Gobierno de arquitecturas híbridas REST y GraphQL basadas en acuerdos de nivel de servicio

Antonio Quiña Mera

Tesis Doctoral

Directores: Dr. Pablo Fernández Montes y Dr. José María García

Universidad de Sevilla

julio 2022

First published in julio 2022 by
Antonio Quiña Mera

https://www.researchgate.net/profile/Antonio-Quina-Mera
aquina@utn.edu.ec

Esta tesis doctoral ha sido parcialmente financiada por la Universidad Técnica del Norte, por FEDER /Ministerio de Ciencia e Innovación - Agencia Estatal de Investigación en el marco del proyecto HORATIO (RTI2018-101204-B-C21), y por la Junta de Andalucía en los proyectos APOLO (US-1264651) y EKIPMENT-PLUS (P18-FR-2895)

Dr. Pablo Fernández Montes, Profesor Titular del Área de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla y Dr. José María García, Profesor Titular del Área de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla,

HACEN CONSTAR

que D. Antonio Quiña Mera, Profesor Titular de la Universidad Técnica del Norte - Ecuador, ha realizado bajo nuestra supervisión el trabajo titulado

Gobierno de arquitecturas híbridas REST y GraphQL basadas en acuerdos de nivel de servicio

Una vez revisado, autorizamos el comienzo de los trámites para su presentación como Tesis Doctoral al tribunal que ha de juzgarlo.

Fdo. Dr. Pablo Fernández Montes y Dr. José María García
en la Universidad de Sevilla
19/07/2022

Agradecimientos

En primer lugar, agradezco a Dios por ser la base y guía de mi vida, por darme la oportunidad de compartir con personas estupendas que de manera generosa desde sus espacios me han brindado su ayuda, experiencia y conocimientos para el culmino de este reto académico.

A mis directores de tesis. A Pablo, por compartir su vasta experiencia en esta disciplina, y por impregnar su sello de calidad en el trabajo realizado. A José Mari, por su practicidad para abordar problemas y aportar soluciones brillantes. Gracias a los dos por haber hecho posible la realización de esta tesis.

A Antonio Ruiz por ser un tutor y líder excepcional en este proyecto, por confiar en mí y abrirme las puertas del programa de doctorado y del grupo de investigación ISA.

A mi amada esposa Cathy por ser mi compañera de viaje, gracias por tu temple e incondicionalidad para superar las pruebas que se nos ha presentado y con la bendición de Dios superar las que vengan a futuro.

A mi hija Antonella que con sus cuatro años ha cambiado mi visión de la vida, y por ser una fuente inagotable de cariño y ocurrencias que refresca, calma, y alborota mis días, pero siempre alimenta mis ganas de seguir a delante.

A mi madre Corita por estar siempre allí alentándome para alcanzar mis sueños, a toda mi familia por su aliento, paciencia, y comprensión por el tiempo no compartido.

Destacar un agradecimiento especial a la Universidad Técnica del Norte y sus autoridades por brindar las condiciones necesarias para la profesionalización de sus profesores y así contribuir al crecimiento de la calidad en la educación superior de nuestro país.

Resumen

En esta memoria, se presenta un trabajo de investigación con el objetivo de fomentar la gobernanza de arquitecturas híbridas REST y GraphQL basadas en acuerdos de nivel de servicio. En donde, se han desarrollado doce capítulos estructurados en seis partes. *Parte I: Introducción*, contiene el planteamiento del problema y la metodología de investigación. *Parte II: Estado de la cuestión*, contiene básicamente el estado de GraphQL y los acuerdos de nivel de servicios. *Parte III: Propuesta*, contiene por un lado, dos diseños experimentales para comprobar los efectos de las arquitecturas REST, GraphQL, e híbrida REST/GraphQL en la calidad de software. Por otro lado, la descripción de un estándar denominado *SLA4GraphQL* para modelar y operativizar acuerdos de nivel de servicio (SLA) para APIs GraphQL. *Parte IV: Validación*, contiene la validación de las propuestas expuestas en la parte III. *Parte V: Conclusiones* del estudio. Y finalmente *Parte VI: Anexos*.

