EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Selección y adaptación de métricas para Microservicios

presentada por

Ing. Iván Daniel Joya de la Cruz

como requisito para la obtención del grado de

Maestro en Ciencias de la Computación

Director de tesis

Dr. Juan Carlos Rojas Pérez

Codirectora de tesis

Dra. Olivia Graciela Fragoso Díaz

Cuernavaca, Morelos, México. Marzo 2022

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Centro Nacional de Investigación y Desarrollo Tecnológico
Departamento de Ciencias Computacionales

Cuernavaca, Morelos, 25/marzo/2022

OFICIO No. DCC/024/2022
Asunto: Aceptación de documento de tesis
CENIDET-AC-004-M14-OFICIO

DR. CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial del C. IVAN DANIEL JOYA DE LA CRUZ, con número de control M20CE039, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis de grado titulado **"SELECCIÓN Y ADAPTACIÓN DE MÉTRICAS PARA MICROSERVICIOS"**, 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.

DR. JUAN CARLOS ROJAS PÉREZ
Director de tesis

DRA. OLIVIA GRACIELA FRAGOSO DÍAZ
Codirectora de Tesis

DR. RENÉ SANTAOLAYA SALGADO
Revisor 1

M.C. HUMBERTO HERNÁNDEZ GARCÍA
Revisor 2

Revisor 3

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

Cuernavaca, Mor., 24/marzo/2022
No. De Oficio: SAC/58/2022
Asunto: Autorización de impresión de tesis

IVÁN DANIEL JOYA DE LA CRUZ
CANDIDATO AL GRADO DE MAESTRO 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 **"SELECCIÓN Y ADAPTACIÓN DE MÉTRICAS PARA MICROSERVICIOS"**, 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.

EDUCACIÓN | TECNOLÓGICO NACIONAL DE MÉXICO
28 MAR 2022
CENTRO NACIONAL DE INVESTIGACIÓN Y DESARROLLO TECNOLÓGICO
SERVICIOS
REC
EBH

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Educación Tecnológica al Servicio de México"

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

DR. CARLOS MANUEL ASTORGA ZARAGOZA
SUBDIRECTOR ACADÉMICO

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

CMAZ/CHG

A mi familia,
el mejor regalo que pudo darme la vida.
Por ser mi apoyo, mi motivación y mi hogar.

Agradecimientos

A mis padres Silvano y Apolonia, por su incondicional apoyo, amor y cariño. A mis hermanos y todos mis sobrinos que siempre estuvieron conmigo a la distancia. ¡Los amo!

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) y al Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET), por brindarme la oportunidad y el apoyo para realizar mis estudios de maestría.

A mi director de tesis, el Dr. Juan Carlos Rojas Pérez, por guiarme con su gran experiencia en el trabajo de investigación, mi sincero agradecimiento por la paciencia, el tiempo y todos los consejos durante mis estudios.

A la Dra. Olivia Fragoso Diaz, por sus observaciones, opiniones y valiosas aportaciones que hicieron que esta investigación adquiriera una mejor dirección.

Al comité revisor: Dr. René Santaolaya Salgado y M.C. Humberto Hernández García, por el tiempo dedicado y contribuir en la revisión de esta investigación.

A Uriel, quien me impulso en la decisión de seguir estudiando, por tú compañía en las noches de estudio y los días de estrés, los increíbles viajes de relajación y compartir todas las facetas de la maestría, este logro también en tuyo, te quiero mucho.

A mi amigo Elioenai, por siempre estar en las buenas, malas, peores y siempre tener palabras de motivación y aliento en el momento indicado, te quiero mucho.

A mis amigos y compañeros de la maestría, Leslie, Alejandra, Edson, Juan Antonio, Yazmín y Javier, que compartieron sus conocimientos, aventuras e hicieron más agradables las clases a distancia.

Resumen

Las arquitecturas de software han evolucionado en las últimas décadas, en general de arquitecturas monolíticas hacia las orientadas a servicios, en este proceso de transición los mecanismos para evaluar la calidad de arquitecturas previas a la de microservicios requieren de cambios para su aplicación.

En este proceso de transición hacia arquitecturas de microservicios, las métricas constituyen medidas que permiten evaluar objetivamente la calidad arquitectónica de los microservicios proporcionando a desarrolladores y arquitectos indicaciones claras de deficiencias y errores.

En este trabajo se presenta un estudio de las métricas utilizadas en arquitecturas orientadas a objetos para su adaptación hacia arquitecturas orientadas a microservicios.

El resultado fue la adaptación de la métrica RFC (Response For a Class) a través de la creación de las métricas NIU (Número de Interfaces Utilizadas) y NO (Número de Operaciones).

Las métricas fueron probadas en repositorios de microservicios, con la ejecución de seis casos de prueba, obteniendo resultados que muestran la comunicación de los microservicios y el entendimiento en su funcionamiento interno.

Abstract

Software architectures have evolved in recent decades, generally from monolithic architectures to service-oriented architectures. In this transition process, the mechanisms for evaluating the quality of architectures previous to those of microservices require changes for their application.

In this transition process towards microservices architectures, the metrics constitute measures that allow objective evaluation of the architectural quality of the microservices, providing developers and architects with indications of deficiencies and errors.

This paper presents a study of the metrics used in object-oriented architectures for their adaptation to microservice-oriented architectures.

The result was the adaptation of the RFC metric (Response For a Class) through the creation of the metrics NIU (Number of Interfaces Used) and NO (Number of Operations).

The metrics were tested in microservices repositories, with the execution of six test cases, obtaining results that show the communication of the microservices and the understanding of their internal operation.

Contenido

Índice de Ilustraciones

Índice de tablas

CAPÍTULO

01

Introducción

En este capítulo se presenta el panorama general del proyecto de tesis, el problema que dio origen a la investigación, los objetivos establecidos, la justificación y los resultados esperados.

El desarrollo de software hoy en día requiere del uso de nuevas tecnologías, paradigmas y metodologías que contribuyan a obtener mejores productos de software [1].

El software se ha vuelto una parte importante en la vida cotidiana, por ello, es importante que los sistemas puedan adaptarse rápidamente a nuevos requisitos y actualizaciones [2].

Las arquitecturas de software han sido un tema de investigación, en donde se han propuesto cambios y mejoras a lo largo del tiempo [3]. En la arquitectura monolítica tradicional, un sistema de software se construye como una unidad lógica única que agrega varios servicios que comparten los mismos recursos computacionales (por ejemplo, espacio de memoria, procesamiento de CPU y base de datos) para proporcionar funcionalidades comerciales [4]. Aunque las aplicaciones monolíticas son fáciles de desarrollar, presentan limitaciones, tales como dificultades para mantener y evolucionar, seguidas de escalado ineficiente de los recursos [3]. Para superar su problema, se han propuesto nuevos estilos arquitectónicos en los últimos años [5].

Las Arquitecturas Orientadas a Servicios (SOA) intentaron resolver este problema, dividiendo las aplicaciones en conjuntos de aplicaciones que ofrecen servicios a través de diferentes protocolos. Sin embargo, el desarrollo de aplicaciones SOA es costoso y complejo, debido a que las aplicaciones están diseñadas para soportar cargas de trabajo muy altas, con una gran cantidad de usuarios [5]. En la mayoría de los casos, es un enfoque ineficiente para responder a cargas de trabajo que cambian rápidamente mientras se mantiene el uso óptimo de los recursos [6].

En este sentido los microservicios surgen como un estilo arquitectónico para desarrollar una aplicación como una colección de servicios independientes, bien definidos e intercomunicados [4]. Amazon, Netflix, LinkedIn, Spotify, SoundCloud y otras compañías han desarrollado sus aplicaciones utilizando una Arquitectura de Microservicios (MSA) [7]. La popularidad de los microservicios ha crecido en los últimos años, sin embargo, la investigación aún carece de una comprensión disciplinada de la transición y del consenso sobre los principios y actividades subyacentes a las arquitecturas “Micro” [8].

En este proceso de transición hacia MSA, las métricas constituyen medidas que permiten evaluar objetivamente la calidad arquitectónica de los Microservicios, proporcionando a desarrolladores y arquitectos indicadores de errores y deficiencias [9]. Si bien las métricas orientadas a objetos (OO) no son completamente aplicables a arquitecturas SOA, la mayoría de estas métricas podrían ser aplicables a microservicios con algunas modificaciones [9], lo que sí está claro, es que la selección de las métricas sigue siendo difícil para los profesionales.

1.1. Descripción del problema

Chidamber y Kemerer propusieron una suite de métricas de diseño orientadas a objetos que se adoptaron en el desarrollo de software [10]. Su utilización creció a lo largo de los años, se incrementó la cantidad de pruebas realizadas por investigadores y desarrolladores de software, logrando la aceptación por la comunidad. Sin embargo, la aplicación de estas métricas en arquitecturas modernas requiere de modificaciones en su gran mayoría, por ello, se han comenzado a proponer extensiones de las métricas, por ejemplo, una extensión de la métrica Acoplamiento entre Objetos (CBO) como Acoplamiento entre Microservicios (CBM) [11].

La popularidad de los microservicios ha crecido en los últimos años, sin embargo, la investigación aún carece de una comprensión disciplinada de la transición, principios y actividades subyacentes a las arquitecturas “Micro” [8]. Debido a esta falta de comprensión, existen formas limitadas de medir la calidad de microservicios, en donde las métricas juegan un papel importante.

La carencia de métricas bien establecidas y herramientas limitadas para medir la calidad de microservicios, conducen a la aplicación errónea de métricas en un proyecto de software.

El resultado son indicadores que no reflejan el estado del microservicio, ni guían a los desarrolladores a mejorar sus desarrollos, obteniendo diseños de software deficientes. Para abordar este problema, diferentes investigadores han identificado posibles métricas: Interdependencia de Servicios en el Sistema (SIY), Importancia Absoluta del Servicio (AIS), Dependencia Absoluta del Servicio (ADS) entre otras como primer paso hacia la identificación de métricas adecuadas en el campo [12].

1.2. Objetivos

1.2.1. General

Desarrollar una métrica de software que permita evaluar la calidad de sistemas orientados a microservicios mediante la adaptación de una métrica aplicada en arquitecturas orientadas a objetos.

1.2.2. Específicos

- Realizar un estudio para la selección y evaluación de métricas orientadas a objetos propuestas por Chidamber y Kemerer y otros autores.
- Revisar y analizar las métricas resultantes del estudio para el diseño, instalación (deploy), tolerancia a fallos o servicios.
- Mediante la revisión y análisis anterior determinar que métricas pueden ser extendidas o adaptadas en el desarrollo de microservicios.

1.3. Alcances y Limitaciones

1.3.1. Alcances

- Explorar las métricas para el desarrollo de software que puedan ser extendidas para su aplicación en el desarrollo de microservicios.
- Considerar métricas orientadas a objetos enfocadas al diseño, instalación (deploy), tolerancia a fallos y servicios.

1.3.2. Limitaciones

- La investigación se enfoca únicamente en las métricas que se encuentren formalizadas y utilizadas por la comunidad científica.
- Solo se consideran para el estudio el acceso a las fuentes de datos del CENIDET, tales como ACM digital library, AMS, Cengage Learnig, IEEE Xplore, Scielo, Thomson Reuters, entre otras como Google académico.
- Adaptar al menos 1 métrica.

1.4. Trabajos relacionados

En una revisión del estado del arte se encontraron los siguientes trabajos: Una investigación en el proceso de migración de sistemas monolíticos a microservicios, presenta un Framework de descomposición basado en minería de procesos, como parte de su metodología analizan métricas para asegurar la calidad del proceso de descomposición, proponen una extensión a la métrica de Acoplamiento entre Objetos (CBO) denominada Acoplamiento entre Microservicios (CBM) [11]; este Framework de descomposición es utilizado como base para el Framework de medición para comparar objetivamente dos opciones de descomposición [1].

El Framework [1] de medición se puede utilizar independientemente de la estrategia de descomposición adoptada, la investigación utiliza la métrica CBM y agrega las métricas CLA (Número de Clases por Microservicio), DUP (Número de Clases duplicadas) y FEC (Frecuencia de Llamadas Externas) para evaluar el resultado de descomposición, siendo este trabajo una extensión al Framework [11].

A nivel arquitectónico se ha propuesto una nueva métrica para evaluar la complejidad del mantenimiento de la arquitectura de software, se le ha denominado Nivel de Desacoplamiento (DL), la cual mide qué tan bien se desacopla el software en módulos pequeños e independientemente reemplazables [13]. Se presentan propuestas de adaptaciones de métricas de software enfocadas a sistemas (complejidad, acoplamiento, centralización, interdependencia de un servicio) y servicios (granularidad, acoplamiento, número, importancia, criticalidad, inestabilidad, acoplamiento indirecto, acoplamiento indirecto relativo, acoplamiento promedio, promedio de cohesión, cohesión de operaciones, complejidad, factor de complejidad, peso) [14].

Un estudio sobre la refactorización arquitectónica discute la noción de refactorización comparando 10 enfoques de refactorización existentes recientemente propuestos en la literatura académica,

dentro de los enfoques revisados algunos describen métricas para una evaluación de resultados, mencionan métricas de rendimiento que miden la calidad de una implementación, métricas basadas en el código fuente y las métricas personalizadas: Reducción del tamaño del equipo (TSR) y Redundancia de dominio promedio (ADR) [6].

Se ha descrito un modelo de mantenimiento para servicios (MM4S) y un modelo de calidad (QM) en capas con métricas de servicio cuyo objetivo es servir como una herramienta simple y práctica para la estimación y el control de mantenibilidad de servicios y microservicios: En la Ilustración 1 se observa una representación visual del MM4S. Si bien 4 de las 11 métricas (CB, CR, CC, TC) no fueron diseñadas específicamente para microservicios, las vemos como una base valiosa [15].

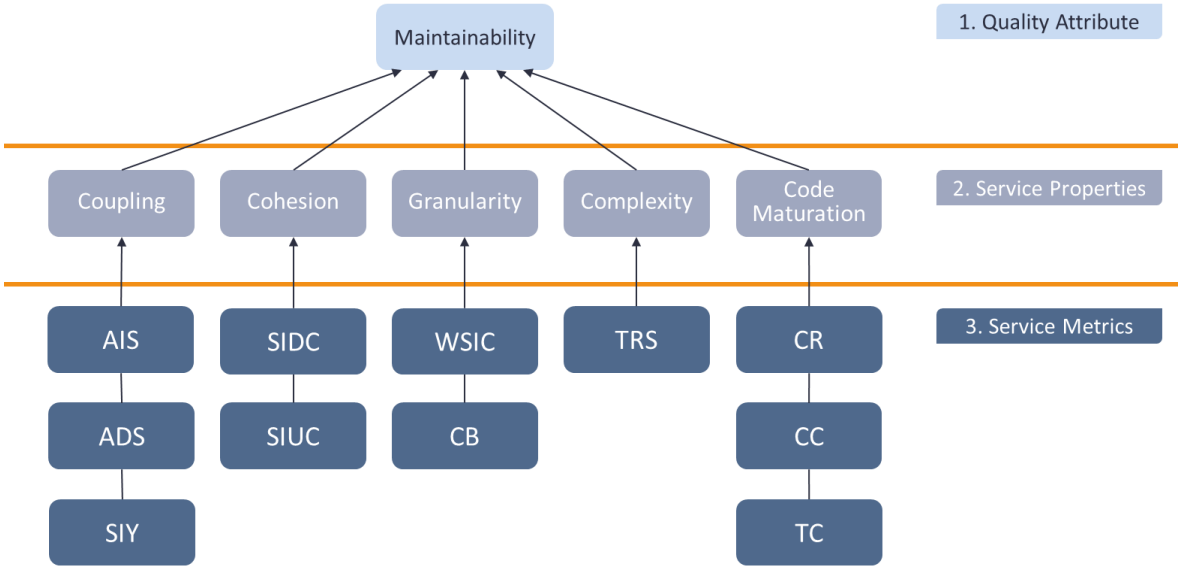


Ilustración 1. Modelo de Mantenibilidad para Servicios (MM4S) [15]

A continuación, se describe la clasificación de las métricas mencionadas:

A. Cohesión
1) Cohesión de datos de interfaz de servicio (SIDC)
2) Cohesión de uso de la interfaz de servicio (SIUC)

B. Complejidad
1) Respuesta total para el servicio (TRS)

C. Acoplamiento
1) Importancia absoluta del servicio (AIS)
2) Dependencia absoluta del servicio (ADS)
3) Interdependencia de servicios en el sistema (SIY)

D. Maduración del Código
1) Comentario Ratio (CR)
2) Cobertura de clonación (CC)
3) Prueba de cobertura (TC)

Otro enfoque del estudio sobre microservicios busca la aplicación de políticas sobre software basado en esta arquitectura. Su objetivo es medir la calidad del software con métricas y políticas recientemente definidas. Las métricas se definieron para medir el tamaño del servicio y la comunicación entre servicios [16]. Las métricas y políticas presentadas son categorizadas de la siguiente forma:

A. Medida del tamaño del servicio
   1) Recuento de recursos (RC)
   2) Recuento de clientes (CC)

B. Compatibilidad de comunicación entre servicios
   1) Recuento de recursos no utilizados (URC)
   2) Recuento de punto final inalcanzable (UEC)

C. Malas prácticas
   1) URI estático (SURI):
   2) URI largo (LURI):

Si bien existen varias posibilidades para evaluar la calidad del software, siendo las más importantes los enfoques basados en escenarios y métricas, todavía no existe un método dominante para evaluar la mantenibilidad. En este contexto, para la medición automática de la mantenibilidad de los sistemas basados en microservicios y servicios presentaron diferentes métricas de la literatura que intentan medir automáticamente la capacidad de mantenimiento de los sistemas de software en el trabajo [12].

A. Tamaño:
   1) Recuento de interfaz de servicio ponderado (WSIC)

B. Complejidad:
   1) Respuesta total por servicio (TRS)
   2) Número de versiones por servicio (NVS)
   3) Servicio de soporte para transacciones (SST)

C. Acoplamiento:
   1) Interdependencia de servicios en el sistema (SIY)
   2) Importancia absoluta del servicio (AIS)
   3) Dependencia absoluta del servicio (ADS)
   4) Crítica absoluta del servicio (ACS)

D. Cohesión:
   1) Cohesión de datos de interfaz de servicio (SIDC)
   2) Cohesión de uso de interfaz de servicio (SIUC)
   3) Cohesión de interfaz de servicio total (TSIC)

En general, la mayoría de las métricas que se han propuesto explícitamente para sistemas basados en servicios también podrían ser aplicables a sistemas basados en microservicios. En el trabajo de [12] se describen algunas limitaciones al referirse a tres características de microservicio con una influencia ligeramente obstaculizadora (gran número de servicios pequeños, heterogeneidad tecnológica y descentralización del control). El principal obstáculo que identificaron para la aplicabilidad práctica de las métricas presentadas es la complicada recopilación automática en un sistema basado en microservicios. Como parte de su investigación comparten una lista de 46 métricas de mantenibilidad descubiertas para servicios. En esa lista están las métricas que aparecen con mayor frecuencia en la literatura, presentando la propiedad del diseño de la forma más integral posible, presentan el enfoque de evaluación más sólido o que fueran aplicables a un solo servicio (no al sistema completo) [12].

