IDictionary<string, Tuple<string, string>> Tokens => tokens;

    public string Authenticate(string username, string password)
    {
        if (!users.Any(u => u.Username == username && u.Password == password))
        {
            return null;
        }

        var token = Guid.NewGuid().ToString();

        tokens.Add(token, new Tuple<string, string>(username,
            users.First(u => u.Username == username && u.Password == password).Role));

        return token;
    }
}
```

Figura 107 Codificación de clase C# CustomAuthenticationManager.cs.

En la Figura 108 se muestra el contenido de C# EmployeeNumberOfYearsProvider.cs

```
public class EmployeeNumberOfYearsProvider : IEmployeeNumberOfYearsProvider
{
    public int Get(string value)
    {
        if(value == "test1")
        {
            return 21;
        }
        return 10;
    }
}
```

Figura 108 Codificación de clase C# EmployeeNumberOfYearsProvider.cs.

En la Figura 109 se muestra el contenido de C# EmployeeWithMoreYearsHandler.cs.

```
public class EmployeeWithMoreYearsHandler : AuthorizationHandler<EmployeeWithMoreYearsRequirement>
{
    private readonly IEmployeeNumberOfYearsProvider employeeNumberOfYearsProvider;

    public EmployeeWithMoreYearsHandler(IEmployeeNumberOfYearsProvider employeeNumberOfYearsProvider)
    {
        this.employeeNumberOfYearsProvider = employeeNumberOfYearsProvider;
    }

    protected override Task HandleRequirementAsync(
        AuthorizationHandlerContext context,
        EmployeeWithMoreYearsRequirement requirement)
    {
        if (!context.User.HasClaim(c => c.Type == ClaimTypes.Name))
        {
            return Task.CompletedTask;
        }

        var name = context.User.FindFirst(c => c.Type == ClaimTypes.Name);

        int numberofyears = employeeNumberOfYearsProvider.Get(name.Value);

        if(numberofyears >= requirement.Years)
        {
            context.Succeed(requirement);
        }

        return Task.CompletedTask;
    }
}
```

Figura 109 Codificación de clase C#.

EmployeeWithMoreYearsHandler.cs

En la Figura 110 se muestra el contenido de C# EmployeeWithMoreYearsRequirement.cs.

```
public class EmployeeWithMoreYearsRequirement : IAuthorizationRequirement
{
    public EmployeeWithMoreYearsRequirement(int years)
    {
        Years = years;
    }

    public int Years { get; set; }
}
```

Figura 110 Codificación de clase C# EmployeeWithMore YearsRequirement.cs.

En la Figura 111 se muestra el contenido C# JWTAuthenticationManager.cs.

```
public interface IJWTAuthenticationManager
{
    AuthenticationResponse Authenticate(string username, string password);
    IDictionary<string, string> UsersRefreshTokens { get; set; }
    AuthenticationResponse Authenticate(string username, Claim[] claims);
}

public class JWTAuthenticationManager : IJWTAuthenticationManager
{
    IDictionary<string, string> users = new Dictionary<string, string>
    {
        { "test1", "password1" },
        { "test2", "password2" }
    };

    public IDictionary<string, string> UsersRefreshTokens { get; set; }

    private readonly string tokenKey;
    private readonly IRefreshTokenGenerator refreshTokenGenerator;

    public JWTAuthenticationManager(string tokenKey, IRefreshTokenGenerator refreshTokenGenerator)
    {
        this.tokenKey = tokenKey;
        this.refreshTokenGenerator = refreshTokenGenerator;
        UsersRefreshTokens = new Dictionary<string, string>();

    if (UsersRefreshTokens.ContainsKey(username))
    {
        UsersRefreshTokens[username] = refreshToken;
    }
    else
    {
        UsersRefreshTokens.Add(username, refreshToken);
    }

    return new AuthenticationResponse
    {
        JwtToken = token,
        RefreshToken = refreshToken
    };
}

public AuthenticationResponse Authenticate(string username, string password)
{
    if (!users.Any(u => u.Key == username && u.Value == password))
    {
        return null;
    }

    var token = GenerateTokenString(username, DateTime.UtcNow);
    var refreshToken = refreshTokenGenerator.GenerateToken();

    if (UsersRefreshTokens.ContainsKey(username))
    {
        UsersRefreshTokens[username] = refreshToken;
    }
```

Figura 111 Codificación de clase C# JWTAuthenticationManager.cs.

En la Figura 112 se muestra el contenido de C# RefreshTokenGenerator.cs