La importancia de elaborar un estándar para modelar y operativizar SLA ha sido motivado por tres aspectos. Primero, debido a que no existe en la comunidad científica una propuesta para modelar y operativizar acuerdos de nivel de servicio de una manera estandarizada. Segundo la importancia de proponer un ciclo de vida para el desarrollo de APIs basado en SLA. Tercero facilitar una herramienta que valide la especificación de documentos SLA basados en un estándar. La validación del estándar *SLA4GraphQL* consistió en evaluar la viabilidad, flexibilidad y la idoneidad funcional mediante la instanciación del estándar en varios casos de APIs públicas. El resultado de la validación indica que el estándar *SLA4GraphQL* es viable, flexible, y tiene una idoneidad del 92.33 % para modelar acuerdos de nivel de servicios para APIs GraphQL.

Abstract

This document presents research work to promote the governance of hybrid REST and GraphQL architectures based on service-level agreements. Twelve chapters structured in six parts have been developed. *Part I: Introduction*, contains the problem statement and the research methodology. *Part II: State of the art*, basically includes the state of GraphQL and service level agreements. *Part III: Proposal*, contains, on the one hand, two experimental designs to test the effects of REST, GraphQL, and hybrid REST/GraphQL architectures on software quality. On the other hand, the description of a standard called *SLA4GraphQL* for modeling and operationalizing service level agreements (SLA) for GraphQL APIs. *Part IV: Validation*, contains the validation of the proposals presented in Part III. *Part V: Conclusions* of the study. And finally, *Part VI: Annexes*.

The importance of developing a standard for modeling and operationalizing SLA has been motivated by three aspects. First, there is no proposal in the scientific community to model and operationalize SLAs in a standardized way. Second, the importance of proposing a lifecycle for API development based on SLAs. Third, to provide a tool to validate the specification of SLA documents based on a standard. The validation of the *SLA4GraphQL* standard consisted of assessing the feasibility, flexibility, and functional suitability by instantiating the standard in several instances of public APIs. The result of the validation indicates that the *SLA4GraphQL* standard is feasible, flexible, and has suitability of 92.33 % for modeling service level agreements for GraphQL APIs.

Índice general

ÍNDICE DE FIGURAS

Índice de tablas

Parte I

Introducción

Planteamiento del problema

Las grandes oportunidades nacen de haber sabido aprovechar las pequeñas

Bill Gates (1955–),
Magnate empresarial estadounidense

Este primer capítulo presenta el problema que se aborda en el estudio, justificando su importancia, y las razones que han motivado su investigación. El trabajo comienza con la sección §1.1, que expone la motivación y planteamiento del problema. En la sección §1.2, se identifica la tendencia de estudio de los paradigmas usados en las arquitecturas de software orientada a servicios y microservicios. En la sección §1.3, se presenta el objetivo principal de este trabajo, como también los objetivos parciales que contribuyen a alcanzar dicho objetivo principal. En la sección §1.4, se describe el diseño de la investigación, y los aportes alcanzados en el desarrollo de los objetivos planteados. En la sección §1.5, se lista los proyectos de investigación en los que se enmarcó el presente trabajo. Finalmente, la sección §1.6 describe la estructura del resto de la memoria.

1.1 Motivación

Hoy en día, el *software como servicio* (SaaS, por sus siglas en inglés de *Software as a Service*) ha recibido una atención significativa como uno de los tres principales modelos de servicios que brinda la computación en nube; los otros dos modelos es la plataforma como servicio (PaaS, por sus siglas en inglés de Platform as a Service), y la infraestructura como servicio (IaaS, por sus siglas en inglés de Infrastructure as a Service) [108, 167]. El SaaS utiliza Internet para ofrecer a sus usuarios aplicaciones en la nube, estas aplicaciones junto con su infraestructura y plataformas subyacentes son gestionadas por un proveedor de servicios [108]; en donde, los arquitectos de software deben construir SaaS fiables y eficientes que superen los importantes problemas técnicos y funcionales de las aplicaciones en la nube, como el multi-tenancy, redundancia, recuperación o escalabilidad [167]. Por lo tanto, para los usuarios la selección del mejor proveedor de SaaS entre los disponibles es una actividad crítica [133]. Para superar estos retos, la arquitectura de microservicios representa una clara tendencia tanto en el ámbito académico como en el industrial [103], donde grandes empresas de todo el mundo han evolucionado sus aplicaciones hacia este estilo arquitectónico [28].