En general, la mayoría de las métricas aplicadas se centraron en la calidad del código fuente, a pesar de que algunos participantes consideraron controvertida su efectividad para todo el sistema. No se aplicaron herramientas y métricas arquitectónicas, a pesar de que la mayoría de los desafíos y problemas reportados, como el corte de servicios, estaban relacionados con la arquitectura de software.

Como se puede observar en los trabajos antes citados, existe una gran diversidad de extensiones y propuestas de métricas, sin embargo, aún carecen de pruebas suficientes para comprobar su eficiencia en las arquitecturas de microservicios.

1.5. Estructura del documento

El documento de tesis está organizado de la siguiente manera:

En el capítulo 2 se presenta el marco teórico, en donde se abordan los conceptos básicos de microservicios, métricas y atributos de calidad, entro otros relacionados con las métricas orientadas a objetos.

En el capítulo 3 se presenta la implementación de la solución, se describe el proceso de análisis, selección y adaptación de la métrica orientada a objetos a una arquitectura de microservicios.

En el capítulo 4 se presentan las pruebas experimentales y sus resultados al aplicar las métricas resultantes en microservicios de diferentes repositorios, así como la interpretación.

Y en el capítulo 5 se presentan las conclusiones obtenidas, las aportaciones de esta tesis y los trabajos futuros sugeridos para continuar con la investigación.

CAPÍTULO

02

Marco Teórico

En este capítulo se presentan las definiciones y conceptos básicos de las métricas y la arquitectura de microservicios.

2.1. Arquitectura de microservicios

Microservicios es un estilo arquitectural que fue acuñado por Martin Fowler en un taller de arquitectos de software como una descripción del nuevo campo [17]. No existe una definición en concreto para microservicio, sin embargo, una aproximación lo define como: ***"Proceso coherente e independiente que interactúa a través de mensajes con otros"*** [3].

2.1.1. Microservicios

La arquitectura de microservicios no es nueva en absoluto. De hecho, es bastante similar al patrón SOA que fue muy popular hace unos años. Ambos enfoques promueven el desarrollo y despliegue de aplicaciones compuestas por unidades independientes, autónomas, modulares, autocontenidas y cada una debe implementar una funcionalidad de negocio individual, lo cual difiere de la forma tradicional o monolítica [18]. En la Ilustración 2, se observa un ejemplo de una arquitectura de microservicios.

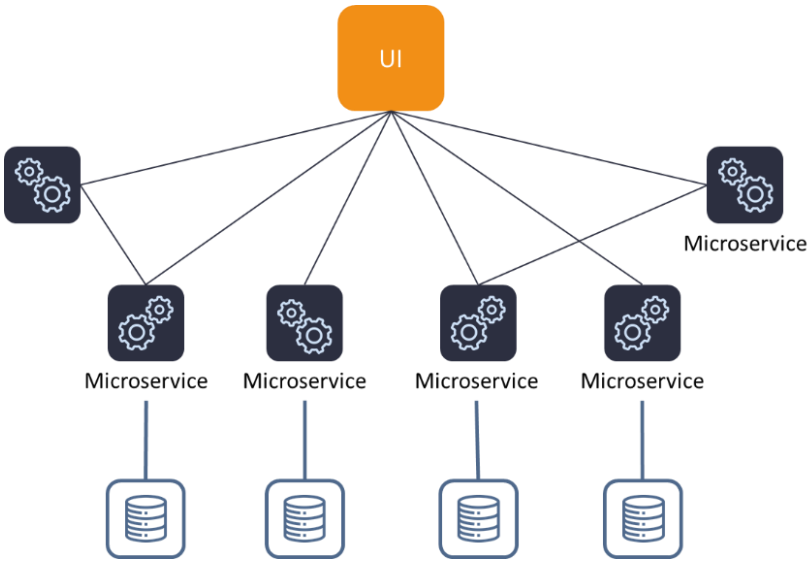


Ilustración 2. Esquema de arquitectura de microservicios [19]

Estructura

Los colaboradores de ThoughtWorks [17], describen la estructura interna de un microservicio (Ilustración 3) así como sus capas:

- *Protocolo*

Recursos: Verifican las peticiones de red y dan una respuesta específica al protocolo de acuerdo al resultado de la solicitud.

- *Dominio*

Capa de servicio: Actúa como mediador por si la aplicación requiere coordinar varios servicios para completar peticiones más personalizadas.

Repositorio: Actúa junto a la colección de entidades de dominios. Tiene como objetivo almacenar en memoria las listas de los objetos del dominio necesarios para manipular la base de datos.

Dominio: Representa un objeto que se asocia a una entidad de la base de datos.

- *Persistencia*

Mapeadores de datos / ORM: Para persistir los objetos del dominio entre las peticiones, se utiliza un mapeo objeto relacional. La base de datos se acopla al sistema, pero como se encuentra por fuera de los límites de la red pueden presentarse problemas de latencia y riesgo de interrupción.

- *Externo*

Si un servicio requiere de la funcionalidad de otro servicio, se necesita cierta lógica para comunicarse con el servicio externo. Una puerta de enlace encapsula un mensaje para transmitirlo hacia un servicio remoto, transformando solicitudes y respuestas desde y hacia objetos del dominio.

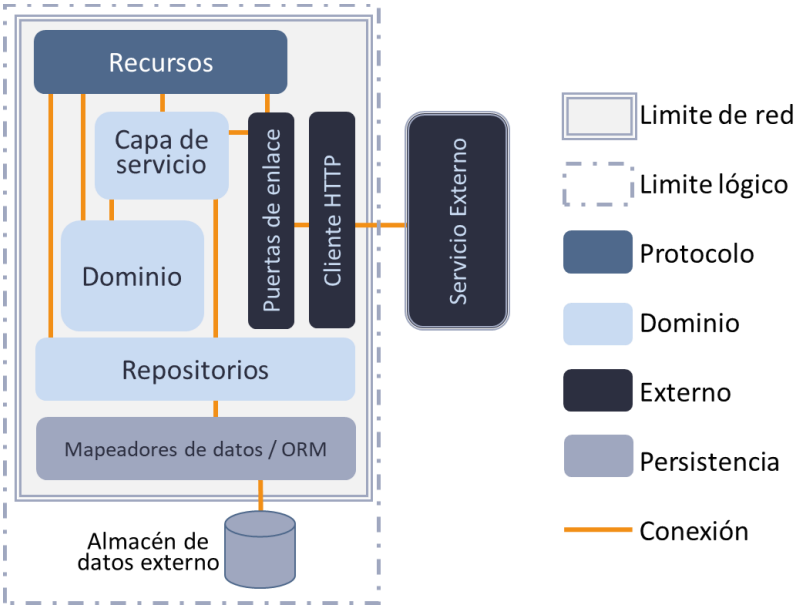


Ilustración 3. Estructura interna de un microservicio [20]

2.1.2. Identificación de los límites de microservicios

¿Qué tamaño debe tener un microservicio? Al desarrollar un microservicio, el tamaño no es lo más importante. En su lugar, el punto importante es crear libremente servicios acoplados que tengan autonomía de desarrollo, implementación y escala para cada servicio. Por supuesto, al identificar y diseñar microservicios, debe intentarse que sean lo más pequeños posible, siempre y cuando no tenga demasiadas dependencias directas con otros microservicios [21].

Más importante que el tamaño del microservicio es la cohesión interna que debe tener y su independencia respecto a otros servicios. El énfasis no está en el tamaño, sino en las capacidades empresariales [22].

2.1.3. Comunicación

En esta arquitectura los servicios deben interactuar mediante un protocolo de comunicación entre procesos como HTTP, AMQP o un protocolo binario como TCP, en función de la naturaleza de cada servicio. Normalmente los microservicios que componen una aplicación de un extremo a otro se establecen sencillamente mediante el protocolo de comunicaciones REST en lugar de protocolos complejos [23].

Tipos de comunicación

1) Protocolo sincrónico

HTTP/HTTPS es un protocolo sincrónico. El cliente envía una solicitud y espera una respuesta del servicio. El código de cliente solo puede continuar su tarea cuando recibe la respuesta del servidor HTTP [23].

2) Protocolo asincrónico

Otros protocolos como AMQP (un protocolo compatible con muchos sistemas operativos y entornos de nube) usan mensajes asincrónicos. El código de cliente o el remitente del mensaje no espera ninguna respuesta. Simplemente envía el mensaje al igual que cuando se envía un mensaje a una cola de RabbitMQ o a cualquier otro agente de mensajes.

3) Receptor único

Cada solicitud debe ser procesada por un receptor o servicio específico.

4) Varios receptores

Cada solicitud puede ser procesada por entre cero o varios receptores. Este tipo de comunicación debe ser asincrónica. Un ejemplo es el mecanismo de publicación o suscripción empleado en patrones como la arquitectura controlada por eventos.

En los ejemplos implementados que se describen en este documento se utiliza un protocolo del tipo más común, la comunicación de un único receptor con un protocolo sincrónico como HTTP/HTTPS al invocar a un servicio normal HTTP Web API. Además, los microservicios suelen usar protocolos de mensajería para la comunicación asincrónica entre microservicios [21].

2.1.4. Características

a. Autónomos

Un microservicios es una entidad separada. Cada servicio/componente en una arquitectura de microservicios se puede desarrollar, implementar, operar y escalar sin afectar el funcionamiento de otros servicios [2, 3].

b. Especializados

Los microservicios pretenden aislar la lógica de negocio en un solo lugar, lo que permite disminuir el riesgo de hacer crecer mucho el componente [24].

c. Organizados

La forma en que se organizan los microservicios suele ser en torno a necesidades, capacidades y prioridades del cliente o negocio en el que se implantará [19].

d. Libres tecnológicamente

Las arquitecturas de microservicios no siguen un enfoque de "diseño único". Los equipos tienen la libertad de elegir la mejor herramienta para resolver sus problemas específicos [19].

e. Escalables

Los microservicios permiten que cada servicio se escale de forma independiente para satisfacer la demanda de la característica de la aplicación que respalda, lo cual permite que nuestra aplicación pueda crecer con mayor rapidez y con un mejor servicio [17].

f. Fácil implementación

Los microservicios permiten la integración y la entrega continua, lo que facilita probar nuevas ideas y revertirlas si algo no funciona. Al ser aplicaciones modulares, su implementación es más ágil y sencilla [25].

g. Reutilizables

La división del software en módulos pequeños y bien definidos les permite a los equipos usar funciones para diferentes propósitos [26].

h. Disponibilidad

Los pequeños servicios tienen menos probabilidades de fallar, ya que implementan solo un método simple y fácilmente comprobable en comparación con módulos grandes y más complejos de un sistema monolítico [26].

2.2. Microservicios y Monolitos

Es útil comparar microservicios con el estilo monolítico: una aplicación construida como una sola unidad.

- En el enfoque monolítico tradicional, la aplicación se escala mediante la clonación de toda la aplicación en varios servidores o máquinas virtuales [21]. En el enfoque de microservicios, la funcionalidad se aísla en servicios más pequeños, por lo que cada servicio puede escalarse de forma independiente [21].

- En el enfoque monolítico, un cambio realizado en una pequeña parte de la aplicación requiere que se reconstruya e implemente todo el monolito [17]. El enfoque de los microservicios permite modificaciones ágiles e iteraciones rápidas de cada microservicio, ya que puede cambiar áreas específicas y pequeñas de aplicaciones complejas, grandes y escalables [21].

- Por otro lado, el enfoque tradicional (datos monolíticos) usado en muchas aplicaciones consiste en una sola base de datos centralizada o solo algunas bases de datos. Suele ser una base de datos SQL normalizada que se usa para toda la aplicación y todos los subsistemas internos. En el enfoque de microservicios, cada microservicio posee sus datos o modelos [21].

- En una aplicación monolítica que se ejecuta en un único proceso, los componentes se invocan entre sí mediante llamadas de función o método de nivel de lenguaje. Pueden estar estrechamente acoplados si se crean objetos con código (por ejemplo, new ClassName()) o pueden invocarse de forma desacoplada si se usa la inserción de dependencias al hacer

referencia a abstracciones en lugar de a instancias de objeto concretas [21]. Una aplicación basada en microservicios es un sistema distribuido que se ejecuta en varios procesos o servicios, normalmente incluso en varios servidores o hosts. Lo habitual es que cada instancia de servicio sea un proceso. Por lo tanto, los servicios deben interactuar mediante un protocolo de comunicación entre procesos como HTTP, AMQP o un protocolo binario como TCP, en función de la naturaleza de cada servicio [21].

- Los monolitos suelen ser de un solo idioma y la tendencia es limitar el número de tecnologías en uso [17]. Al dividir los componentes en servicios, tenemos la opción de construir cada uno de ellos con la herramienta adecuada para el trabajo y, si bien las aplicaciones monolíticas pueden aprovechar diferentes lenguajes hasta cierto punto, no es tan común [17].

Muchos equipos de desarrollo han encontrado que el estilo arquitectónico de microservicios es un enfoque superior a una arquitectura monolítica. Pero otros equipos han descubierto que son una carga que reduce la productividad. Como cualquier estilo arquitectónico, los microservicios traen costos y beneficios. Para tomar una decisión sensata, debe comprenderlos y aplicarlos a su contexto específico [17].

2.3. Métricas

La métrica es una medida o conjunto de medidas cuantitativas (que puede ser un indicador de extensión, cantidad, dimensión, capacidad y/o tamaño) cuyo valor es asociado al grado en que un producto de software posee un atributo especifico, siguiendo reglas claramente definidas [27] [28].

2.3.1. Componentes de una métrica

La medición se define como el proceso por el cual se asignan números o símbolos a atributos o características de entidades del mundo real de tal forma que los describa de acuerdo con reglas claramente definidas [29]. Por tanto, al definir una métrica, debemos especificar al menos 3 elementos:

- ***Entidad***: La entidad sobre la que se mide, puede ser un producto tangible (por ejemplo, la plataforma o sistema), intangible (por ejemplo, un diseño) o un proceso (por ejemplo, la tutorización de alumnos) [30].

- ***Atributo:*** El atributo o característica que se mide sobre dicha entidad, por ejemplo, la adaptabilidad de uso (atributo) de un contenido (entidad) [30].

- **Reglas:** Reglas claras, bien definidas y consistentes de asignación de números o símbolos para cualquier caso de entidades [30].

2.3.2. Tipos de mediciones

Cuando existen relaciones complejas entre atributos, o cuando un atributo debe medirse combinando varios de sus aspectos, entonces se necesita combinar las medidas relacionadas. Por esta razón, se distinguen las medidas directas e indirectas.

Directas

La medición directa de un atributo de entidad no implica ningún otro atributo o entidad. Por ejemplo, podemos medir la longitud de un objeto físico sin ningún otro objeto [31].

Indirectas

Las medidas indirectas son medidas de un atributo obtenidas al comparar diferentes medidas [31], por ejemplo: el número de defectos dividido por el tamaño del módulo proporciona la densidad de defectos del módulo.

2.3.3. Escalas de medición

Las escalas de medición son las asignaciones utilizadas para representar el sistema de relaciones empíricas. Es principalmente de 5 tipos:

1) *Nominal*

Una escala nominal coloca a cada entidad en una categoría particular, según el valor de los atributos. Es el mismo proceso cuando identificamos un lenguaje de programación [31]. En otras palabras, la medición a escala nominal coloca elementos en un esquema de clasificación. Las clases no están ordenadas; incluso si las clases están numeradas del 1 al n para su identificación, no hay un orden implícito de las clases.

2) *Ordinal*

Una escala ordinal clasifica los elementos en un orden, como cuando asignamos a las fallas una gravedad progresiva como menor, mayor y catastrófica [31].

3) *Intervalo*

Una escala de intervalo define una distancia de un punto a otro, de modo que no son intervalos iguales entre números consecutivos [31]. Esta propiedad permite cálculos que no están disponibles con la escala ordinal, como calcular el valor medio.

4) *Porcentaje*

La escala con más información y flexibilidad es la escala de razones, que incorpora un cero absoluto, conserva las razones y permite los más sofisticados análisis [31]. Medidas como líneas de códigos o números de defectos son medidas de relación. Es por esta escala que podemos decir que A es el doble del tamaño de B [31].

5) Absoluto

La escala absoluta de medición es la escala más restrictiva. Para dos medidas cualesquiera, M y M', solo hay una transformación admisible: la transformación de identidad. Por lo tanto, solo hay una forma en la que se puede realizar la medición, por lo que M y M' deben ser iguales [31].

Las características principales y un ejemplo se pueden consultar en el Anexo A

En la Tabla 1 se enumeran diferentes tipos de datos que se identifican en métricas, descripción y posibles operadores presentes en ellas [32].

Tabla 1.- Tipos de datos medidos

Tipo de datos	Posibles operadores	Descripción de datos
Nominal	=, ≠	Categorías
Ordinal	<, >	Clasificación
Intervalo	+, –	Diferencias
Proporción	/	Cero absoluto

2.3.4. Características

Dentro de las características de las buenas métricas, se describen algunas de los atributos deseables [32]:

- *Válida*

  Claramente relacionado con la característica que se está midiendo, p. Ej. Aumenta monótonamente a medida que aumenta la característica.

- *Objetiva*

Independiente de la opinión personal.

- *Reproducible*

Las mediciones se pueden repetir de forma constante.

- *Precisa*

Sensible a los cambios en la característica medida.

- *Robusta*

No se manipula fácilmente ni es sensible a factores externos.

- *Comparable*

Altamente correlacionado con otras métricas que miden la misma característica.

- *Universal*

Se puede traducir en submétricas para las partes inferiores del producto o proceso.

2.4. Granularidad

La granularidad de los servicios es un aspecto importante al diseñar arquitecturas orientadas a microservicios. Tener una correcta granularidad nos permite aprovechar al máximo las ventajas de dicha arquitectura.

Existen limitaciones en cuanto a tener mecanismos establecidos para definir el número de funciones que se deben agrupar en un servicio, o la cantidad de código que debe contener cada una, pero, sí existen algunas pautas de utilidad [33].

- Descomponer por capacidad comercial y definir los servicios correspondientes a las capacidades comerciales.
- Descomponer por subdominio de diseño controlado por dominio.
- Descomponer por verbo o caso de uso y definir los servicios que son responsables de acciones particulares. Por ejemplo, “Shipping Service” que es responsable de enviar pedidos completos.
- Descomponer por sustantivos o recursos definiendo un servicio que sea responsable de todas las operaciones en entidades/recursos de un tipo determinado, por ejemplo, un “Account Service” que es responsable de administrar las cuentas de usuario.
- Un servicio debe manejar un único dominio, es decir solo realizar un tipo de actividad.