```
public class RefreshTokenGenerator : IRefreshTokenGenerator
{
    public string GenerateToken()
    {
        var randomNumber = new byte[32];
        using (var randomNumberGenerator = RandomNumberGenerator.Create())
        {
            randomNumberGenerator.GetBytes(randomNumber);
            return Convert.ToBase64String(randomNumber);
        }
    }
}
```

Figura 112 Codificación de clase C# RefreshTokenGenerator.cs.

En la Figura 113 se muestra el contenido de C#Startup.cs

```
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();
    var tokenKey = Configuration.GetValue<string>("TokenKey");
    var key = Encoding.ASCII.GetBytes(tokenKey);

    services.AddAuthentication(x =>
    {
        x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(x =>
    {
        x.RequireHttpsMetadata = false;
        x.SaveToken = true;
        x.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(key),
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = true,
            ClockSkew = TimeSpan.Zero,
        };
    });

    services.AddSingleton<ITokenRefresher>(x =>
        new TokenRefresher(key, x.GetService<IJWTAuthenticationManager>()));
    services.AddSingleton<IRefreshTokenGenerator, RefreshTokenGenerator>();
    services.AddSingleton<IJWTAuthenticationManager>(x =>
        new JWTAuthenticationManager(tokenKey, x.GetService<IRefreshTokenGenerator>()));
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
```

Figura 113 Codificación de clase C#Startup.cs.

En la Figura 114 se muestra el contenido de C# TokenRefresher.cs

```
private readonly byte[] key;
private readonly IJWTAuthenticationManager jWTAuthenticationManager;

public TokenRefresher(byte[] key, IJWTAuthenticationManager jWTAuthenticationManager)
{
    this.key = key;
    this.jWTAuthenticationManager = jWTAuthenticationManager;
}

public AuthenticationResponse Refresh(RefreshCred refreshCred)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    SecurityToken validatedToken;
    var pricipal = tokenHandler.ValidateToken(refreshCred.JwtToken,
        new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(key),
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = false,
        }, out validatedToken);
    var jwtToken = validatedToken as JwtSecurityToken;
    if(jwtToken == null || !jwtToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256, StringComparison.InvariantCultureIgnoreCase))
    {
        throw new SecurityTokenException("Invalid token passed!");
    }

    var userName = pricipal.Identity.Name;
    if(refreshCred.RefreshToken != jWTAuthenticationManager.UsersRefreshTokens[userName])
    {
        throw new SecurityTokenException("Invalid token passed!");
    }

    return jWTAuthenticationManager.Authenticate(userName, pricipal.Claims.ToArray());
}
```

Figura 114 Codificación de clase C# TokenRefresher.cs.

F.14 Pruebas de patrón de seguridad implementado.

Se realizaron algunas pruebas preliminares en el sistema cada que se realizaba una modificación o se creaba una clase nueva.

A continuación, se describen algunas de las pruebas realizadas.

F.15. Creación de microservicios.

Al ejecutar el comando >>run, el sistema proporciona la confirmación de que el primer microservicio está operativo. En esta confirmación, es posible identificar la ubicación y el puerto en el que el microservicio está alojado.

Al seguir el enlace se mostró el error 404, indicando que no se pudo acceder al primer microservicio debido a que el servicio de API Gateway no estaba compilado en ese momento, como se observa en la Figura 115.

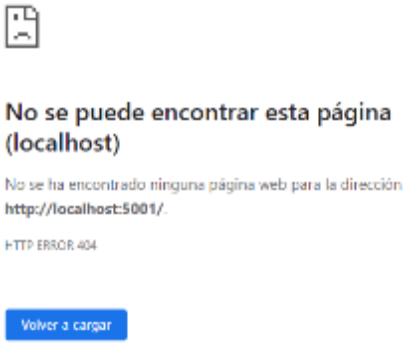


Figura 115 Pruebas de denegación al microservicio uno.

En la Figura 116 muestra el siguiente error que indica que el puerto es redireccionado por un middleware.