En concreto, la *Arquitectura de Microservicios* (MSA, por sus siglas en inglés de *Microservice Architecture*) se compone de pequeñas aplicaciones (microservicios) con una única responsabilidad (requisitos funcionales, no funcionales o multifuncionales) que pueden ser desplegados, escalados y probados de forma independiente [7, 163]. Los microservicios pueden desarrollarse y modificarse de forma independiente utilizando diferentes lenguajes de programación o frameworks de desarrollo, apoyando así la agilidad del proceso de desarrollo de software [7, 69]. Sus principales ventajas son la mantenibilidad, reutilización, escalabilidad, disponibilidad y despliegue automatizado [103]. Los microservicios se comunican a través de mecanismos ligeros, normalmente utilizando el estilo arquitectónico *REST* (por sus siglas en inglés de *REpresentational State Transfer*) para construir *interfaces de programación de aplicaciones* (API, por sus siglas en inglés de *Application Programming Interface*) denominadas *REST API*. Las organizaciones de software han adoptado rápidamente las APIs REST en el desarrollo de sus aplicaciones tras su creación en el año 2000 [178].

REST es el paradigma más utilizado en el desarrollo de microservicios [188]. Sin embargo, a pesar de su popularidad, presenta algunos problemas de versionamiento de sus APIs y acceso a los datos, tales como: (i) la complejidad de las consultas (es decir, REST requiere múltiples peticiones HTTP para obtener múltiples recursos); (ii) el

over-fetching (es decir, una petición REST devuelve más datos de los que una aplicación necesita); y (iii) el *under-fetching* (es decir, un endpoint concreto de una API no devuelve suficiente información, por lo que hay que hacer peticiones adicionales para obtener la información requerida), lo que también se conoce en la literatura como el problema de n+1 peticiones [110, 174]. Para ilustrar los problemas de over-fetching y under-fetching encontrados en REST, se considera un escenario de consumo de una API REST y una API GraphQL de consultas de Blogs. En este escenario, una aplicación cliente necesita consultar los títulos de los Posts de un usuario específico y los nombres de los tres últimos seguidores de ese usuario, ver figura §1.1 [125].

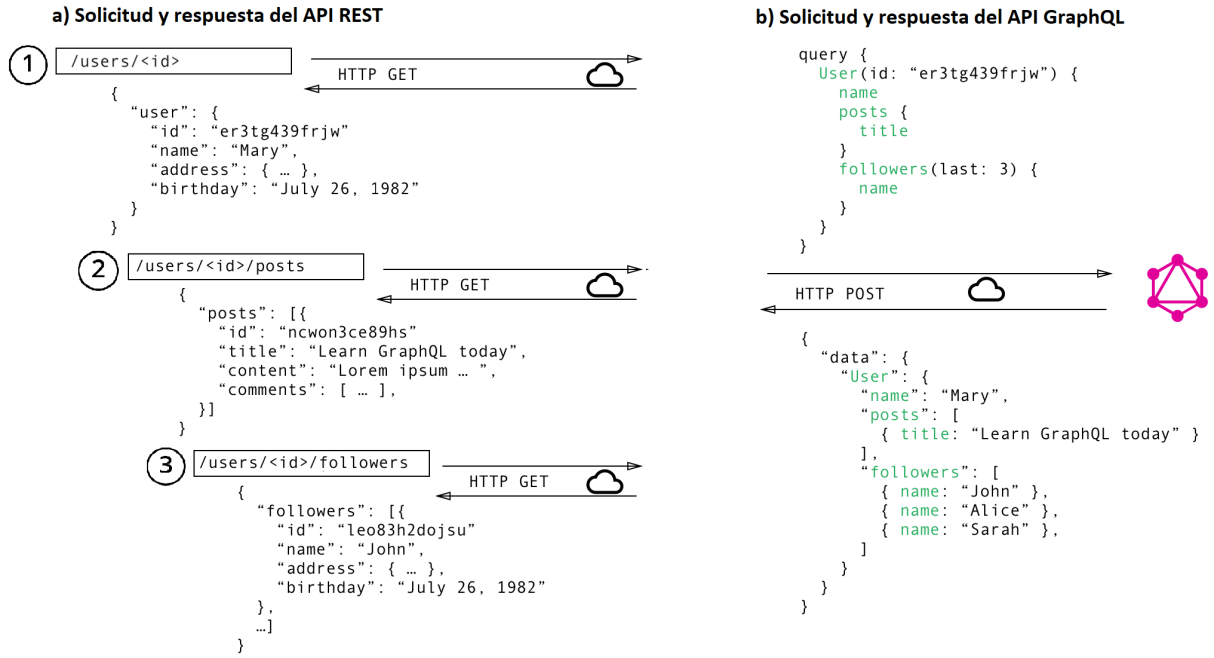


Figura 1.1: Ejemplo de over-fetching y under-fetching de una API REST frente a una API GraphQL