Por ello, uno de los problemas fundamentales de la transición a microservicios es la finalización de su nivel de granularidad [34]. Los marcos de clasificación del lenguaje de definición de arquitectura (ADL) [35] indican que es necesario modelar los aspectos estructurales/topológicos y de comportamiento de una arquitectura.

Entonces, ¿Cuáles son los enfoques de modelado que se utilizan para definir la granularidad de un microservicio? [8].

Los enfoques estructurales capturan la topología y/o las dependencias entre las unidades de construcción de la arquitectura de microservicios. Por otro lado, los enfoques conductuales capturan las acciones de estas unidades en varios escenarios de tiempo de ejecución en los que opera el microservicio modelado [8].

Por lo tanto, se requiere un enfoque orientado a la arquitectura para microservicios que facilite el razonamiento sobre las necesidades de granularidad que permitan capturar los aspectos estructurales y de comportamiento de la arquitectura [8].

Una forma de *modelar los aspectos estructurales y de comportamiento de una arquitectura,* es mediante atributos estructurales, como el acoplamiento, que se pueden medir para predecir y determinar la calidad de la arquitectura al principio de la fase de diseño [9]**.**

Los atributos estructurales se refieren a qué tan bien está desarrollado el software y pueden medirse directamente. Sin embargo, no tienen en cuenta qué tan bien un sistema satisface sus necesidades [36]. A continuación, se enumeran tres importantes atributos estructurales:

- Acoplamiento
- Cohesión
- Complejidad

Atributos de calidad

Los atributos de calidad se utilizan para describir el rendimiento, reusabilidad, etc. de un sistema, aunque solo se pueden medir una vez que el sistema está completamente desarrollado [37]. Existen un gran número de atributos de calidad para medir software, pero cuando se habla de granularidad de microservicios, se puede reducir la lista (Ilustración 4) [8].

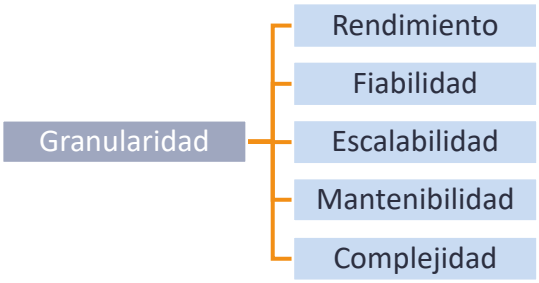


Ilustración 4. Atributos relacionados con granularidad

A continuación se describe cada uno ellas:

- *Rendimiento*
  "El grado en que un sistema o componente cumple sus funciones designadas dentro de las restricciones dadas, como velocidad, precisión o uso de la memoria" [38].
  Entre los medios para capturar este atributo de manera objetiva se incluyen el tiempo de respuesta, el rendimiento, la duración de la invocación, la identificación de los obstáculos de desempeño y/o la duración de la transacción [8].

- *Fiabilidad*
  "Un conjunto de atributos que se relacionan con la capacidad del software para mantener su nivel de rendimiento en condiciones establecidas durante un período de tiempo establecido" [39].
  La fiabilidad implica exhibir características como: tolerancia a fallas, recuperación ante desastres, resiliencia, eliminando puntos únicos de falla y/o robustez [8].

- *Escalabilidad*

"Es la capacidad del sistema para manejar aumentos de carga sin disminuir el rendimiento, o la posibilidad de aumentar rápidamente la carga" [3].
Exhibir escalabilidad implica emplear estrategias como auto escalado, balanceo de carga y/o distribución de carga. Se puede medir la escalabilidad objetivamente observando el número de transacciones completadas por segundo [8].

- *Mantenibilidad*
  "La facilidad con la que se puede modificar un sistema o componente de software para cambiar o agregar capacidades, corregir fallas o defectos, mejorar el rendimiento u otros atributos, o adaptarse a un entorno modificado" [38].
  Exhibir mantenibilidad implica exhibir adaptabilidad, capacidad de cambio, capacidad de expansión y/o dinamismo. Estas propiedades se capturan objetivamente en términos de costo de mantenimiento, gastos generales de mantenimiento y/o costo de esfuerzo [8].

- *Complejidad*
  "El grado en el que el diseño o el código de un sistema es difícil de entender debido a los numerosos componentes o relaciones entre los componentes" [38].
  Minimizar la complejidad implica seguir 2 principios de diseño cruciales: acoplamiento estrecho y/o baja cohesión. Esto se mide en términos de comunicación, costo de complejidad y/o de desarrollo.

Los atributos de calidad descritos anteriormente se utilizaron como apoyo en la selección de la métrica adecuada para realizar la adaptación.

CAPÍTULO

03

Implementación de la solución

En este capítulo se presenta el análisis de las métricas, el proceso de selección y la interpretación para su aplicación en otra arquitectura.

3.1. Criterios de selección

Alcanzar un nivel de calidad deseado es fundamental para todo el desarrollo de software. En consecuencia, hay un gran cuerpo de investigación en el área de medición de software en el desarrollo Orientado a Objetos, lo que resulta en un gran número de métricas que se han propuesto para medir varios aspectos de los atributos de calidad internos (estructurales) del software [36].

Existe un gran número de métricas que se han propuesto para medir varios aspectos de los atributos de calidad, por lo que, para seleccionar una métrica se determinaron los criterios necesarios de selección para la adaptación, los cuales se listan a continuación:

- *C1.- Atributo estructural*

  La métrica debe medir un atributo estructural. Los atributos estructurales se refieren a qué tan bien está desarrollado el software, en sus dimensiones de: acoplamiento, cohesión y complejidad.

- *C2.- Relación con Granularidad*

  La métrica debe medir uno de los atributos relacionados con la granularidad de microservicios [8].

- *C3.- Aplicación*

  La métrica debe ser aplicable a un microservicio, es decir, debe considerar las características de éstos.

- *C4.- Documentación*

  Debido al tiempo establecido para el desarrollo de la investigación, la métrica debe contar con suficiente información de su definición, utilización y resultados.

3.2. Selección de métrica

La selección de la métrica se realizó con una compilación de las métricas orientadas a objetos propuestas en diferentes trabajos de investigación por diferentes autores. Durante la revisión de la literatura se consideraron métricas propuestas por Chidamber y Kemerer *(C&K)* [40], Lorenz y Kidd *(L&K)* [26] y Abreu y Melo *(A&M)* [41].

La lista de métricas recopiladas se muestra en la Tabla 2.

Tabla 2. Métricas estudiadas

Nombre	Abreviación	Atributo	Autor	Referencia
Coupling Between Objects	CBO	Acoplamiento	*C&K*	[40]
Coupling Factor	COF	Acoplamiento	*A&M*	[41]
Lack of Cohesion in Methods	LCOM	Cohesión	*C&K*	[40]
Response For a Class	RFC	Complejidad	*C&K*	[40]
Weighted Methods per Class	WMC	Complejidad	*C&K*	[40]
Cyclomatic Complexity		Complejidad	*McCabe*	[42]
Method Hiding Factor	MHF	Encapsulamiento	*A&M*	[41]
Attribute Hiding Factor	AHF	Encapsulamiento	*A&M*	[41]
Method Inheritance Factor	MIF	Herencia	*A&M*	[41]
Attribute Inheritance Factor	AIF	Herencia	*A&M*	[41]
Specialisation Index per Class	SIX	Herencia	*L&K*	[26]
Depth of Inheritance Tree	DIT	Herencia	*C&K*	[40]
Polymorphism Factor	POF	Polimorfismo	*A&M*	[41]
Number of Children	NOC	Reutilización	*C&K*	[40]
Lines of Code per method	LOC	Tamaño	*L&K*	[26]
Number of Messages Send	NOM	Tamaño	*L&K*	[26]

Criterio 1

Describe que la métrica debe medir un atributo estructural, sin embargo, sucede que al ser un atributo de calidad interno no puede medir directamente la calidad de un producto; si no que se utiliza como parte de la medición para atributos externos, por ejemplo, la fiabilidad, la eficiencia, etc.

Su aplicación en las métricas estudiadas da como resultado la Tabla 3.

Tabla 3. Métricas seleccionadas con el criterio 1

Nombre	Abreviación	Atributo	Autor	Referencia
Coupling Factor	COF	Acoplamiento	*A&M*	[41]
Lack of Cohesion in Methods	LCOM	Cohesión	*C&K*	[40]
Response For a Class	RFC	Complejidad	*C&K*	[40]
Weighted Methods per Class	WMC	Complejidad	*C&K*	[40]
Cyclomatic Complexity		Complejidad	*McCabe*	[42]
Method Hiding Factor	MHF	Encapsulamiento	*A&M*	[41]
Attribute Hiding Factor	AHF	Encapsulamiento	*A&M*	[41]
Method Inheritance Factor	MIF	Herencia	*A&M*	[41]
Attribute Inheritance Factor	AIF	Herencia	*A&M*	[41]
Specialisation Index per Class	SIX	Herencia	*L&K*	[26]
Depth of Inheritance Tree	DIT	Herencia	*C&K*	[40]
Polymorphism Factor	POF	Polimorfismo	*A&M*	[41]
Lines of Code per method	LOC	Tamaño	*L&K*	[26]
Number of Messages Send	NOM	Tamaño	*L&K*	[26]

Criterio 2

Describe que la métrica debe medir algún atributo relacionados con la granularidad de microservicios, es decir, con todos aquellos atributos que en conjunto puedan determinar la comunicación ideal de un microservicio.

La aplicación del criterio 2 da como resultado la Tabla 4.

Tabla 4. Métricas seleccionadas con el criterio 2

Nombre	Abreviación	Atributo	Autor	Referencia
Response For a Class	RFC	Complejidad	*C&K*	[40]
Weighted Methods per Class	WMC	Complejidad	*C&K*	[40]
Cyclomatic Complexity		Complejidad	*McCabe*	[42]
Lines of Code per method	LOC	Tamaño	*L&K*	[26]
Number of Messages Send	NOM	Tamaño	*L&K*	[26]

Criterio 3

Describe que la métrica debe ser aplicable a un microservicio, es decir, debe considerar las características similares con la de los microservicios.

La aplicación del criterio 3 da como resultado la Tabla 5.

Tabla 5. Métricas seleccionadas con el criterio 3

Nombre	Abreviación	Atributo	Autor	Recurso
Response For a Class	RFC	Complejidad	*C&K*	[40]
Lines of Code per method	LOC	Tamaño	*L&K*	[26]
Number of Messages Send	NOM	Tamaño	*L&K*	[26]

Criterio 4

Describe que la métrica debe contar con suficiente información de su definición, utilización y resultados.

La aplicación del criterio 4 da como resultado la Tabla 6.

Tabla 6. Métricas seleccionadas con el criterio 4

Nombre	Abreviación	Atributo	Autor	Recurso
Response For a Class	RFC	Complejidad	*C&K*	[40]
Lines of Code per method	LOC	Tamaño	*L&K*	[26]

El resultado de aplicar los criterios de selección es de dos métricas, por lo que se aplicó un último criterio que consideró el tiempo de la investigación, entendimiento de la métrica, facilidad de adaptación y mejor relación con la arquitectura de microservicios.

Tabla 7. Métrica seleccionada

Nombre	Abreviación	Atributo	Autor	Recurso
Response For a Class	RFC	Complejidad	*C&K*	[40]

3.3. Métrica seleccionada

Response For a Class

Esta métrica cuenta las ocurrencias de llamadas a otras clases de una clase en particular. En otras palabras, el conjunto de todos los métodos que se pueden invocar en respuesta de un mensaje enviado a un objeto de la clase) [40]. Chidamber y Kemerer (*C&K*) describen RFC como una indicación de la complejidad de la clase (por lo tanto, un reflejo del esfuerzo de prueba requerido) [40].

C&K expresan RFC en [39] de la siguiente forma:
Definición

$$RFC = |\, RS \,|$$

Donde:
RS = Es el conjunto de respuestas para la clase.

Bases teóricas
El conjunto de respuestas para la clase se puede expresar como:

$$RS = \{\, M \,\} \cup_{all\ i} \{\, R_i \,\}$$

Donde:
$\{\, R_i \,\} =$ conjunto de métodos llamados por el método *i*
$\{\, M \,\} =$ conjunto de todos los métodos en la clase.

El conjunto de respuestas de una clase es un conjunto de métodos que potencialmente pueden ejecutarse en respuesta a un mensaje recibido por un objeto de esa clase. La cardinalidad de este conjunto es una medida de los atributos de los objetos de la clase. Dado que incluye específicamente métodos llamados desde fuera de la clase, también es una medida de la comunicación potencial entre la clase y otras clases [40].

Puntos de vista:

1) Si se puede invocar una gran cantidad de métodos en respuesta a un mensaje, la prueba y depuración de la clase se vuelve más complicada ya que requiere un mayor nivel de comprensión por parte del probador [40].

2) Cuanto mayor sea el número de métodos que se pueden invocar desde una clase, mayor será la complejidad de la clase [40].

3) Un valor en el peor de los casos para posibles respuestas ayudará en la asignación adecuada del tiempo de prueba [40].

La definición de *C&K* se enfoca a todas aquellas aplicaciones monolíticas orientadas a objetos que poseen todas las características de las clases, sin embargo, aspectos interesantes como la posibilidad de encontrarnos con clases que son independientes (es decir, sin padres, sin hijos, sin acoplamiento), la variedad del lenguaje, estructura e interacción brindan puntos a examinar en el diseño de las jerarquías [40].

Ejemplo [43]

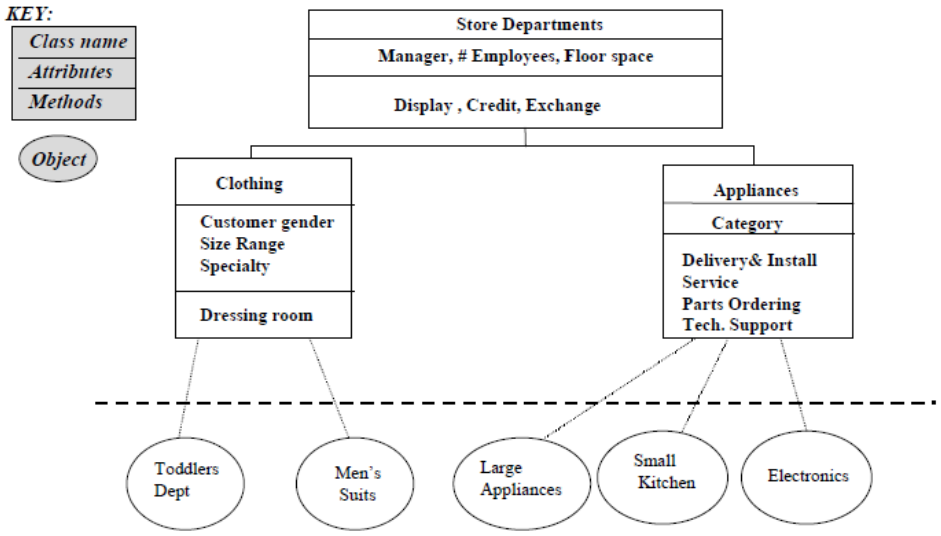


Ilustración 5. Ejemplo de aplicación de clases [43]

El RFC para Store_dept en Ilustración 5 es el número de métodos que se pueden invocar en respuesta a mensajes por sí mismo (*Store_dept*), por *Clothing_dept* y por *Appliance_dept*.

RFC(Store_dept) = 3 (*El mismo*) + 1 (*Clothing_dept*) + 4 (*Appliance_dept*) = 8

RFC se aplica en arquitecturas monolíticas identificando las clases y los métodos que interactúan cuando se realizan una llamada, tal como se observa en el ejemplo anterior. En las arquitecturas de microservicios la identificación de las clases y los métodos que realizan llamadas se identifican pero no es tan simple de medir.

- Las solicitudes que se generan a un microservicio se realizan a partir de una API, por la que únicamente se pueden realizar solicitudes a otros microservicios o bases de datos, por lo que no se realizan solicitudes hacia el mismo.
- El idioma en el que esté construido no siempre será el mismo, variando en sintaxis, paradigma y objetivo. La variación de esto no permitirá medir siempre con las características de RFC.
- Los microservicios interactúan a través de interfaces en la API, lo que impide entrar directamente a métodos de otro, independientemente si son públicos o privados.

3.4. Adaptación de la métrica

La métrica resultante de la adaptación de RFC es una métrica a nivel de diseño, misma que pretende medir respuestas entre los sistemas orientados a microservicios.

Los valores obtenidos de la aplicación de la métrica se podrán utilizar como indicadores/predictores tempranos de las características de calidad del software como acoplamiento, cohesión y complejidad, lo que permitiría identificar problemas potenciales en las primeras fases del ciclo de vida del desarrollo del software.

Así mismo, la métrica RFC (Response For a Class) de C&K [40] es utilizada como base para desarrollar la adaptación de la métrica pretendiendo medir las *Respuestas en la Arquitectura*.

Análisis de características de métricas OO

En esta sección se pretende identificar las diferencias o similitudes que permitieron identificar las variables que se puedan reutilizar o se deban reemplazar de la métrica seleccionada (Tabla 7).

Se analizaron diferentes microservicios obtenidos de repositorios en la web [44, 45, 46], mismos que fueron descargados, revisados y replicados en un servidor local. Adicionalmente, se observaron 5 microservicios más contenidos en repositorios en internet para utilizarse en el análisis [16].

a) Clases

Las clases son una construcción que permite crear tipos propios con personalización a través de la agrupación de variables de otros tipos, métodos y eventos. Así se definen los datos y el comportamiento de un tipo.

En los repositorios analizados, se seleccionaron diferentes microservicios para observar la composición de sus clases, en su estructura interna se puede observar la lógica de un microservicio encapsulada en una misma clase o módulo según sea el lenguaje en el que fue desarrollado.

Por ejemplo, en las ilustraciones 6, 7 y 8, se presentan las clases de tres servicios, cada uno con su propia lógica funcional. En estás se puede observar parte de la definición de los microservicios.