```
warn: Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware[3]
      Failed to determine the https port for redirect.
info: Microsoft.Hosting.Lifetime[0]
      Application is shutting down...
```

Figura 116 Warn de consola microservicio uno.

En la Figura 117 se muestra la creación correcta del segundo microservicio. Al ejecutar el comando >>run, el sistema informa que el segundo microservicio está funcionando adecuadamente. En esta notificación, es posible observar la ubicación y el puerto en el que el microservicio está alojado.

```
PS C:\Users\KMoni\Desktop\APIGATEWAY> cd twoservice
PS C:\Users\KMoni\Desktop\APIGATEWAY\twoservice> dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5002
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\KMoni\Desktop\APIGATEWAY\twoservice
```

Figura 117 Pruebas de creación correcta de microservicio dos.

Al seguir el vínculo se mostró el error 404 ya que no se puede acceder al segundo microservicio si no está compilando el servicio de API Gateway. En la Figura 118 muestra el error que indica que el puerto es redireccionado por un middleware.

```
      Content root path: C:\Users\KMoni\Desktop\APIGATEWAY\twoservice
warn: Microsoft.AspNetCore.HttpsPolicy.HttpsRedirectionMiddleware[3]
      Failed to determine the https port for redirect.
```

Figura 118 Warn de consola microservicio dos.

Prueba de API Gateway

En la Figura 119 se muestra la creación correcta de API Gateway. Al ejecutar el comando >>run, el sistema proporciona la confirmación de que la API Gateway está funcionando de manera adecuada. En esta confirmación, es posible observar la ubicación y el puerto en los que la API Gateway se está ejecutando.