Por un lado, la figura §1.1 a) muestra la solicitud y respuesta a una API REST, que solicita datos accediendo a tres URLs (también conocidos como endpoints). El escenario propuesto se trata de una API REST de Blogs, en donde, el endpoint `/users/<id>` devuelve datos del usuario; el endpoint `/users/<id>/posts` devuelve las publicaciones del usuario; y el endpoint `/users/<id>/followers` devuelve una lista de seguidores por usuario. Por otro lado, la figura §1.1 b) muestra la petición y respuesta a una API GraphQL; utilizando este paradigma, el usuario puede enviar sólo una petición con los datos específicos que necesita, entonces el servidor responde a esta única consulta con los datos solicitados [125, 174]. En consecuencia, se observa que el under-fetching en

REST surge cuando la petición necesita acceder a tres endpoints para obtener los requisitos de datos, mientras que en GraphQL sólo se necesita un acceso para obtener los datos requeridos. A su vez, el over-fetching en REST se produce cuando cada acceso a un endpoint obtiene más datos de los que necesita, mientras que GraphQL sólo se obtiene los datos solicitados.

De acuerdo con lo antes mencionado, se observa que en la última década han surgido varios lenguajes de consulta como SPARQL, Cypher, Gremlin y *GraphQL* como alternativas de REST para el acceso a datos de APIs [146]. En esta memoria, se centrará en el estudio de GraphQL debido al creciente interés de su aplicación en la industria, en donde se ha mostrado como una alternativa para resolver los problemas encontrados en las API REST tradicionales [146, 174]. GraphQL empezó en 2012 como una especificación desarrollada internamente en Facebook. En 2015, Facebook decidió compartir GraphQL con una comunidad más amplia [101], liberando tanto la especificación de código abierto como una implementación de referencia a través de la comunidad graphql.org[1]. Desde entonces, la comunidad ha crecido, así como la adopción en el entorno de producción por parte de cientos de organizaciones como Atlassian, GitHub, Netflix, The New York Times y Twitter, entre otras [101, 115].

Motivados por la acogida de GraphQL en la industria, se realizó una revisión y búsqueda inicial de la literatura en las principales bases de datos científicas de la ingeniería de software, en donde se analizó y se evidenció que:

- No existen estudios detallados que muestren una visión global de este tema en la comunidad científica, y por ende, *se motiva la realización de una investigación sistemática de la literatura y un estudio profundo acerca de GraphQL*.
- Existe la necesidad de entender a profundidad en que condiciones y circunstancias es mejor utilizar una arquitectura REST, o una arquitectura GraphQL, o una arquitectura híbrida REST y GraphQL, por tal motivo, *se motiva el estudio del efecto de este tipo de arquitecturas en la calidad del software*.

  En este sentido, se ha observado que una práctica común de estudio de las arquitecturas híbridas REST y GraphQL (REST/GraphQL) es analizarlas desde un punto del vista en donde, una API GraphQL sea la interfaz principal de comunicación y está consuma como origen de datos a las APIs REST, a esta práctica se le conoce como envoltura (wrapper) [10, 41, 49, 68, 183], con la cual además, se puede reutilizar las APIs existentes. Para ofrecer una interfaz GraphQL, los

[1] Ver http://graphql.org

proveedores tienen que implementar, operar y evolucionar el API GraphQL para que envuelva y resuelva las consultas de GraphQL mediante la formulación de solicitudes contra las APIs REST existentes [183].

- No existen estudios que hayan tratado la gestión del ciclo de vida del desarrollo y operación del servicio de las arquitecturas GraphQL o arquitecturas híbridas REST/GraphQL, por tal razón, *se motiva el estudio de la gobernanza de este tipo arquitecturas*.