ContactService.java

src > main > java > com > sinbugs > contacts > service > ContactService.java

```java
package com.sinbugs.contacts.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.sinbugs.contacts.dao.ContactRepository;
import com.sinbugs.contacts.dto.Contact;

@Service
public class ContactService {

    @Autowired
    ContactRepository dao;

    public Contact save(Contact contact){
        return dao.saveAndFlush(contact);
    }
}
```

Ilustración 6. Microservicio agenda [46]

movie.js

api-service > src > services > movie.js > ...

```javascript
module.exports = {
    /**
     *
     * @param {object} data
     * @returns {Promise<void | Error>}
     */
    createMovie(data){ ···
    },

    getAllMovies(){ ···
    },

    getMovieById(id){ ···
    },

    getTrailer(title, year){ ···
    },

    createOrder(data){ ···
    },

    getAllOrders(){ ···
    }
};
```

Ilustración 7. Microservicio películas [44]

```
13
14  @Service
15  public class StationServiceImpl implements StationService {
16
17      @Autowired
18      private StationRepository repository;
19
20      String success = "Success";
21
22      @Override
23      public Response create(Station station, HttpHeaders headers) {
24          if (repository.findById(station.getId()) == null) {
25              station.setStayTime(station.getStayTime());
26              repository.save(station);
27              return new Response<>(1, "Create success", station);
28          }
29          return new Response<>(0, "Already exists", station);
30      }
31
32
33      @Override
34      public boolean exist(String stationName, HttpHeaders headers) {···
40      }
41
42      @Override
43      public Response update(Station info, HttpHeaders headers) {···
53      }
```

Ilustración 8. Microservicio estación [45]

En cada una de las ilustraciones anteriores se encontraron similitudes, en el caso de ciertos lenguajes, por ejemplo Java, C#, Python o PHP en donde las clases están delimitadas por la palabra reservada "Class", delimitada por llaves o una sangría (Ilustración 6 y 8), mientras que en otros casos se define la lógica en funciones que son exportadas para su utilización por el resto del microservicio (Ilustración 7). En cada clase o modulo exportado se encuentran contenidos los métodos/funciones utilizados para resolver las solicitudes.

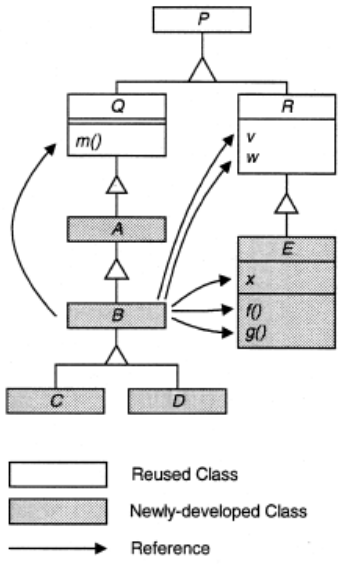


Ilustración 9. Un ejemplo para la medición de RFC [47].

Ejemplo de valor de medición de métrica revisada. Estos valores de métrica se miden para la clase B en la Ilustración 9. Las relaciones son las entidades que se cuentan en cada medición de métrica [47].

Métrica	***Valor***	***Relación***
RFC	*3*	*{f de E, g de E, m de Q}* [47]

La métrica *RFC* es aplicada en arquitecturas orientadas a objetos para medir la comunicación entre clases (Ilustración 9). En la arquitectura de microservicios se observó que su libertad tecnológica da pauta a tener lógica interna definida en métodos, funciones o módulos.

En resumen, se puede decir que:

1. La comunicación entre clases es similar entre microservicios de un sistema, en algunos se crean clases de objetos del dominio, mientras que en otras, se crean clases de implementación de servicio, que se utilizan para realizar nuevas peticiones a otros microservicios o base de datos.

2. En muchos de los métodos de una clase en arquitecturas orientadas a objetos, la comunicación es síncrona, sin embargo, en una aplicación monolítica completa se pueden ejecutar procesos de forma asíncrona. En microservicios, la comunicación puede ser síncrona o asíncrona, por ejemplo, síncrona en procesar la información o asíncronas al publicar notificaciones nuevas en un bus de datos como RabbitMQ.

b) Modelo de datos

De igual forma, existen clases que son utilizadas como modelos de objetos para manejar información de una forma más sencilla. Por ejemplo, en las ilustraciones 10 y 11 se pueden observar clases que son utilizadas como modelos de datos, contiene al menos tres aspectos: definición de atributos, constructor y Sets/Gets.

Contact.java ×

c > main > java > com > sinbugs > contacts > dto > Contact.java

```
10  @Entity
11  public class Contact implements Serializable {
12
13      private static final long serialVersionUID = 4894729030347835498L;
14
15      @Id
16      @GeneratedValue (strategy = GenerationType.IDENTITY)
17      private Long id;
18      private String firstName;
19      private String lastName;
20      private String phoneNumber;
21      private String email;
22
23      public Contact(){ }
24
25      public Contact(Long id, String firstName, String lastName, String phoneNumber, String email) {
32      }
33
34      public Long getId() {···
36      }
37
38      public void setId(Long id) {···
40      }
```

Ilustración 10. Clase contacto [46]

Station.java M

ts-station-service > src > main > java > fdse > microservice > entity > Station.java

```java
@Data
@Document(collection="station")
public class Station {
    @Valid
    @NotNull
    @Id
    private String id;

    @Valid
    @NotNull
    private String name;

    private int stayTime;

    public Station(){…
    }

    public Station(String id, String name) {…
    }

    public Station(String id, String name, int stayTime) {…
    }
```

Ilustración 11. Definición estación [45]

En la ilustración 12, únicamente observamos una estructura de acuerdo al contexto que es utilizado, en este caso en específico, la base de datos.

movie.js

movie-service > src > models > movie.js > ...

```javascript
const Sequelize = require('sequelize');

/**
 *
 * @param db
 * @returns {Model}
 */
module.exports = function (db) {
    return db.define('movie', {
        title: {type: Sequelize.STRING, allowNull: false},
        imdbID: {type: Sequelize.STRING, unique: true, allowNull: false},
        hall: {type: Sequelize.TINYINT, allowNull: false},
        date: {type: Sequelize.DATE, allowNull: false},
        year: {type: Sequelize.STRING, allowNull: false}
    });
};
```

Ilustración 12. Definición de película [44]

En las figuras anteriores se ve la definición de objetos utilizados en el microservicio, mismos que apoyan en la respuesta de solicitudes.

En resumen con respecto a la organización de las clases, se puede decir que:

1. Las clases únicamente se utilizan para crear objetos y acceder a su información.
2. La mayoría de los métodos de una clase son de definición.

c) Comunicación

Como ultima consideración, los puntos de acceso a los microservicios, cada uno cuenta con rutas de acceso, cada ruta de acceso es específica para atender la funcionalidad para la que fue diseñado el microservicio, como se observa en la Ilustración 13.

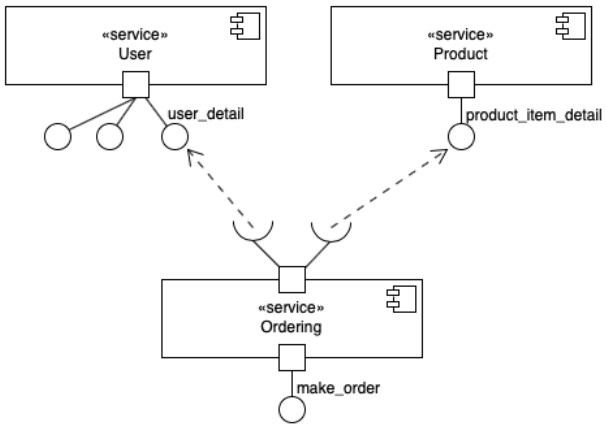


Ilustración 13. Ejemplo de comunicación entre microservicios [48]

Se ha definido un microservicio como un proceso coherente e independiente que interactúa a través de mensajes con otros, si bien se puede ver como un componente lógico que brinda funcionalidad a los consumidores de servicios a través de una interfaz. La interfaz de servicio es a menudo una API web, una interfaz programática accesible a través de solicitudes HTTP(s). Se accede a la API de un servicio a través de un *endpoint* en una API, que son las URL de un API que responden a una petición, se encuentran en una ubicación direccionable por red dentro del entorno de ejecución [19].

La API de un servicio puede tener más de un *endpoint*, en la Ilustración 14 se observa un microservicio con dos puntos de acceso por los que puede resolver peticiones.

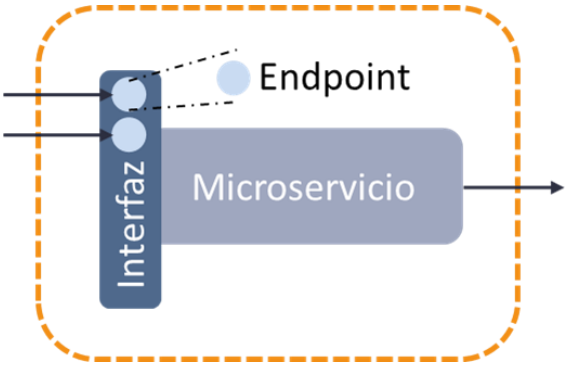


Ilustración 14. Ejemplo interfaz en un microservicio

Una solicitud de API es un mensaje enviado a un punto final de API que activa la ejecución del servicio, y una respuesta de API es un mensaje enviado a cambio para comunicar el resultado de la ejecución del servicio.

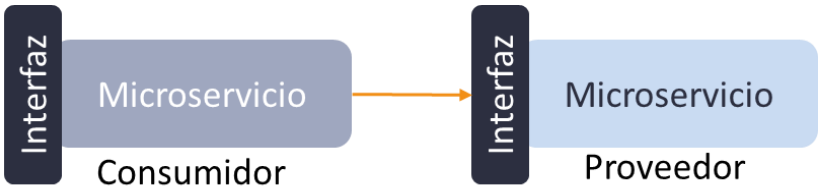


Ilustración 15. Ejemplo de solicitud a microservicio

Un componente que envía una solicitud de API asume el rol de consumidor de API, mientras que el servicio que recibe la solicitud de API y envía la respuesta de API al consumidor asume el rol de proveedor de API, como se observa en la Ilustración 15.

Un servicio puede desempeñar el papel tanto de consumidor de API como de proveedor de API, según el contexto del mensaje (Ilustración 16). Ambos roles también pueden ser desempeñados por componentes distintos de los servicios.

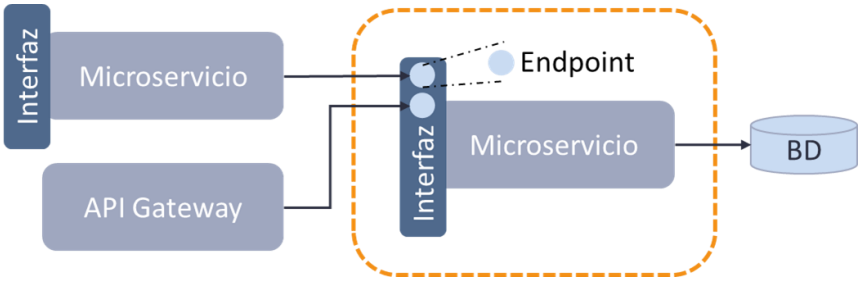


Ilustración 16. Solicitudes reutilizando clases

En un entorno real, los *endpoints* se verían como en las Ilustraciones 17, 18 y 19 para diferentes microservicios.

ContactsApi.java

src > main > java > com > sinbugs > contacts > api > ContactsApi.java

```java
import com.sinbugs.contacts.dto.Contact;
import com.sinbugs.contacts.service.ContactService;

@RestController
public class ContactsApi {

    @Autowired
    ContactService contactService;

    @Autowired
    Mapper mapper;

    @RequestMapping(value="/contact", method=RequestMethod.POST)
    public ContactResponse updateOrSave(@RequestBody @Valid ContactRequest ContactRequest){…
    }

    @RequestMapping(value="/contact", method=RequestMethod.GET)
    public Contact getById(){…
    }
}
```

Ilustración 17. Endpoint microservicio contacto [46]

api-service > src > routes > JS index.js > ...

```javascript
const movieController = require('../controllers/movie');
const orderController = require('../controllers/order');

let router = express.Router();

router.get('/', (req, res) => res.status(200).json({text: 'OK'}));

router.post('/movies', (req, res, next) => {…
});

router.get('/movies', (req, res, next) => {…
});

router.get('/movies/:id', (req, res, next) => {…
});

router.get('/movies/:id/trailer', (req, res, next) => {…
});

router.post('/movies/:id/orders', (req, res, next) => {…
});
```

Ilustración 18. Endpoint microservicio películas [44]

StationController.java

ts-station-service > src > main > java > fdse > microservice > controller > StationController.java

```java
import static org.springframework.http.ResponseEntity.ok;

@RestController
@RequestMapping("/api/v1/stationservice")
public class StationController {

    @Autowired
    private StationService stationService;

    private static final Logger LOGGER = LoggerFactory.getLogger(StationController.class);

    @GetMapping(path = "/welcome")
    public String home(@RequestHeader HttpHeaders headers) {…
    }

    @GetMapping(value = "/stations")
    public HttpEntity query(@RequestHeader HttpHeaders headers) {…
    }

    @PostMapping(value = "/stations")
    public ResponseEntity<Response> create(@RequestBody Station station, @RequestHeader HttpHeaders headers) {…
    }
```

Ilustración 19. Endpoint microservicio estación [45]

Si bien se ha venido hablando de las diferencias entre los lenguajes en los que están construidos, la utilización y definición de los *endpoints*, todos ellos mantienen la similitud de URL que hace referencia a los métodos que apoyarán en la respuesta a la solicitud.

La comunicación entre microservicios se lleva a cabo a través de mensajes, si bien puede ser de diferente forma (síncronos o asíncronos), se mantienen siempre presentes algunos elementos importantes como las interfaces de acceso, el origen de la solicitud o bien el rol que toman los componentes involucrados, todos ellos son aspectos interesantes a considerar en microservicios que no están presentes del todo en arquitecturas monolíticas.

Un enfoque de uso común es la mensajería a través de un bus de mensajes ligero. La infraestructura elegida suele ser tonta (tonta porque actúa solo como un enrutador de mensajes): las implementaciones simples como RabbitMQ o ZeroMQ no hacen mucho más que proporcionar una estructura asíncrona confiable: la inteligencia aún vive en los *endpoints* que están produciendo y consumiendo mensajes en los servicios [17].

En un monolito, los componentes se ejecutan en proceso y la comunicación entre ellos se realiza mediante la invocación de métodos o la llamada a funciones. El mayor problema al convertir un monolito en microservicios radica en cambiar el patrón de comunicación. Una conversión ingenua de llamadas a métodos en memoria a RPC conduce a comunicaciones comunicativas que no funcionan bien. En su lugar, debe reemplazar la comunicación de grano fino con un enfoque de grano más grueso [17].

Base teórica

C&K en [40] describen la base teórica para las métricas de Diseño Orientado a Objetos (DOO). RFC se define de la siguiente forma:

Métodos como medidas de comunicación:

En el enfoque orientado a objetos, los objetos se comunican principalmente a través del paso de mensajes. Un mensaje puede hacer que un objeto se "comporte" de una manera particular al invocar un método particular. Los métodos pueden verse como definiciones de respuestas a posibles mensajes. Por lo tanto, es razonable definir un conjunto de respuestas para una clase de objetos de la siguiente manera:

{ Conjunto de respuestas de una clase de objetos }	=	{ Conjunto de todos los métodos que se pueden invocar en respuesta de un mensaje a un objeto de la clase }

Considerando la definición teórica de RFC, se planteó el mismo objetivo de medición en el contexto de microservicios.

1. En la comunicación entre servicios se consideró a todas las interfaces por las que se puede realizar una petición a un servicio. Una forma de definir las solicitudes recibidas por un servicio es como se define en [16], en donde se representa una única clase por la que las solicitudes son recibidas a un servicio identificado por un URI.

   Por lo tanto, podemos definir el conteo de interfaces de la siguiente forma.

{ Conjunto de interfaces definidas en un microservicio }	=	{ El conjunto de todas las solicitudes recibidas por un servicio identificado por una URI }

2. Una solicitud debe resolverse ejecutando diferentes operaciones. Una forma de definir el número de operaciones realizadas para responder una solicitud es identificar el número de métodos/clases ejecutadas para resolverla.

   Por lo tanto, podemos definir el número de operaciones a realizar por solicitud de la siguiente forma:

{ Conjunto de operaciones por solicitud }	=	{ El conjunto de todos aquellos métodos ejecutados para resolver una petición a través de una interfaz }

3. Por último, el conjunto de operaciones por solicitud es la cantidad de clases utilizadas en más de una ocasión al resolver todas las peticiones posibles.

La última consideración mencionada está directamente relacionada con el acoplamiento entre clases, que se puede definir como: “Una medida de la interdependencia entre módulos en un programa de computadora”, según el Estándar Internacional ISO/IEC/IEEE 24765.

Un bajo acoplamiento nos garantiza:

- Mejorar la mantenibilidad de los módulos del software, facilitar los cambios en el software sin tener que revisar todos los módulos dependientes.
- Mejorar la reutilización de las unidades del software.
- Facilitar las pruebas unitarias de cada módulo, al ser más independientes.

Durante las fases de implementación y mantenimiento del desarrollo de sistemas, el conjunto de respuestas no debe cambiar, ya que las interfaces de acceso no pueden ser modificadas.

3.5. Métrica propuesta

Se propone un conjunto de métricas que permite apoyar en la evaluación de la calidad de sistemas orientados a microservicios, utilizando como base la métrica RFC (Response For a Class) de C&K [10], esto al considerar las solicitudes entre microservicios.

Formalización

La formalización de las nuevas métricas consiste en tomar las variables utilizadas en la métrica RFC como punto de inicio para la adaptación. Así mismo, se considera la forma de medir la comunicación de los microservicios en sistemas con esta arquitectura.

Existen diferentes formas de realizar la formalización de conceptos, por ejemplo: teoría de números, teoría de conjuntos , lógica proposicional, etc. Específicamente para formalizar las métricas de software se puede utilizar la teoría de la medición.

El objetivo es contar con una semántica formal, que se pueda utilizar para deducir o definir conceptos del mundo real. En esta investigación se utilizó la teoría de conjuntos para la formalización de las métricas.

RFC

En el enfoque orientado a objetos, los objetos se comunican principalmente a través del paso de mensajes. Un mensaje puede hacer que un objeto se “comporte” de una manera particular al invocar un método particular.

$$RS = \{ M \} \cup_{all\ i} \{ R_i \}$$

Microservicios

En el nuevo modelo de [41], cada microservicio se modela a través de un componente que se puede conectar con otros microservicios usando diferentes puertos. Estos puertos representan las interfaces hacia las funcionalidades requeridas y son proporcionadas y descritas a través de diferentes interfaces que se utilizan como nombres para ellos. Por esto, se define lo siguiente:

Definición 1. Microservicio

Con base en la definición formal de microservicio en [41]. Un microservicio $M = (I, O, R)$ definido como dos conjuntos finitos I, O y una relación binaria $R \subseteq I \times O$, donde:

- $I = \{ i_1, i_2, i_n\}$ es un conjunto finito no vacío de interfaces definidas en el microservicio.
- $O = \{ método_1, método_2, método_n\}$ es un conjunto finito no vacío de métodos.
- La relación binaria R especifica el conjunto de pares ordenados en donde el primer elemento del par es una interfaz $i \in I$ y el segundo elemento del par es un $método \in O$.