```
PS C:\Users\KMoni\Desktop\APIGATEWAY> cd apigateway
PS C:\Users\KMoni\Desktop\APIGATEWAY\apigateway> dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\KMoni\Desktop\APIGATEWAY\apigateway
```

Figura 119 Pruebas de creación correcta de API Gateway.

La Figura 120 muestra un mensaje de bienvenida ya que está compilando el servicio de API Gateway.

Figura 120 Pruebas de compilación correcta de API Gateway mensaje de bienvenida.

Prueba de API Gateway general

Una vez que los componentes, incluyendo la API Gateway, el primer microservicio y el segundo microservicio, han sido compilados exitosamente, el siguiente paso consiste en

dirigirse a: http://localhost:5000/API/oneservice y http://localhost:5000/API/twoservice

En este punto, se puede observar que la API que ha sido creado está funcionando correctamente. Además, se ha establecido el acceso a cada uno de los microservicios. Como muestra la Figura 121.

Figura 121 Pruebas de compilación correcta de API Gateway acceso a microservicios.

Anexo G – Plan de pruebas

Propósito

El plan de pruebas descrito a continuación tiene el propósito de mostrar la planificación, estructura y documentación, de las pruebas realizadas en el Patrón de seguridad: "Microservice Security Pattern API Gateway" definidas en esta investigación, su aplicación y los resultados.

G.1 Identificador

A continuación, se observa la nomenclatura que se utiliza en la identificación de los diferentes documentos que serán generados en las diferentes pruebas a partir del nombre Microservice Security Pattern API Gateway (MSPAG).

MSPAG- TD- XX

En la **¡Error! No se encuentra el origen de la referencia.** se presenta l a composición detallada de cada uno de los elementos, como se muestra a continuación:

Tabla 23 Identificadores de nomenclaturas

Nomenclatura	Identificador	Descripción
MSPAG	Patrón de seguridad	MSPAG
TD	Tipo de Documento	PP: Plan de pruebas DP: Diseño de Pruebas CP: Casos de Prueba
XX	Número de prueba	Determina el número del documento

G.2 Documentación

La documentación desarrollada para las pruebas, se encuentra detallada en los siguientes documentos:

- MSPAG-DP-XX: Especificación de diseños de pruebas.
- MSPAG-CP-XX: Especificación de casos de pruebas.

La descripción de las pruebas se muestra a continuación.

G.3 Elementos de pruebas

En la Tabla 24Tabla 24 se muestra el Patrón de seguridad: "MSPAG" como elementos de prueba que serán evaluados para comprobar su correcto funcionamiento, mismas que cumplen con los requerimientos mínimos en su definición.

Tabla 24 Elementos de prueba del MSPAG

Id	Medida de seguridad	Descripción
AG	*API Gateway*	*API Gateway*, gestiona las solicitudes y las redirige a los microservicios.
	Descripción: La API Gateway es capaz de impedir el acceso a los microservicios, aunque el cliente tenga conocimiento de la URL específica de estos.	
J/H	*JWT/HASH*	JWT, auténtica y autoriza el acceso al microservicio, mientras que el algoritmo HASH, verifica que el token no ha sido corrupto.