  En este sentido, para entender de mejor manera la gobernanza de las arquitecturas GraphQL y arquitecturas híbridas REST/GraphQL. Por un lado, se observó que la definición de gobernanza difiere de acuerdo al dominio que se gobierne, como por ejemplo gobierno de TI, gobierno de la arquitectura, o gobierno de arquitectura orientada a los servicios (SOA, por sus siglas en inglés de Service Oriented Architecture) [58]. Debido a que esta tesis se centra en el estudio del gobierno de arquitecturas orientadas a microservicios para el desarrollo de APIs; se acoge la definición del modelo y gobierno de arquitecturas SOA. El modelo de gobierno SOA es simplemente un marco que permite a una organización (proveedor) llegar a un consenso sobre el alcance, definición y uso del gobierno de SOA; el gobierno de la SOA es una especialización que amplía el gobierno de la TI y se centra en la gestión eficaz del ciclo de vida de un servicio [170].

Por estos motivos, y porque hasta donde sabemos, no existe ninguna propuesta formal para gobernar arquitecturas GraphQL o arquitecturas híbridas REST/GraphQL (vista desde el punto de vista de envoltorios) mediante la gestión de acuerdos de nivel de servicio (SLA); se motiva el *estudio de la gobernanza de arquitecturas híbridas REST y GraphQL mediante acuerdos de nivel de servicio*.

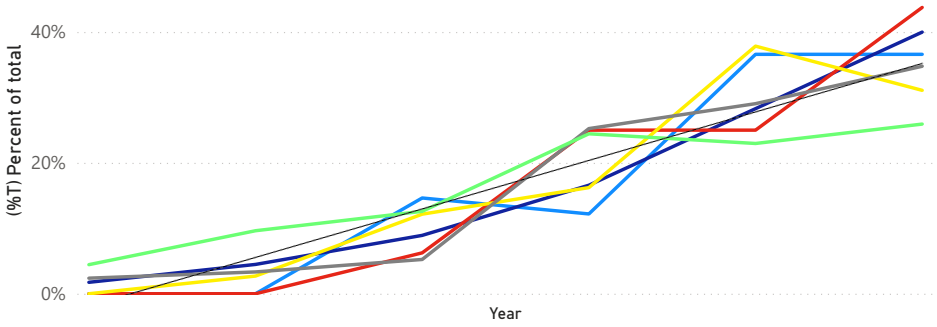
1.2 RELEVANCIA DEL PROBLEMA

La comunidad científica y tecnológica ha mostrado una tendencia en el desarrollo de aplicaciones utilizando arquitecturas microservicios basadas en REST, las cuales son expuestas como servicios en la nube. Por un lado, REST es un estilo arquitectónico ampliamente aceptado, sin embargo, ha presentado varios problemas de manejo de datos [188]. Por otro lado, GraphQL se muestra como una alternativa para atenuar estos inconvenientes presentados por REST [110, 174].

Por tal razón, se revisó el estado de GraphQL, en donde se evidenció que es un

paradigma importante que ha llamado la atención de la industria [101, 115], sin embargo, no está muy claro cómo ha sido su acogida en la comunidad científica. Por lo cual, se realizó una revisión en cinco bases de datos científicas más representativas en la ingeniería del software: ACM Digital Library[2], SpringerLink[3], IEEE Explore Library[4], Scopus[5], Google Scholar[6] y DBLP[7] [32, 88, 96]. Buscando estudios de revisión sistemática de la literatura (SLR, por sus siglas en ingles de Systematic Literature Review) y estudios de mapeo sistemático (SMS, por sus siglas en inglés de Systematic Mapping Study) acerca de GraphQL que muestren una visión general de este tema; pero no se encontró ningún estudio de este tipo, por este motivo, se encontró relevante empezar un estudio profundo de GraphQL.

Para comprobar la relevancia de estudio de GraphQL en la comunidad científica, se hizo una encuesta a las bases de datos científicas antes mencionadas, preguntando la frecuencia de las publicaciones realizadas en el transcurso del tiempo. Además, se complementó esta comprobación mediante un análisis de la tendencia de búsqueda del término "graphql" en el buscador Google[8] mediante la herramienta Google Trends[9].