Número de Interfaces utilizadas (NIU)

Las llamadas entre microservicios se definieron de la siguiente forma: Un microservicio puede ser utilizado dentro de un sistema. Este elemento es necesario para realizar una petición.

Definición 2. Número de Interfaces Accedidas

De acuerdo a la definición 1.
Sea $W \subseteq I$, donde:

- $W = \{ w_1, w_2, w_n \}$ es un subconjunto finito de todas las interfaces no utilizadas.

Se puede definir NIA, como el conjunto de todas las interfaces utilizadas en un microservicio, expresado como:

$NIA = I - W$

Con la restricción de que:

$NIA \neq W$
$NIA \neq \emptyset$

Con base en lo anterior, una interfaz accedida se puede definir con la siguiente condición:

El valor del atributo de *NIU* está dado por la división del número de interfaces accedidas entre el número de interfaces totales. El cálculo de su valor está expresado en la métrica (1).

$$NIU = \frac{NIA}{I} \quad (1)$$

En donde:

NIA = Número de interfaces accedidas.
I = Número de interfaces totales.

Esta métrica pretende medir el número de interfaces *i* definidas en un microservicio *m*, por las que se pueden realizar solicitudes s.

Ilustración 20. Intervalo de NIU

El valor resultante de la métrica se determina dentro de un rango de 0 a 1 (Ilustración 20), en donde, el peor valor se acerca a 0, esto indicaría que hay una tendencia a tener muchas interfaces de acceso que no son utilizados para la comunicación con otros microservicios y/o componentes. Y el mejor valor se acerca 1, esto indicaría que el microservicio tiene las interfaces necesarias para cumplir con su capacidad empresarial.

Número de Operaciones (NO)

Las llamadas entre microservicios se definieron de la siguiente forma. Como se muestra a continuación, para que un microservicio pueda ser utilizado dentro un sistema, la interfaz definida debe tener al menos 1 *endpoint*. Este elemento es necesario para realizar un petición.

Definición 3. Número de Operaciones Utilizadas

De acuerdo a la definición 1.
Sea $V \subseteq O$, donde:

- $V = \{v_1, v_2, v_n\}$ es un subconjunto finito de todas los métodos no utilizados.

Se puede definir MU, como el conjunto de todos los métodos utilizados en un microservicio, expresado como:

$MU = O - V$

Con la restricción de que:

$MU \neq V$
$MU \neq \emptyset$

Esta métrica pretende medir el número de operaciones *o* realizadas para resolver una solicitud *s* realizada a un microservicio *m*.

El valor del atributo de *NO* está dado por la división del número operaciones accedidas entre el número de operaciones totales. El cálculo de su valor esta expresado en la métrica (2).

$$NO = \frac{MU}{O} \quad (2)$$

En donde:

MU = Número de operaciones accedidas.
O = Número de operaciones totales.

Ilustración 21. Intervalo de NO

A medida que esta razón tienda a 0, existen un mayor problema (Ilustración 21), indicaría que hay una tendencia a tener muchas funciones/métodos que no son utilizados para resolver alguna solicitud, mientras que si la razón tienda a 1, existirán los métodos necesarios para resolver una petición

CAPÍTULO

04

Pruebas experimentales

En este capítulo se presentan las pruebas y evaluación de las métricas desarrolladas, en donde, su principal objetivo es: proporcionar indicadores de calidad en sistemas con arquitectura orientada a microservicios.

4.1. Plan de pruebas

A continuación, se detallan las pruebas programadas para un sistema con arquitectura de microservicios.

4.1.1. Propósito

El plan de pruebas descrito a continuación tiene el propósito de mostrar la planificación, estructura y documentación de las pruebas realizadas en las métricas definidas en esta investigación, su aplicación y los resultados.

4.1.2. Identificador

A continuación, se observa la nomenclatura que se utiliza en la identificación de los diferentes documentos que serán generados en las diferentes pruebas.

MMic - TD - XX

En donde, podemos observar la composición de cada una de las partes en la Tabla 8.

Tabla 8. Identificadores de nomenclatura

Nomenclatura	Identificador	Descripción
MMic	Arquitectura	Métricas para Arquitecturas Orientadas a Microservicios
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

4.1.3. Documentación

La documentación desarrollada para las pruebas, se encuentra detallada en los siguientes documentos:

- MMic-DP-XX: Especificación de diseños de pruebas.
- MMic-CP-XX: Especificación de casos de pruebas.

Descripción
La descripción de las pruebas se muestra a continuación.

Elementos de pruebas

En la Tabla 9 se muestran las métricas como elementos de prueba que serán evaluados para comprobar su correcto funcionamiento, mismas que cumplen con los requerimientos mínimos en su definición.

Tabla 9. Elementos de prueba

Id	Métrica	Descripción
M1	*NIU*	Número de Interfaces Utilizadas, mide el grado en que el número de interfaces definidas en un microservicio son utilizadas, está dado por la división del número de Interfaces accedidas entre el número de interfaces totales.
	Intervalo: El valor resultante de la métrica se determina dentro de un rango de 0 a 1, en donde, el peor valor se acerca a 0.	
M2	*NO*	Número de Operaciones, mide el grado en que el número de métodos definidos en un microservicio son utilizados, está dado por la división del número operaciones utilizadas entre el número de operaciones totales.
	Intervalo: El valor resultante de la métrica se determina dentro de un rango de 0 a 1, en donde, el peor valor se acerca a 0.	

Repositorios de microservicios

Para aplicar las pruebas de las métricas se replicaron repositorios de internet en un equipo local, cada uno con un dominio diferente en entornos diferentes. En la Tabla 10, se definen los repositorios en donde fueron aplicados los elementos de prueba.

Tabla 10. Sistemas de pruebas

Sistema	Descripción
Agenda	El sistema es resultado de una consultoría en implementación de arquitecturas de microservicios. En esta se realiza un pequeño taller práctico en el que un equipo de desarrollo que suele trabajar con JavaEE, aprendió a construir un microservicio sencillo con Spring Boot. El objetivo fue construir un servicio web REST con el Framework Spring y una base de datos relacional, con ello se desarrolló un servicio web para la administración de información de contactos, como nombre, teléfono, email, etc. [46].

Boletos de Cine	Una aplicación para reservar entradas en el cine. Implementado en la arquitectura de microservicios, utilizando Docker y *docker-compose*. Muestra el funcionamiento básico para vender boletos para las funciones de un cine. Los detalles de las películas se obtienen de OMDb (Open Movie Database) y TMDb (The Movie Database) a través de una API externa. Los pedidos se almacenan en la base de datos local. Y las reservas se implementan mediante el uso de Socket.io almacenándolas en la memoria en api-gateway [44].
Boletos de Tren	El proyecto es un sistema de reserva de billetes de tren basado en una arquitectura de microservicios que contiene 41 microservicios. Muestra el funcionamiento básico para vender boletos de tren, detalle de las rutas, reserva de asientos, venta de comida e itinerarios de los viajes [45].

4.1.4. Criterios de aceptación y suspensión

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 obtuvo un valor con el cálculo manual de las métricas que deberá estar dentro del intervalo de valores de la escala establecidos en la definición de la métrica.

Criterios de suspensión
El criterio de suspensión se basa en el incumplimiento de alguna instancia, con la obtención de valores fuera del rango establecido (de 0 a 1) o imposibles de calcular. Cada vez que un caso no-pasa la prueba, inmediatamente se procederá a evaluar y documentar las condiciones en el que se presentó, permaneciendo en las pruebas hasta concluirlas en su totalidad.

4.1.5. 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.

4.1.6. 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.

4.1.7. 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

4.1.8. Condiciones de aplicación

En la Tabla 11 se muestran los requisitos tecnológicos mínimos para ejecutar las pruebas de las métricas descritas en esta investigación.

Tabla 11. Requisitos de aplicación

Ambiente	
Hardware	**Software**
• Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz. • 8.00 GB (7.89 GB utilizable)	• Sistema operativo Windows 7, 8, 10. • Docker Desktop 2.5 • Visual Studio Code (O algún editor de código) • MySQL (v8.0) • Node.js • Navegador web • Java (jre1.8.0_261) • Java (jdk1.8.0_261) • Spring Tool Suite 4 • MySQL (Community v8.0) • Postman (v7.34)

4.1.9. Personal para aplicación

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

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

4.1.10. Riesgos y contingencias

A continuación, se enlistan los riesgos y contingencias para la ejecución del presente plan de pruebas.

Tabla 12. Riesgos y contingencias

No.	Riesgo	Contingencia
1	No disponibilidad del ambiente de trabajo.	Se debe replicar el entorno adecuado en otro equipo.
2	Planeación incorrecta en el diseño de los casos de prueba.	Ajustar 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 proyecto/replicarlo en otro quipo.
4	El intervalo definido para las métricas están fuera de los rangos obtenidos.	Documentar las anomalías para un posible ajusté en futuras investigaciones.

4.1.11. Aprobación

El plan de pruebas fue revisado por el Dr. Juan Carlos Rojas Pérez.

4.2. Diseño de Pruebas

Las pruebas que se proponen a continuación buscan validar las métricas desarrolladas: *NIU* (Número de Interfaces Utilizadas) y *NO* (Número de Operación Utilizadas), con el objetivo de verificar que cumplan con la definición, elementos e información necesaria para ser aplicadas en sistemas desarrollados en arquitecturas orientadas a microservicios.

Propósito
Especificar el diseño de las pruebas implementadas con diferentes casos de prueba de esta investigación para determinar si son aceptados.

Especificación
Durante la elaboración del plan de pruebas se realizó un diseño de prueba. la Tabla 13 presenta el diseño incluido en el plan de pruebas. Mismo que fue utilizado en la realización de todas las pruebas, ya que, el objetivo perseguido es el mismo.

MMic-DP-01
Diseño de pruebas para las métricas NO y NIU.

Tabla 13. Diseño de prueba MMic-DP-01

<table>
<tr><th colspan="2">MMic-DP-01</th></tr>
<tr><td colspan="2">Nombre:
Evaluación de métricas.</td></tr>
<tr><td colspan="2">Objetivo:
Evaluar la métrica NO y NIU.</td></tr>
<tr><td colspan="2">Características a probar:
• Los valores esperados deben estar dentro del intervalo de valores de la escala. NO >= 0; NO <= 1; NIU >= 0; NIU <= 0
• La dirección de la métrica.
• El valor obtenido debe estar relacionado con el código evaluado.</td></tr>
<tr><td>Caso de prueba</td><td>Criterios de aceptación/rechazo</td></tr>
<tr><td>MMic-CP-01</td><td rowspan="20">En cada caso de prueba se especificarán las entradas que las métricas requieren y las salidas o resultados que se calculen, obtenidos de manera manual.

Un caso de prueba debe considerarse válido cuando los resultados manuales estén contenidos entre 0 y 1.

Para que se pase la prueba, cada métrica debe pasar todos sus casos de prueba.</td></tr>
<tr><td>MMic-CP-02</td></tr>
<tr><td>MMic-CP-03</td></tr>
<tr><td>MMic-CP-04</td></tr>
<tr><td>MMic-CP-05</td></tr>
<tr><td>MMic-CP-06</td></tr>
<tr><td>MMic-CP-07</td></tr>
<tr><td>MMic-CP-08</td></tr>
<tr><td>MMic-CP-09</td></tr>
<tr><td>MMic-CP-10</td></tr>
<tr><td>MMic-CP-11</td></tr>
<tr><td>MMic-CP-12</td></tr>
<tr><td>MMic-CP-13</td></tr>
<tr><td>MMic-CP-14</td></tr>
<tr><td>MMic-CP-15</td></tr>
<tr><td>MMic-CP-16</td></tr>
<tr><td>MMic-CP-17</td></tr>
<tr><td>MMic-CP-18</td></tr>
<tr><td>MMic-CP-19</td></tr>
<tr><td>MMic-CP-20</td></tr>
</table>

4.3. Ejecución de Pruebas

Una vez definido el plan de pruebas y concluida la definición de las métricas, se llevó a cabo la ejecución de las pruebas con proyectos de microservicios obtenidos de repositorios en internet. Los equipos utilizados en las pruebas se describen en la Tabla 14.

Tabla 14. Ambiente de ejecución de pruebas de los equipos utilizados

<table>
<tr><th colspan="2">Ambiente de ejecución de pruebas</th></tr>
<tr><th>Hardware</th><th>Software</th></tr>
<tr><td>Laptop</td><td rowspan="4">Sistema operativo Windows 10.
Docker Desktop 2.5
Visual Studio Code
MySQL (v8.0)
Node.js (v8)
Chrome (v96.0.46)
Java (jre1.8.0_261)
Java (jdk1.8.0_261)
Spring Tool Suite 4
MySQL (Community v8.0)
Postman (v7.34)</td></tr>
<tr><td>Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz.
8.00 GB (7.89 GB utilizable)
Sistema operativo de 64 bits, procesador x64</td></tr>
<tr><td>Escritorio</td></tr>
<tr><td>Intel(R) Core(TM) i7-8300U CPU @ 2.40GHz 2.50 GHz.
8.00 GB
Sistema operativo de 64 bits, procesador x64.
Virtualización Intel® (VT-x)</td></tr>
</table>

A continuación se describe cada uno de los repositorios de microservicios utilizados en las pruebas, cada uno con su entorno de ejecución y lenguajes de desarrollo. En el Anexo 1, se presentan pantallas de la ejecución de los repositorios.

- **Agenda**

Este repositorio muestra el funcionamiento básico de almacenar datos básicos de contacto, mismo que únicamente describe un microservicio con la capacidad de procesar la información y almacenarla en una base de datos (Ilustración 22) [46].

Las herramientas utilizadas en la ejecución de este repositorio son los siguientes:

Herramientas	**Lenguaje**
Java (jre1.8.0_261)	Java
Java (jdk1.8.0_261)	
Spring Tool Suite 4	
MySQL (Community v8.0)	
Postman (v7.34)	

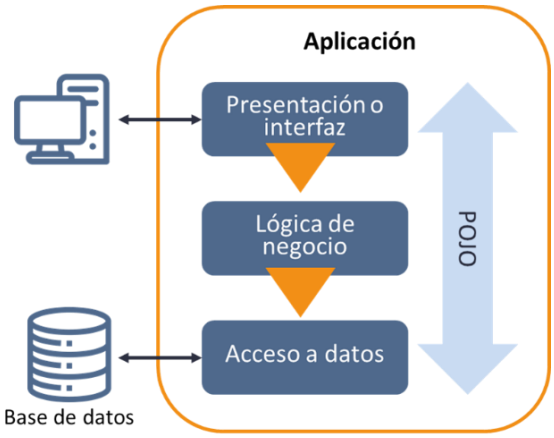


Ilustración 22. Esquema de la aplicación agenda [46]

- **Boletos de Cine**

Este repositorio muestra el funcionamiento básico para vender boletos [45] para las funciones de un cine, mismo que únicamente describe dos microservicios:

- Películas, se encarga de realizar las consultas de todo el catálogo de películas, así como de generar sus horarios, salas y precios, de igual forma se encarga de generar órdenes de boletos con su respectiva información, como lo son el costo, asientos, sala y película.
- Notificaciones, se encarga de realizar las notificaciones a través de correo electrónico enviando a los clientes la información de sus boletos de cine, todo ello se realiza de forma automática al procesarse una compra en el microservicio de Películas.

El esquema de este sistema se puede ver en la Ilustración 23.

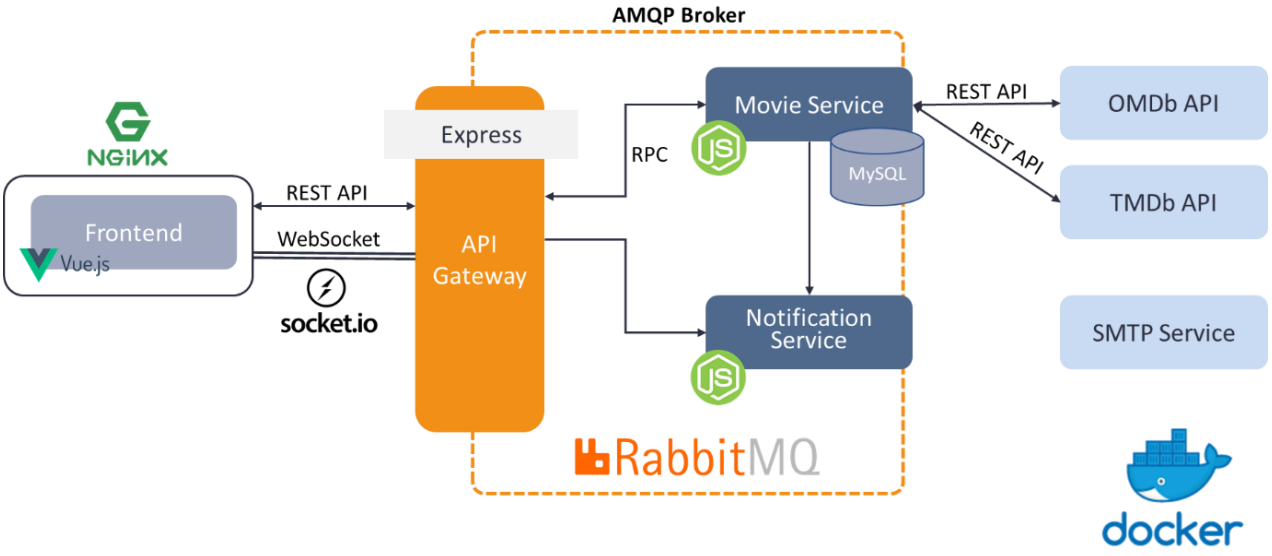


Ilustración 23. Esquema de aplicación boletos de cine [44]

Las herramientas utilizadas en la ejecución de este repositorio son los siguientes:

Herramientas	**Lenguajes**
Docker Desktop (v3.0)	Java
Visual Studio Code (v1.51)	JavaScript
MySQL (v8.0)	Node
Node.js (v12)	

- **Boletos de Tren** [44]

Este repositorio muestra el funcionamiento básico para vender boletos de tren, mismo que describe una serie de 41 microservicios, cada uno de ellos **con una responsabilidad única** que atiende las necesidades del sistema. A continuación de describen algunos de ellos:

- Travel Plan, se encarga de realizar las consultas del catálogo de trenes, en donde filtra diferentes tipos de viajes dependiendo de las necesidades del cliente, por ejemplo, el viaje más corto, el viaje más barato, el viaje sin escalas, de esta forma puede devolver los planes de viajes que cumplan con las necesidades de los clientes.
- Routes, se encarga de realizar las consultas de todo el catálogo de rutas, en este microservicio se registran, eliminan o modifican las rutas de los diferentes trenes, considerando únicamente la estación de origen y la de destino. Así mismo permite consultar el total de rutas disponibles para los trenes.
- Food, se encarga de realizar la administración de órdenes de comida realizadas en los viajes, a través de este microservicio se puede crear, eliminar o buscar órdenes específicas de comida o en su defecto obtener el catálogo completos de las órdenes generadas.