	Descripción: El JWT específica, genera y valida el token, que incluye: un usuario y una contraseña. Posteriormente, el algoritmo criptográfico H256 protege los datos de transmisión.	

G.4 Criterios de aceptación y suspensión

G.4.1 Criterios de aceptación

Los criterios de aceptación se basan en el cumplimiento exitoso de cada una de las instancias de prueba. Para cada caso de prueba se analiza el funcionamiento y protección de los microservicios con el Patrón de seguridad: MSPAG que deberá cumplir con los criterios antes mencionados.

G.4.2 Criterios de suspensión

El criterio de suspensión se basa en el incumplimiento de alguna instancia, es decir, cada vez que un caso cumpla con la prueba, inmediatamente se procederá a evaluar y documentar los errores arrojados en el proceso específico que se haya presentado, permaneciendo en las pruebas hasta que cumpla los criterios en su totalidad.

G.5 Entregables

Como resultado del diseño y ejecución de las pruebas de aceptación, se generaron los siguientes documentos:

Plan de Pruebas:

- Diseños de pruebas.

Reporte de Ejecución de Pruebas:

- Bitácora de pruebas.

G.6 Liberación

La liberación de una prueba se llevó a cabo cuando su estado es aceptado y cumple con la documentación necesaria para la ejecución de la misma desde su inicio hasta su conclusión.

G.7 Actividades

Las actividades desarrolladas durante la preparación y ejecución de las pruebas de aceptación fueron las siguientes:

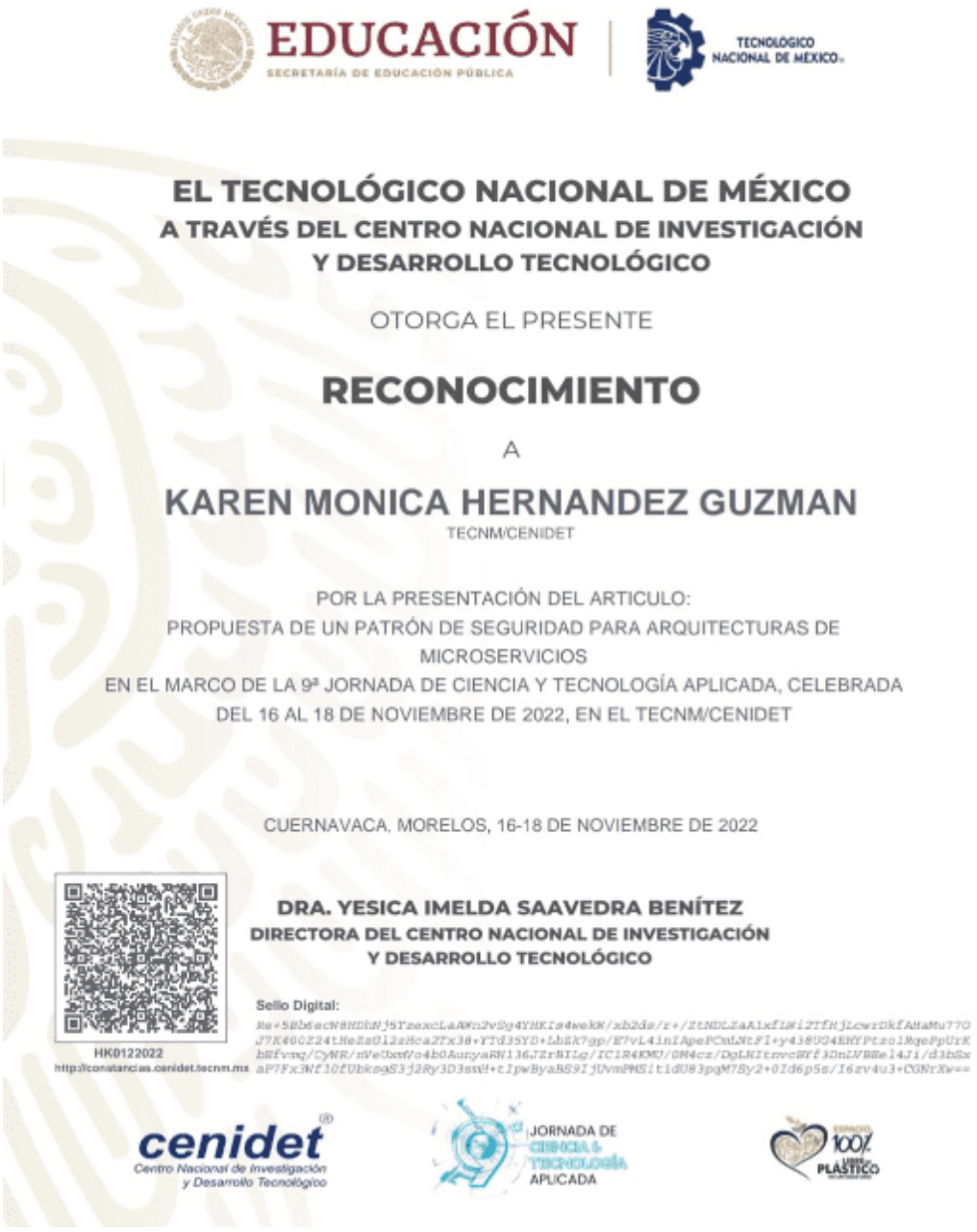
- Generación del plan de pruebas.
- Diseño de casos de prueba.
- Ejecución de casos de prueba.
- Generación de la bitácora de pruebas

G.8 Aprobación

El plan de pruebas fue revisado por el Dr. Juan Carlos Rojas Pérez.

Anexo H - Escritura de artículo para JCyTA

Escritura de artículo para JCyTA.



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

EL TECNOLÓGICO NACIONAL DE MÉXICO
A TRAVÉS DEL CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO

OTORGA EL PRESENTE

RECONOCIMIENTO

A

KAREN MONICA HERNANDEZ GUZMAN
TECNM/CENIDET

POR LA PRESENTACIÓN DEL ARTICULO:
PROPUESTA DE UN PATRÓN DE SEGURIDAD PARA ARQUITECTURAS DE MICROSERVICIOS
EN EL MARCO DE LA 9ª JORNADA DE CIENCIA Y TECNOLOGÍA APLICADA, CELEBRADA DEL 16 AL 18 DE NOVIEMBRE DE 2022, EN EL TECNM/CENIDET

CUERNAVACA, MORELOS, 16-18 DE NOVIEMBRE DE 2022

HK0122022
http://constancias.cenidet.tecnm.mx

DRA. YESICA IMELDA SAAVEDRA BENÍTEZ
DIRECTORA DEL CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO

Sello Digital:

Rs+SEb6acN8NDbNj9TzexcLaAKn2v2g4YHKIs4wekN/xb2dz/r+/ZtNDLZaA1xfLWi2TfHjLcwrUkfAHaMu77O
J7K40C224tHeZzQ12zHca2Tx38+YTd39SYD+LbZk7gp/E7vL4inIApePCmLNtFl+y438UO4EHYPtzolRgePpUrK
bEfvmg/CyNR/nVe3nMfo4b0AuryaRN136JZrNILg/IC1R4KMJ/6M4cz/DpLNItnvcRYf3DnlVBBe14J1/d3bSx
aP7FxXNf10fUbkogS3j2Ry3D3smV+clpwBya8S9IjUvmPMSIt1dU83pgM7Sy2+0Id6p5s/I6zv4u3+CGNrXw==

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

JORNADA DE CIENCIA & TECNOLOGÍA APLICADA

ESTAMOS 100% LIBRES DE PLÁSTICO

Figura 122 Reconocimiento de JCyTA.