El esquema del proyecto se puede observar en la Ilustración 24.

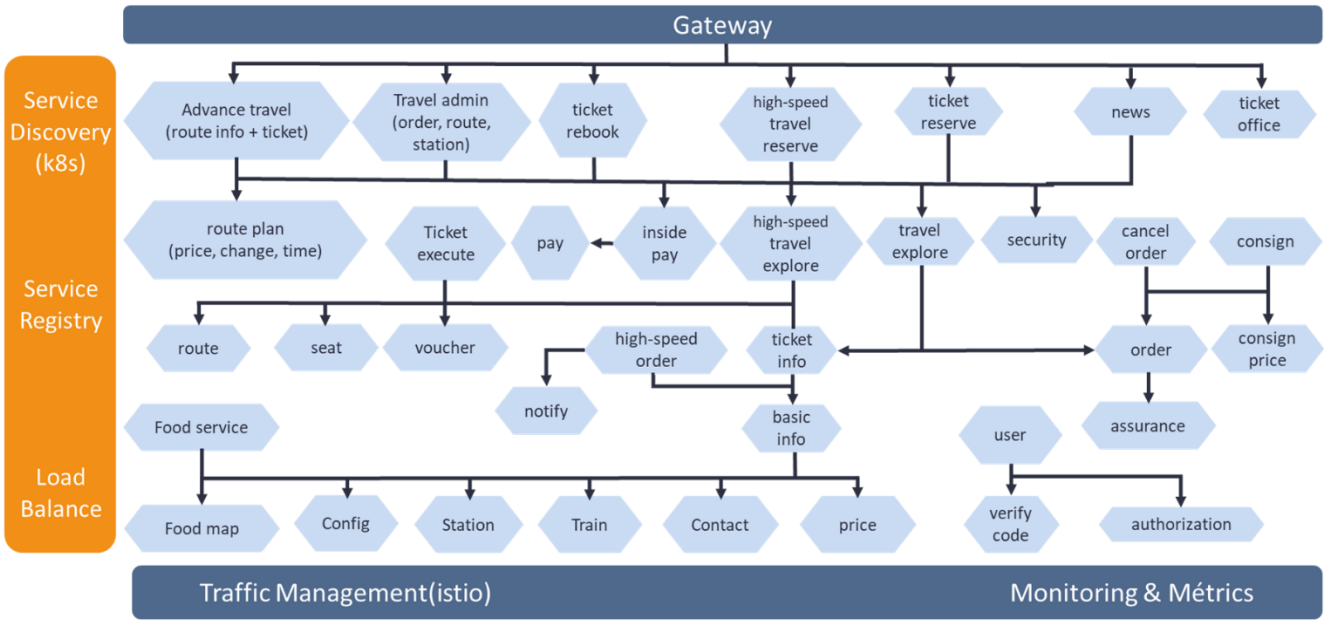


Ilustración 24. Esquema de aplicación de boletos de tren [45]

Las herramientas utilizadas en la ejecución de este repositorio son los siguientes:

Herramientas
Docker Desktop (v3.0)
Visual Studio Code (v1.51)
Node.js (v12)

Lenguajes
Java
Python
Go
Node

Contenedores
Mongo DB
MySQL

4.4. Bitácora de pruebas

En la Tabla 16 se muestra la plantilla para la ejecución del plan de pruebas, que sirvió para registrar los resultados obtenidos en cada prueba.

Tabla 15. 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.		
Repositorio	Nombre del repositorio en el que será aplicado.		
Microservicio	Microservicio al que se le aplica la prueba.		
Objetivo	Objetivo buscado con la correcta ejecución del caso de prueba.		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	Valores de las variables de la métrica 1.	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 dos etapas: La etapa 1 consistió en 6 pruebas aplicados en los tres repositorios presentados. La etapa 2 consistió en 14 pruebas realizadas en el repositorio “Boletos de Tren”. Con ello se logró un total de 20 microservicios evaluados.

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

Ejecución de la prueba MMic-CP-01

Nombre de prueba	MMic-CP-01		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	23/Agosto/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Agenda: Sistema de un microservicio que se encarga de almacenar los datos básicos de un contacto.		
Microservicio	Contacto		
Objetivo	• Evaluar los resultados de la métrica *NIU* (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica *NO* (Número de Operaciones).		
Métricas de la prueba			
No. de Métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{3}{4} = 0.75$	
Observaciones	No se presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{2}{2} = 1$	
Observaciones	El sistema no cuenta con una interfaz gráfica, las solicitudes se realizan a través de POSTMAN, por lo que al aplicar la métrica se consideraron una solicitud al menos para cada interfaz.		

4.5. Resultados de las Pruebas

En este apartado se muestran los resultados obtenidos al ejecutar cada uno de los casos de prueba definidos en este documento. Los casos de prueba fueron elaborados con el propósito de comprobar que las métricas cumplieron su objetivo de medición.

Como se ha mencionado anteriormente, cada caso de prueba incluye una o más instancias de microservicio provenientes de diferentes repositorios. Los casos de prueba se diseñaron para observar las salidas obtenidas utilizando diferentes entradas y ver si se reflejaban posibles errores.

En la Tabla 16, se ven reflejados todos los valores obtenidos en las pruebas.

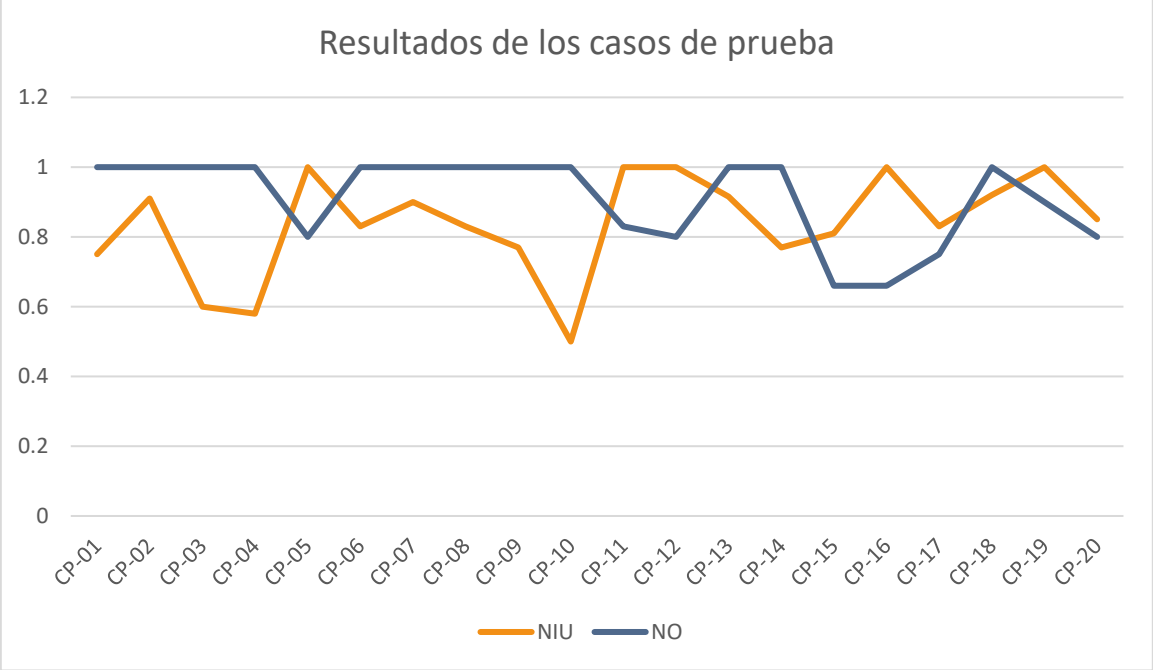
Tabla 16. Resultados de las métricas aplicadas en las pruebas

Caso de prueba	Microservicio	Métrica/Valor	
		NO	NIU
MMIC-CP-01	Agenda	0.75	1
MMIC-CP-02	Movies	0.91	1
MMIC-CP-03	Notifications	0.6	1
MMIC-CP-04	Travel Plan	0.58	1
MMIC-CP-05	Routes	1	0.8
MMIC-CP-06	Food	0.83	1
MMIC-CP-07	Order	0.9	1
MMIC-CP-08	Price	0.83	1
MMIC-CP-09	Station	0.77	1
MMIC-CP-10	News	0.5	1
MMIC-CP-11	User	1	0.83
MMIC-CP-12	Notification	1	0.8
MMIC-CP-13	Food Map	0.916	1
MMIC-CP-14	Contacts	0.77	1
MMIC-CP-15	Seat	0.81	0.66
MMIC-CP-16	Rebook	1	0.66
MMIC-CP-17	Route Plan	0.83	0.75
MMIC-CP-18	Preserve	0.92	1
MMIC-CP-19	Travel2	1	0.90
MMIC-CP-20	Train	0.85	0.8

Las pruebas se realizaron en dos etapas, la etapa 1 consistió en evaluar los primeros 6 casos de prueba, si bien los resultados fueron favorables para los microservicios, se presentó una anomalía

en el caso MMIC-CP-03, que, a diferencia de los otros, la métrica NIU presentó un valor óptimo (1 en la escala de 0 a 1) mientras que en NO, su valor no fue el más óptimo (0.6).

La etapa 2, consistió en evaluar 14 casos de prueba más, los resultados fueron más variables, con lo que la anomalía de la etapa 1 se puede considerar normal, no alterando los resultados esperados de las métricas.



Gráfica 1 Resultados de las métricas para todos los casos de prueba.

En la Gráfica 1, se observan los resultados obtenidos en todos los casos de prueba, con ellos tenemos la comparativa entre los resultados obtenidos en la evaluación de las métricas. Es importante mencionar que, a pesar de ser diferentes repositorios desarrollados por diferentes autores, los valores se encuentran arriba de 0.5, que indica el punto medio del rango establecido en las métricas (0 a 1) en la evaluación de los microservicios, esto indica que los microservicios evaluados se encuentran en un estado aceptable para funcionar pero con algunas dificultades para mantenerlos, por ello, con base en las métricas se puede tener una idea de los errores más comunes al diseñar y desarrollar microservicios.

4.6. Análisis de resultados

Una vez concluidas las pruebas, se analizan los resultados como indicadores de mejora.

El resultado obtenido de la investigación fue la adaptación de la métrica RFC (Response For a Class) de Chidamber y Kemerer. Las métricas obtenidas son: *NIU (Número de Interfaces Utilizadas)* y *NO (Número de Operaciones Utilizadas).*

NO describe el comportamiento de los microservicios en relación a las operaciones utilizadas para resolver peticiones. El mejor valor (cuando el resultado de *NO* tiende a 1) indica que utiliza la totalidad de los métodos/funciones en el microservicio para resolver peticiones, por lo que la

granularidad del microservicio es la adecuada para su responsabilidad. *NIU* describe el comportamiento de los microservicios con relación al número de interfaces utilizadas para resolver peticiones. El valor más alto (cuando el resultado de *NIU* tiende a 1) indica que la comunicación y utilización del microservicio está respondiendo exactamente a su capacidad empresarial.

Los resultados obtenidos de los casos de prueba muestran una idea más precisa del estado de los microservicios en un sistema, dando solución al problema planteado. Con estos resultados se comprobó que:

- Los valores calculados en los diferentes casos de prueba se encuentran en ambos extremos de los umbrales:
  - Con resultados más cercano a cero se plantea la hipótesis de que existe una menor comprensión del microservicio cuando en la estructura del microservicio, el lenguaje de desarrollo y la comunicación externa se ven afectados con exceso de componentes/métodos/funciones/endpoints, que podrían ser autogenerados por herramientas de desarrollo o lenguaje de programación.
  - Mientras que en casos con resultados más cercanos a 1, se plantea la hipótesis de que existe:
    a) Una mejor comprensión de la estructura del microservicios, teniendo los elementos necesarios sin código basura.
    b) La comunicación con otros microservicios se vuelve más sencilla al conocer exactamente con que elementos esta interactuando.
    c) El alcance del microservicio se puede identificar con base a las funciones definidas, así como el límite de su capacidad empresarial.
- En los casos de prueba MMic-CP-02 y MMic-CP-18, los resultados de las métricas dan indicadores cercanos al 1, lo que se puede interpretar como microservicios con cambios mínimos, examinando el código se encontró con posible código basura o con exceso de métodos. Sin embargo, la estructura interna de los microservicios contiene métodos que son ejecutados al iniciarse en contenedores virtuales, por lo que se puede decir que el lenguaje utilizado en el desarrollo de microservicio puede generar código "auxiliar" que complica la lectura y funcionamiento del mismo.

Caso de prueba	Microservicio	NO	NIU
MMic-CP-02	Movies	0.91	1
MMic-CP-18	Preserve	0.92	1

- Muchas de las interfaces definidas en los microservicios son utilizados en solicitudes específicas del sistema, por ejemplo, en la realización de pruebas. En el caso de prueba MMic-CP-05, se definen diferentes interfaces para acceder a las rutas de tren, sin embargo, una de ellas no se utiliza en ningún momento dentro del ciclo de vida del sistema, lo que a interpretación propia, su definición indica que fue diseñada como consulta especifica.

Y de igual forma en el caso de prueba MMic-CP-16 se observa que al menos una interfaz de acceso al microservicio no es utilizada en ningún momento.

Caso de prueba	Microservicio	NO	NIU
MMic-CP-05	Routes	1	0.8
MMic-CP-16	Rebook	1	0.66

- Durante la realización de los diferentes casos de prueba, se analizaron los diferentes indicadores de las métricas, con lo que se observó una relación entre el valor obtenido con el código fuente de los microservicios evaluados y su tecnología, lenguaje y organización; Si bien es cierto que muchos de ellos son diferentes, se puede decir que, en los proyectos en donde los resultados fueron más elevados, no tiene una relación con su tamaño. Sin embargo, se observó que los indicadores más bajos dan lugar a que estructuralmente sean más complejos y difíciles de entender, una de las hipótesis es que su alto contenido de código inutilizable/inalcanzable/autogenerado es confundible con el que sí es importante, mismo que podría interpretarse como lógica de negocio excesiva.
- Los casos de prueba del MMic-CP-04 al MMicCP-20 se aplicaron sobre un repositorio compuesto por 41 microservicios, la evaluación dio resultados favorables, sin embargo, una de las características de esta arquitectura es la libertad tecnológica, por lo que podemos tener microservicios desarrollados con diferentes lenguajes, plataforma o herramientas, por ello, para tener una idea del nivel de complejidad del repositorio sería necesario aplicar la evaluación en varios microservicios de la misma tecnología, por ejemplo: si el repositorio contiene microservicios desarrollados en 3 lenguajes diferentes, convendría evaluar un pequeño porcentaje de microservicios desarrollado en cada lenguaje, esto con el fin de tener una idea general de estado del repositorio.
- La aplicación de las métricas se realizó de forma manual, sin embargo, en sistemas muy grandes con cientos de microservicios funcionando entre sí, esto significaría algo insostenible, por ello, es necesario tener en cuenta el desarrollo de una herramienta que permita la automatización de la evaluación de las métricas.

CAPÍTULO

05

Conclusiones

En este capítulo se presentan las conclusiones del desarrollo de la investigación y la sugerencia de algunos trabajos futuros.

5.1. Conclusiones

La arquitectura de microservicios es un tema de investigación creciente en los últimos años. Muchos puntos de vista se han presentado en múltiples publicaciones en diferentes plataformas, muchas de ellas han ayudado a comprender el funcionamiento, adopción y transición hacia esta arquitectura.

En esta investigación se desarrolló un estudio de la adopción de microservicios, en donde se abordó el funcionamiento de sistemas orientados a microservicios, las características que los componen y algunas buenas prácticas.

Los hallazgos encontrados en el análisis de métricas dieron un panorama amplio de que se está haciendo para comprender mejor esta arquitectura, de lo que podemos decir:

- La arquitectura de microservicios sigue siendo objeto de investigación para comprender y establecer metodologías de adaptación.
- Se han desarrollado Frameworks que proyectan facilitar la transición y adopción.
- Las métricas han sido una herramienta fundamental en la evaluación de la calidad de arquitecturas de microservicios.
- Se están utilizando métricas de las arquitecturas SOA para aplicarlas en MSA, algunas de forma exitosa.
- Aún existe un largo camino por recorrer en la comprensión de las arquitecturas de microservicios.

La aportación de este trabajo de investigación son dos métricas que nos indican la comunicación y utilización de los recursos (métodos/funciones) definidos con respecto a microservicios. El objetivo de realizar la adaptación de una métrica existente es utilizar el conocimiento generado por la comunidad científica en la aplicación de métricas en el desarrollo de software, con el que se redefine su estructura para ser adecuada en contextos diferentes (Microservicios), dando como resultado nuevas métricas.

Las métricas que se obtuvieron: NIU y NO, son dos métricas de diseño orientadas a microservicios, que dan una idea de la autonomía y organización dentro de un sistema, por lo que en una aplicación en la fase de diseño se pueden utilizar como predictores tempranos de las características de calidad del software, lo que permitiría identificar problemas potenciales en las primeras fases del ciclo de vida del desarrollo del software. Su aplicación en sistemas existentes se puede utilizar como indicadores del estado de los microservicios, con ello se tiene un panorama de cómo están definidos dando al desarrollador una idea de lo que se puede mejorar.

El mejor caso es obtener el valor de 1 en las métricas, indicador de un microservicio con los elementos (*endpoints*, métodos/funciones) exactos para solventar su capacidad empresarial, sin necesidad de realizar cambios.

- Para NIU, los resultados obtenidos (inferiores a 0.5) denotan cambios en la comunicación externa, en donde es de gran importancia establecer los límites y el alcance de los microservicios.

- Para NO, los resultados obtenidos (inferiores a 0.6) denotan cambios internos, en donde es importante definir las funciones que atenderán las necesidades del microservicios.

En el análisis de las métricas propuestas, se observó una relación en las características de MSA. La definición de autonomía cita “Los microservicios son una entidad separada. Cada servicio/componente en una arquitectura de microservicios se puede desarrollar, implementar, operar y escalar sin afectar el funcionamiento de otros servicios” [2, 3], en donde, conjuntamente las métricas (NO, NIU) podrían ser indicadores para detectar falta de autonomía e independencia, teniendo recursos (*endpoints*/métodos/funciones) sobrantes para el funcionamiento de microservicios.

En este contexto de adaptación de métricas se explora un enfoque de reutilizar el conocimiento existente para llevarlo a otros contexto (arquitecturas), con base en el análisis, pruebas y resultados de los autores y la comunidad.

La investigación cumplió con los alcances propuestos al inicio:

- Explorar las métricas para el desarrollo de software que puedan ser extendidas para su aplicación en el desarrollo de microservicios.
- Considerar métricas orientadas a objetos enfocadas al diseño, instalación, tolerancia a fallos y servicios.

Sin embargo, existen factores que no fueron considerados en su desarrollo, por ejemplo: el código autogenerado por las herramientas de desarrollo, métodos reutilizados, clases auxiliares y llamadas externas para resolver solicitudes, mismas que podrían incluirse en trabajos futuros después de aplicar pruebas más extensas.

Al final se puede concluir que, a pesar de tener cada día más investigaciones, modelos, marcos, métricas y múltiples herramientas, la experiencia puede ser un factor definitorio a la hora de diseñar microservicios, pero, siempre es importante ver como los demás están abordando los problemas, comprobar, evaluar y mejorar el diseño y desarrollo de microservicios.

5.2. Trabajos futuros

La investigación da pauta para el desarrollo de trabajos posteriores que den continuidad al trabajo realizado, por lo que se propone:

- Realizar la adaptación de las métricas que no fueron seleccionadas.
- Desarrollar una herramienta que permita la aplicación de estas métricas de forma automatizada.
- Afinar las métricas presentadas en esta investigación para considerar aspectos importantes no presentes en las métricas originales.

Referencias

[1] K. S. Davide Taibi, «A Decomposition and Metric-Based Evaluation Framework for Microservices.,» *Communications in Computer and Information Science (Bajo Revisión),* 2020.

[2] A. W. S. W. A. Z. Justus Bogner, «Exploring Maintainability Assurance Research for Service- and Microservice-Based Systems: Directions and Differences,» de *Joint Post-proceedings of the First and Second International Conference on Microservices (Microservices 2017/2019)*, vol. 78, Dagstuhl, Alemania, Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik, 2020, pp. 3:1--3:22.

[3] S. G. A. L. L. M. M. F. M. R. M. L. S. Nicola Dragoni, «Microservices: yesterday, today, and tomorrow,» de *Present and Ulterior Software Engineering*, Springer, Cham, 2017, pp. 195-216.

[4] C. A. M. M. P. P. H. M. Alan Bandeira, «We Need to Talk about Microservices: an Analysis from the Discussions on StackOverflow,» de *IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, Montreal, QC, Canada, Canada, 2019.

[5] V. L. C. P. A. J. Davide Taibi, «Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages,» de *XP '17 Workshops: Proceedings of the Scientific Workshops of XP2017*, Cologne Germany, 2017.

[6] J. B. A. Z. S. W. Jonas Fritzsch, «From Monolith to Microservices: A Classification of Refactoring Approaches,» de *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. DEVOPS 2018*, Chateau de Villebrumier, France, 2018.

[7] P. L. I. M. Paolo Di Francesco, «Architecting with microservices: A systematic mapping study,» *Journal of Systems and Software,* vol. 150, pp. 77-97, 2019.

[8] R. B. R. K. Sara Hassan, «Microservice Transition and its Granularity Problem: A Systematic Mapping Study,» *Software: Practice and Experience,* vol. 50, nº 9, pp. 1651-1681, 2020.

[9] D. Ulander, «Software Architectural Metrics for the Scania Internet of Things Platform - From a Microservice Perspective,» Teknisk- naturvetenskaplig fakultet, 2017.

[10] C. K. S.R. Chidamber, «A Metrics Suite for Object Oriented Design.,» *IEEE Transactions on Software Engineering,* vol. 20, nº 6, pp. 476-493, 1994.

[11] K. S. Davide Taibi, «From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining.,» *9th International Conference on Cloud Computing and Services Science,* vol. 1, pp. 153-164, 2019.

[12] S. W. A. Z. Justus Bogner, «Automatically Measuring the Maintainability of Service - and Microservice-based Systems – a Literature Review,» de *IWSM/Mensura '17: 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, Gothenburg Sweden, 2017.

[13] Y. C. R. K. L. X. Q. F. Ran Mo, «Decoupling Level: A New Metric for Architectural Maintenance Complexity,» de *IEEE/ACM 38th IEEE International Conference on Software Engineering*, Austin, TX, USA, 2016.

[14] D. Ulander, Software Architectural Metrics From a Microservice Perspective, Suecia: Examensarbete, 2017.

[15] S. W. A. Z. Justus Bogner, «Towards a Practical Maintainability Quality Model for Serviceand Microservice-based Systems,» de *ECSA '17: 11th European Conference on Software Architecture Canterbury United Kingdom*, Canterbury, United Kingdom, 2017.

[16] Y. E. S. Tugrul Asik, «Policy Enforcement upon Software Based on Microservice Architecture,» de *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, London, UK, 2017.

[17] M. F. James Lewis, «Microservices,» 25 Marzo 2014. [En línea]. Available: https://martinfowler.com/articles/microservices.html.

[18] E. M. Daniel López, «Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web,» de *Séptima Conferencia de Directores de Tecnología de Información, TICAL 2017*, San José, 2017.

[19] https://aws.amazon.com/es/microservices/, «aws,» Amazon, 2020. [En línea]. Available: https://aws.amazon.com/es/microservices/.

[20] C. Toby, «Testing Strategies in a Microservices Architecture,» [En línea]. Available: http://martinfowler.com/articles/microservice-testing/. [Último acceso: 14 08 2020].

[21] Microsoft, «Microsoft,» 30 01 2020. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture.

[22] M. Wasson, «Microsoft,» 30 10 2019. [En línea]. Available: https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices.

[23] J. I. P. H. A. P. Ángela Indira Rodríguez, «Arquitectura basada en micro-servicios para aplicaciones web,» *Tecnologia, Investigación y Academia,* vol. 7, nº 2, pp. 12 - xxxxxx, 2019.

[24] D. A. B. Contreras, «Market Cart App: aplicación móvil para la gestión de compra,» *Tecnología, Investigación y Academia (TIA),* vol. 6, nº 1, pp. 36-46, 2018.

[25] B. Atkisson, «RedHad,» 4 Mayo 2017. [En línea]. Available: https://www.redhat.com/es/topics/microservices.

[26] J. K. M. Lorenz, Object-Oriented Software Metrics, Prentice Hall, 1996.

[27] IEEE, «IEEE Standard Glossary of Software Engineering Terminology,» *ISO/IEC/IEEE 24765,* pp. 1-84, 2010.

[28] J. H. B. L, «Control de Calidad en el Software,» de *ICESI*.

[29] S. L. P. Norman E. Fenton, Software Metrics: A Rigorous and Practical Approach, Pws Pub Co, 1998.

[30] C. P. M. J. R. Luis Férnandez, «Proceso de definición de métricas y criterios de calidad en el ciclo de vida de la educación virtual accesible,» de *IV Congresso Internacional sobre Qualidade e Acessibilidade da Formação Virtual*, Lisboa, Portugal, 2013.

[31] S. Alexandre, Software Metrics An Overview Version 1.0, Belgium: University of Namur, 2002.

[32] D. Thakur, «Classification of Software Metrics in Software Engineering,» Computer Notes, 2016. [En línea].

[33] C. Richardson, «Microservice Architecture,» [En línea]. Available: https://microservices.io/patterns/microservices.html. [Último acceso: 13 Diciembre 2021].

[34] N. Ford, «Building Microservice Architectures,» [En línea]. Available: http://nealford.com/downloads/Building_Microservice_Architectures_Neal_Ford.pdf. [Último acceso: 12 04 2021].

[35] N. Medvidovic, «A Classification and Comparison Framework for Software Architecture Description Languages,» *IEEE Transactions on Software Engineering,* vol. 26, nº 1, pp. 70 - 93, 2000.

[36] C. R. K. F. a. Z. T. Mikhail Perepletchikov, «Coupling Metrics for Predicting Maintainability in Service-Oriented Designs,» de *2007 Australian Software Engineering Conference (ASWEC'07)*, Australian, 2007.

[37] D. G. EBRAHIM BAGHERI, «Assessing the Maintainability of Software Product Line Feature Models using Structural Metrics,» *Software Quality Journal,* p. 579–612, 2011.

[38] IEEE, «INTERNATIONAL STANDARD ISO/IEC/IEEE 24765,» 2010.

[39] L.-O. D. J. E. T. G. K. H. P. J. S. K. D. M. F. M. K. R. P. T. Patrik Berander, Software quality attributes and trade-offs, Blekinge Institute of Technology, 2005.

[40] C. F. K. Shyam R. Chidamber, «A Metrics Suite for Object Oriented Design,» *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,* vol. 20, nº 6, pp. 476-493, 1994.

[41] M. G. R. E. Fernando Brito e Abreu, «Toward the Design Quality Evaluationof Object-Oriented Software Systems,» de *5th InternationalConference on Software Quality*, Austin, Texas, 1995.

[42] T. J. McCabe, «A Complexity Measure,» *IEEE Transactions on Software,* vol. 5, 1976.

[43] D. L. Rosenberg, «Applying and Interpreting Object Oriented Metrics,» de *Software Technology Conference*, Utah, 1998.

[44] M. Zarach, «github,» 29 05 2019. [En línea]. Available: https://github.com/michalzaq12/movie-ticket-booking-system. [Último acceso: 21 05 2021].

[45] FudanSELab, «github,» 16 09 2017. [En línea]. Available: https://github.com/FudanSELab/train-ticket. [Último acceso: 18 05 2021].

[46] sinbugs, «http://sinbugs.com/,» 13 08 13. [En línea]. Available: http://sinbugs.com/como-crear-un-microservicio-o-servicio-web-rest-con-spring-boot-1/. [Último acceso: 04 05 2021].

[47] S. K. K. I. Y. M. T. Kamiya, «Empirical evaluation of reuse sensitiveness of complexity metrics,» *Information and Software Technology 41,* p. 297–305, 1999.

[48] Christophe, «Microservices dependence,» 2020.

Anexo A - Escalas de medición de una métrica

Nominal

Esta escala tiene dos características principales:

- El sistema de relación empírica consta únicamente de diferentes clases; no hay noción de ordenamiento entre las clases.
- Cualquier numeración distinta o representación simbólica de las clases es una medida aceptable. Pero no hay ninguna noción de magnitud asociada con el número o símbolo.

Ejemplo:
Tratamos de clasificar el conjunto de fallas de software en el código. Elegimos una escala de medida donde las fallas son entidades y su ubicación son atributos. Entonces, la ubicación de fallas podría estar en tres conjuntos diferentes: especificación, diseño o código. Entonces podemos definir un mapeo M que asigne las diferentes clases a un número en particular. 45 si x es una falla de especificación, 2 si x es una falla de diseño y 37 si x es una falla de código. El valor no es importante aquí.

Escala

Esta escala tiene tres características:

- Sistema empírico de relaciones formado por clases ordenadas con respecto al atributo.
- Cualquier mapeo que conserve el orden es aceptable.
- Los números representan solo rangos, por lo que la suma y otras operaciones matemáticas no tienen sentido.

Ejemplo:
Se desea clasificar los diferentes módulos de su software en tres clases que denotan la complejidad (trivial, simple, complejo). Entonces eliges una aplicación M como en la escala nominal: 1 si x es trivial, 2 si x es simple y 3 si x es complejo. La diferencia con la escala anterior radica en que el mapeo de medidas debe conservar el orden de complejidad. 3 es mayor que 1 conserva la relación más compleja.

Intervalo

Las tres características principales son:

- Se conserva el orden.
- Se conservan las diferencias pero no las proporciones.
- La suma y la resta son aceptables, pero no la multiplicación y la división.

Ejemplo:

Tome la medida de la temperatura en grados Celsius o Fahrenheit, donde cada grado es una clase relacionada con el calor. Decimos que la temperatura en un lugar X es de 20 grados centígrados y, al mismo tiempo, de 30 grados centígrados en el lugar Y. Si la temperatura se mueve, en X, de 20 a 21, el calor aumentará exactamente de la misma manera si cambie de 30 a 31 en el lugar Y. Entonces la relación se conserva.

Proporción

Hay cuatro características:

- Se conservan el orden, el tamaño de los intervalos entre entidades y las proporciones.
- Hay un elemento cero (representa la falta total de atributo).
- El mapeo de medición comienza en cero y aumenta a intervalos iguales (unidades).
- Toda la aritmética se puede aplicar a las clases en el rango del mapeo.

Ejemplo:
Utiliza una escala de razón cuando mide el tamaño físico de las entidades. La escala comienza en cero, lo que representa la falta total de tamaño (teórico, sin existencia). Puedes medir el tamaño en centímetros, metros, etc.

Absoluta

La escala absoluta respeta las cuatro propiedades siguientes:

- La medida se realiza simplemente contando el número de elementos en el conjunto de entidades.
- El Atributo siempre toma la forma "número de ocurrencias de x en la entidad".
- Solo hay un mapeo de medición, a saber, el recuento real.
- Todo análisis aritmético del conteo resultante es significativo.

Ejemplo:
Lines Of Codes (LOC) es una medida de escala absoluta del atributo "número de líneas de códigos" de un programa. Pero "número de centímetros" no es una medida de escala absoluta del tamaño de una persona porque también puedes usar pulgadas, metros,...

Anexo B - Repositorios de microservicios

Repositorio de Agenda

- Estructura del repositorio.

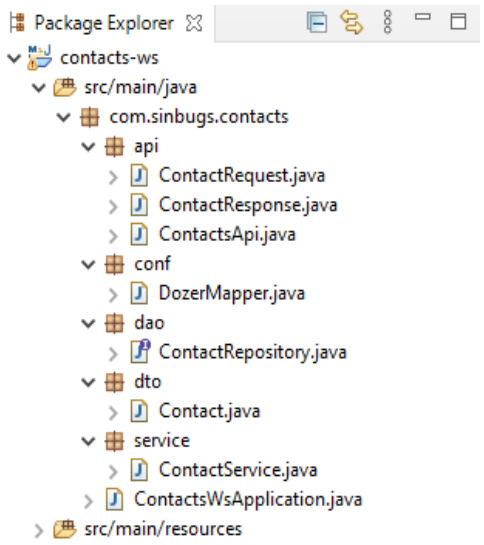


Ilustración 25. Estructura del repositorio Agenda

- Contenido del microservicio

Contact.java | ContactService.java | ContactRequest.java | ContactsApi.java | ContactResponse.java

```java
package com.sinbugs.contacts.api;

import javax.validation.Valid;

import org.dozer.Mapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.sinbugs.contacts.dto.Contact;
import com.sinbugs.contacts.service.ContactService;

@RestController
public class ContactsApi {

    @Autowired
    ContactService contactService;

    @Autowired
    Mapper mapper;

    @RequestMapping(value="/contact", method=RequestMethod.POST)
    public ContactResponse updateOrSave(@RequestBody @Valid ContactRequest ContactRequest){
        Contact contact = mapper.map(ContactRequest, Contact.class);
        // Invoca lógica de negocio
        Contact updatedContact = contactService.save(contact);
        // Mapeo entity ==> response dto
        ContactResponse contactResponse = mapper.map(updatedContact, ContactResponse.class);
        return contactResponse;
    }

    @RequestMapping(value="/contact", method=RequestMethod.GET)
    public Contact getById(){
        return new Contact(1L, "John", "Doe", "+57 311 222 3344", "john@sinbugs.com");
    }
}

```

Ilustración 26. API de acceso al microservicio

- Pruebas del microservicio

  1. Se realiza una petición al microservicio Contact.

  2. La estructura que lleva el cuerpo de la solicitud es JSON.

  3. El contenido de la solicitud lleva información de un contacto.

  4. La respuesta a la solicitud es la información del contacto con su identificador en la base de datos.

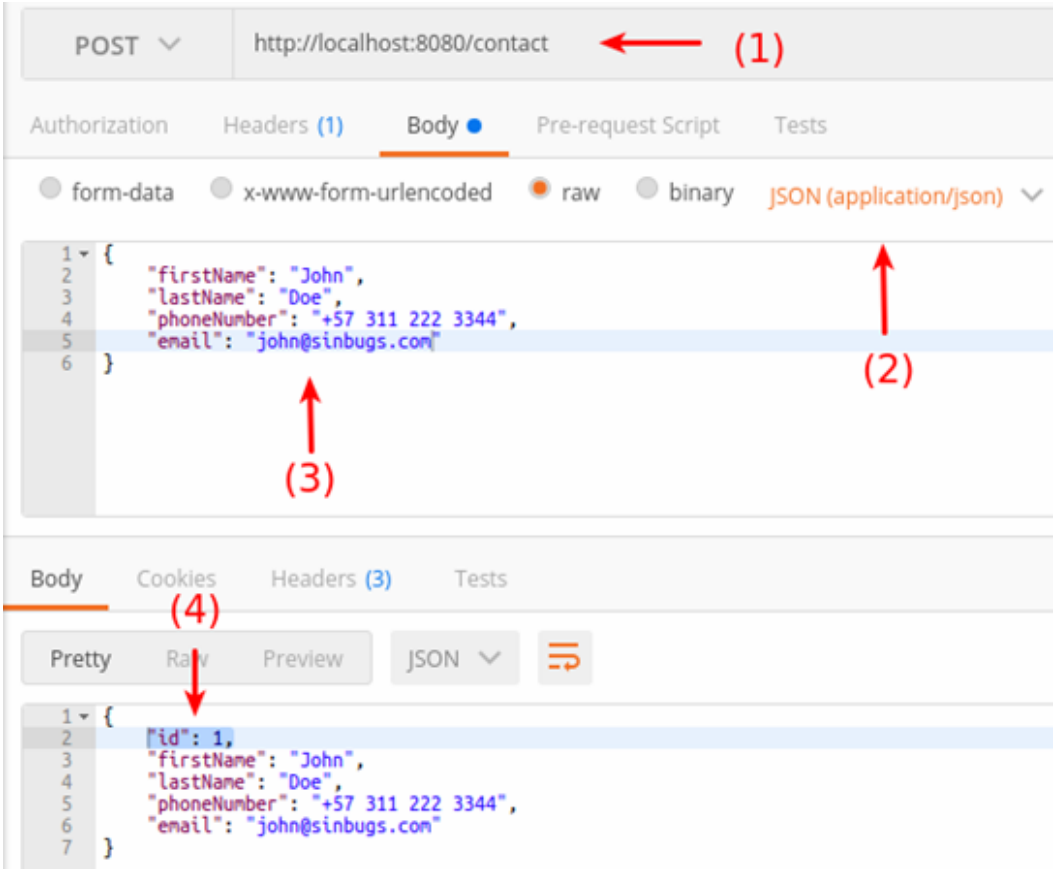


Ilustración 27. Pruebas del microservicio

Repositorio de Boletos de Cine

- Estructura del repositorio

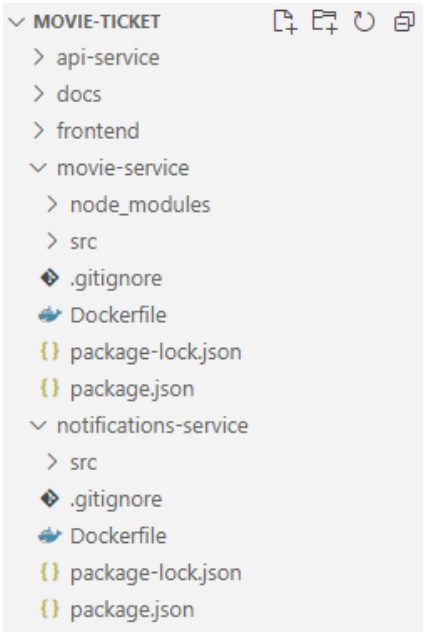


Ilustración 28. Estructura del repositorio boletos de cine

- Contenido de los microservicios

movie.js

movie-service > src > controllers > movie.js > ...

```
1   const Op = require('sequelize').Op;
2   const {Movie, Seats} = require('../models');
3   const openMovieService = require('../services/openMovie');
4   const movieTrailerService = require('movie-trailer');
5
6   const alphabet = 'abcdefghijklmnopqrstuvwxyz';
7
8   const SEATS_ROWS = 6;
9   const SEATS_COMUNS = 6;
10
11
12  module.exports = {
13      async create(data){
14          const movie = await Movie.create(data);
15
16          const seats = [];
17
18          for(let x = 0; x < SEATS_ROWS; x++){
19              const row = alphabet[x].toUpperCase();
20              for(let y = 0; y < SEATS_COMUNS; y++){
21                  const seat = Seats.create({
22                      row,
23                      column: (y + 1),
24                      movieId: movie.id
25                  });
26                  seats.push(seat);
27              }
```

Ilustración 29. Contenido del microservicio de películas

movie.js

api-service > src > services > movie.js > ...

```
let globalChannel = null;
let replyQueue = null;
const promises = new Map(); // correlationId => Function - callback

/**
 *
 * @param {string} q
 * @returns {Promise<*|Error>}
 */
async function createChannel(q) {···
}

function sendMessage(data){···
}

createChannel(config.services.movies_q).then(channel => {···
});

module.exports = {
    /**
     *
     * @param {object} data
     * @returns {Promise<void | Error>}
     */
    createMovie(data){
        return sendMessage({action: 'movie.create', body: data})
    },

    getAllMovies(){
        return sendMessage({action: 'movie.getAll'})
    },
```

Ilustración 30. API de acceso al microservicio

- Prueba del proyecto

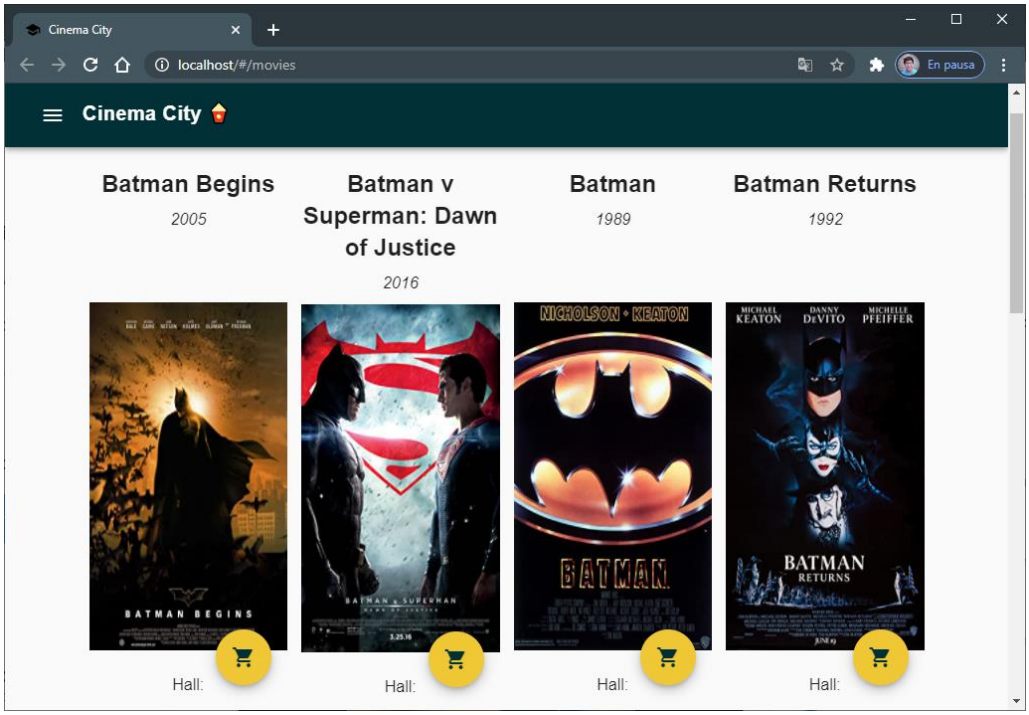


Ilustración 31. Prueba del microservicio

Repositorio de Boletos de Tren

- Estructura del repositorio

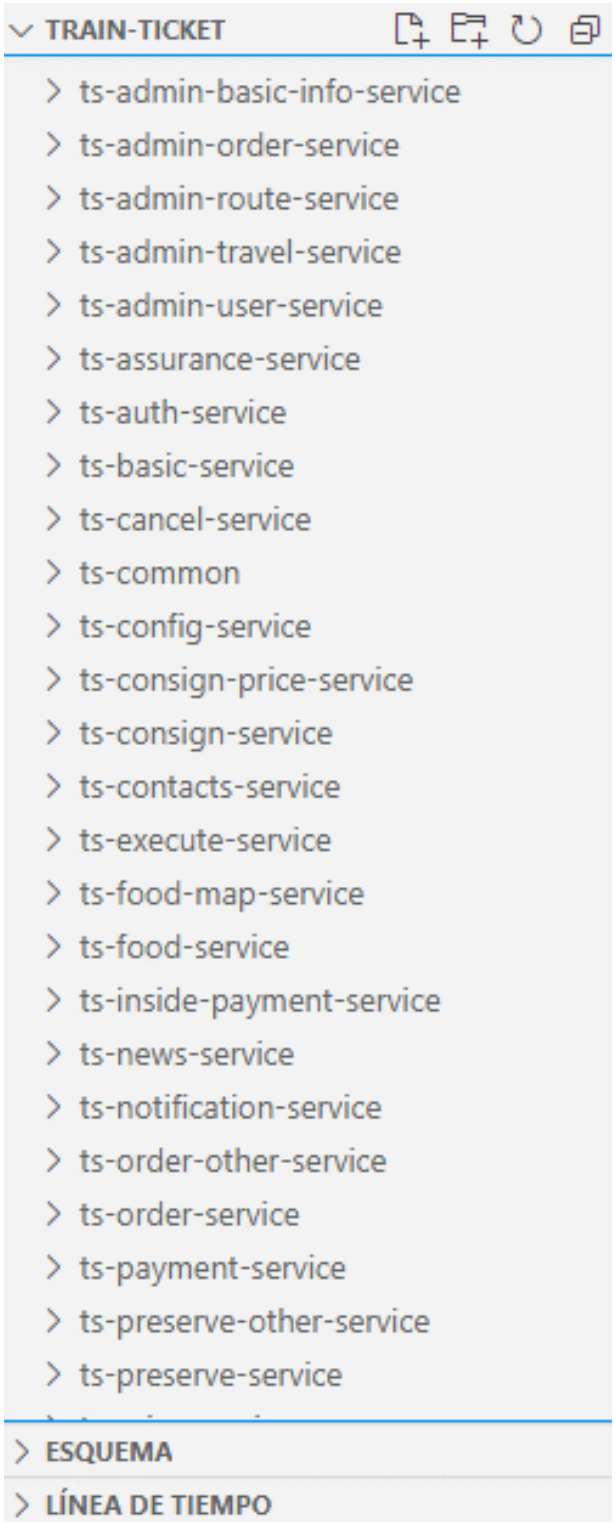


Ilustración 32. Estructura del repositorio boletos de tren

- Contenido del microservicio

FoodServiceImpl.java

ts-food-service > src > main > java > foodsearch > service > FoodServiceImpl.java

```java
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.stream.Collectors;

@Service
public class FoodServiceImpl implements FoodService {

    @Autowired
    private RestTemplate restTemplate;

    @Autowired
    private FoodOrderRepository foodOrderRepository;

    private static final Logger LOGGER = LoggerFactory.getLogger(FoodServiceImpl.class);

    String success = "Success.";
    String orderIdNotExist = "Order Id Is Non-Existent.";

    @Override
    public Response createFoodOrder(FoodOrder addFoodOrder, HttpHeaders headers) {…
    }

    @Override
    public Response deleteFoodOrder(String orderId, HttpHeaders headers) {…
    }
```

Ilustración 33. Contenido del microservicio food

FoodController.java

ts-food-service > src > main > java > foodsearch > controller > FoodController.java

```java
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import static org.springframework.http.ResponseEntity.ok;

@RestController
@RequestMapping("/api/v1/foodservice")
public class FoodController {

    @Autowired
    FoodService foodService;

    private static final Logger LOGGER = LoggerFactory.getLogger(FoodController.class);

    @GetMapping(path = "/welcome")
    public String home() {
        return "Welcome to [ Food Service ] !";
    }

    @GetMapping(path = "/orders")
    public HttpEntity findAllFoodOrder(@RequestHeader HttpHeaders headers) {
        FoodController.LOGGER.info("[Food Service]Try to Find all FoodOrder!");
        return ok(foodService.findAllFoodOrder(headers));
    }
```

Ilustración 34. API de acceso al microservicio

- Pruebas

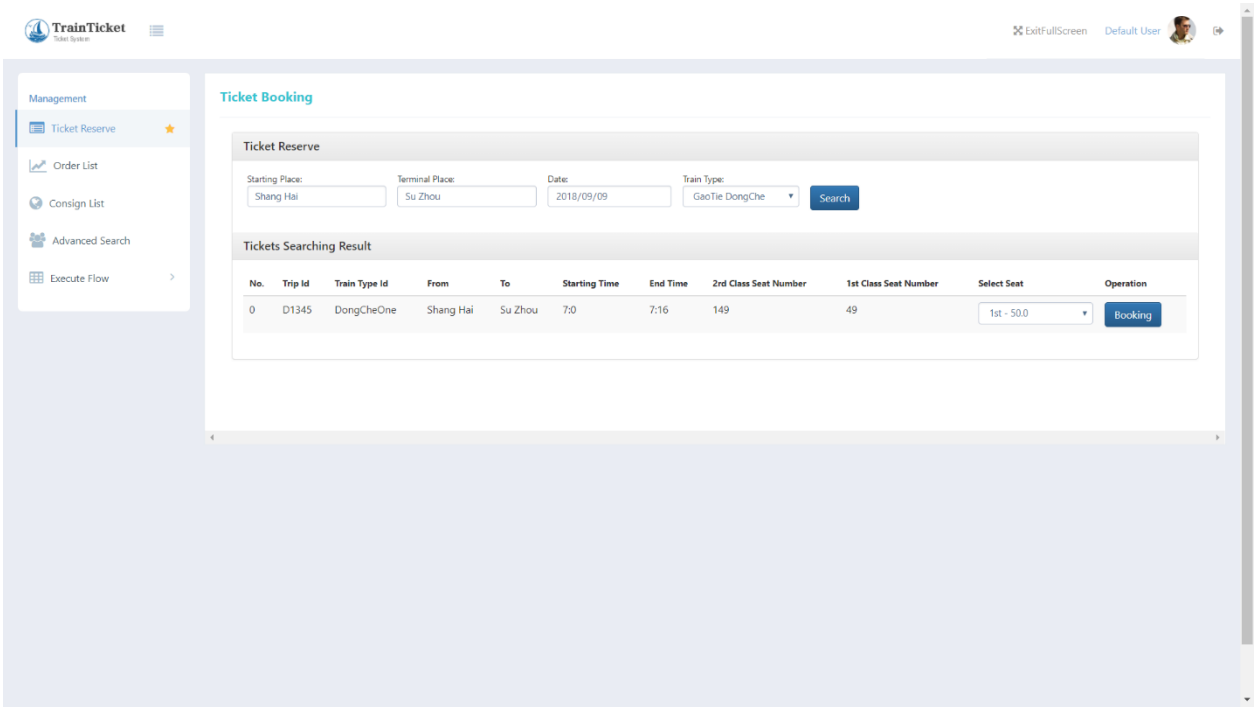


Ilustración 35. Prueba del microservicio

Anexo C - Bitácora de pruebas

Nombre de prueba	MMic-CP-02		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	23/Agosto/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de cine: Sistema de dos microservicios que se encarga de procesar la venta de boletos para funciones de cine.		
Microservicio	Películas		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{11}{12} = 0.91$	
Observaciones	El microservicio tiene un método de inicialización del proyecto, este no fue considerado ya que es generado por el lenguaje de desarrollo y el contenedor virtual.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{5}{6} = 0.83$	
Observaciones	No sé presentó problema.		

Nombre de prueba	MMic-CP-03		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	25/Agosto/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de cine: Sistema de dos microservicios que se encarga de procesar la venta de boletos para funciones de cine.		
Microservicio	Notificaciones		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{3}{5} = 0.6$	
Observaciones	El microservicio tiene un método de inicialización del proyecto, este no fue considerado ya que es generado por el lenguaje de desarrollo y el contenedor virtual.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{1}{1} = 1$	
Observaciones	No sé presentó problema.		

Nombre de prueba	**MMic-CP-04**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	26/Agosto/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Travel Plan Service		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{7}{12} = 0.58$	
Observaciones	No se presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{4}{4} = 1$	
Observaciones	No se presentó problema.		

Nombre de prueba	MMic-CP-05		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	30/Agosto/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Route		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{5}{5} = 1$	
Observaciones	No se presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{4}{5} = 0.8$	
Observaciones	No se presentó problema.		

Nombre de prueba	MMic-CP-06		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	31/Agosto/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Food		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{5}{6} = 0.83$	
Observaciones	No se presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{6}{6} = 1$	
Observaciones	No se presentó problema.		

Nombre de prueba	MMic-CP-07		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	20/Septiembre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Order		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{19}{20} = 0.95$	
Observaciones	El número de métodos utilizados son totales en comparación con otros microservicios, sin embargo, existe uno de autenticación que no se entiende por completo el momento de su utilización.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{15}{15} = 1$	
Observaciones	Las interfaces de esta microservicio están divididas por administrador y usuario normal.		

Nombre de prueba	**MMic-CP-08**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	21/Septiembre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Price		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{5}{6} = 0.83$	
Observaciones	No se presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{5}{5} = 1$	
Observaciones	El microservicio únicamente realiza un CRUD sobre los precios de un producto.		

Nombre de prueba	MMic-CP-09		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	23/Septiembre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Station		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{7}{9} = 0.77$	
Observaciones	Los métodos definidos están divididos en dos secciones, por lo se pudiera hablar de una separación en dos microservicios.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{9}{9} = 1$	
Observaciones	Las interfaces definidas se encuentran en dos partes, la primera son solicitudes de operaciones sobre las estaciones, la segunda es sobre querys de consulta complejos.		

Nombre de prueba	**MMic-CP-10**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	11/Octubre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	News		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{1}{2} = 0.5$	
Observaciones	Las funciones sugieren un servicio que se ejecuta en consecuencia al iniciar el sistema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{1}{1} = 1$	
Observaciones	El lenguaje deja en duda la definición de las interfaces para este microservicio, sugiere un servicio conectado directamente aun repositorio externos en la web.		

Nombre de prueba	MMic-CP-11		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	12/Octubre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	User		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{12}{12} = 1$	
Observaciones	Los métodos definidos con exactamente los utilizados en la autenticación y operaciones sobre los usuarios.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{5}{6} = 0.83$	
Observaciones	Contiene una interfaz de prueba que no fue removida.		

Nombre de prueba	**MMic-CP-12**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	13/Octubre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Notification		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{4}{4} = 1$	
Observaciones	Los métodos están definidos de acuerdo al número de interfaces utilizadas, sin considerar la de prueba.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{4}{5} = 0.8$	
Observaciones	Existe una interfaz de bienvenida/prueba para el servicio de notificaciones, que sirve para verificar que está funcionando.		

Nombre de prueba	MMic-CP-13		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	18/Octubre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Food Map		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{11}{12} = 0.916$	
Observaciones	No se presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{2}{2} = 1$	
Observaciones	No se presentó problema.		

Nombre de prueba	MMic-CP-14		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	18/Octubre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Contacts		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{7}{9} = 0.77$	
Observaciones	Los método están bien definidos de acuerdo a sus interfaces, pero, tiene excedente de funciones auxiliares.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{7}{7} = 1$	
Observaciones	No se presentó problema.		

Nombre de prueba	**MMic-CP-15**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	19/Octubre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Seat		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{9}{11} = 0.81$	
Observaciones	No se presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) *e I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{2}{3} = 0.66$	
Observaciones	No se presentó problema.		

Nombre de prueba	**MMic-CP-16**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	03/Noviembre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Reebok		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{30}{30} = 1$	
Observaciones	No se presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) e *I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{2}{3} = 0.66$	
Observaciones	No se presentó problema.		

Nombre de prueba	**MMic-CP-17**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	04/Noviembre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Route Plan		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{16}{18} = 0.83$	
Observaciones	La utilización de otros servicios auxiliares no están considerados en las métricas.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) e *I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{3}{4} = 0.75$	
Observaciones	La interfaces definidas son las necesarios, a excepción de una de prueba.		

Nombre de prueba	**MMic-CP-18**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	04/Noviembre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Preserve		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{35}{38} = 0.92$	
Observaciones	No presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) e *I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{1}{1} = 1$	
Observaciones	No presentó problema.		

Nombre de prueba	**MMic-CP-19**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	08/Noviembre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Travel2		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{33}{33} = 1$	
Observaciones	No presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) e *I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{10}{11} = 0.90$	
Observaciones	No presentó problema.		

Nombre de prueba	**MMic-CP-20**		
Autor	Iván Daniel Joya de la Cruz	**Fecha**	11/Noviembre/2021
Tipo de prueba	Prueba de aceptación		
DP a evaluar	MMic-DP-01		
Repositorio	Boletos de tren: Sistema de múltiples microservicios que se encarga de procesar la venta de boletos para viajes de tren.		
Microservicio	Train		
Objetivo	• Evaluar los resultados de la métrica NIU (Número de Interfaces utilizadas). • Evaluar los resultados de la métrica NO (Número de Operaciones).		
Métricas de la prueba			
No. de métrica	**Entrada**	**Salida**	
Métrica 1	El valor de *MU* (Número de operaciones utilizadas) y *O* (Número de operaciones totales) de *NO*.	El valor de la métrica: $NO = \frac{MU}{O}$ $NO = \frac{6}{7} = 0.85$	
Observaciones	No presentó problema.		
Métrica 2	El valor de *NIA* (Número de interfaces accedidas) e *I* (Número de interfaces totales) para la métrica *NIU*.	El valor de la métrica: $NIU = \frac{NIA}{I}$ $NIU = \frac{4}{5} = 0.8$	
Observaciones	No presentó problema.		