BDD	2015	%T.	2016	%T.	2017	%T.	2018	%T.	2019	%T.	2020	%T.
Google Scholar	39	2 %	100	4 %	199	9 %	370	17 %	631	28 %	892	40 %
Springer Link	5	2 %	7	3 %	11	5 %	53	25 %	61	29 %	73	35 %
ACM Digital Library	6	4 %	13	10 %	17	13 %	33	24 %	31	23 %	35	26 %
Scopus	0	0 %	2	3 %	9	12 %	12	16 %	28	38 %	23	31 %
DBLP	0	0 %	0	0 %	6	15 %	5	12 %	15	37 %	15	37 %
IEEE Explore	0	0 %	0	0 %	1	6 %	4	25 %	4	25 %	7	44 %
Total (Sum, Avg)	**50**	**1 %**	**122**	**3 %**	**243**	**10 %**	**477**	**20 %**	**770**	**30 %**	**1045**	**35 %**

Figura 1.2: Tendencia de publicaciones por año

[2]Ver https://dl.acm.org
[3]Ver https://link.springer.com
[4]Ver https://ieeexplore.ieee.org
[5]Ver https://www.scopus.com/
[6]Ver https://scholar.google.com
[7]Ver https://dblp.org
[8]Ver https://www.google.com
[9]Ver https://trends.google.com

Por un lado, la figura §1.2 muestra la tendencia de las publicaciones acerca de GraphQL en las bases de datos; esto se logró tabulando e ilustrando la frecuencia de publicaciones, y el porcentaje del total de publicaciones (%T.) por años. En este análisis se tomó en cuenta publicaciones desde 2015 (lanzamiento de GraphQL a la comunidad) hasta 2020, en donde se evidenció un promedio de crecimiento anual del 7,4%, también se evidenció que en los últimos años se obtuvo los mayores crecimientos, e inclusive la mayoría de las publicaciones (67%)..

Por otro lado, la figura §1.3 muestra la tendencia del interés de búsqueda del término "graphql" en la herramienta *Google Trends*. Los resultados obtenidos en cuanto al interés de búsqueda se promediaron, catalogaron y tabularon por año en un rango de 0 a 100, en donde el valor de 100 indica la máxima popularidad del término de búsqueda, en el análisis indica que existe una tasa media de crecimiento anual del 13,61%.

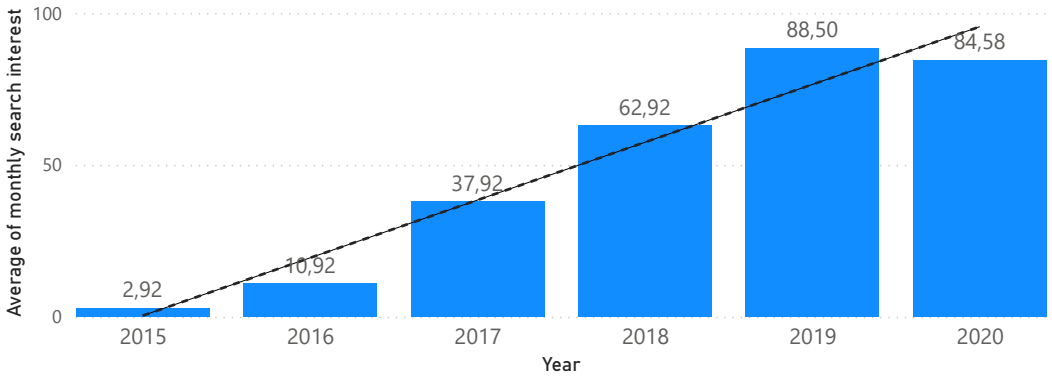


Figura 1.3: Tendencia de búsquedas por año

En base a los valores obtenidos en los dos análisis anteriores, y a la problemática que se aborda en este trabajo se concluye que GraphQL es un tema relevante de estudio.

1.3 Objetivos de la investigación

En este trabajo se estudiará el estilo arquitectónico REST y el lenguaje de consulta GraphQL en el contexto del desarrollo de software con arquitecturas GraphQL y arquitecturas híbridas REST/GraphQL y la descripción de contratos de niveles de servicio (SLA) para APIs GraphQL, por tal razón:

El objetivo principal de esta tesis es **mejorar la gobernanza de arquitecturas híbridas REST y GraphQL mediante acuerdos de nivel de servicio**.

Para alcanzar este objetivo se plantea como subobjetivos dar respuesta a las siguientes preguntas de investigación (PI):

- **PI**$_1$: ¿Cuál es el estado actual de GraphQL? (Capítulo §5).
- **PI**$_2$: ¿Qué condiciones y circunstancias hacen más recomendable seguir el paradigma GraphQL frente al paradigma REST? (Capítulos §6 y §9).
- **PI**$_3$: ¿En qué escenarios resultan adecuadas las arquitecturas híbridas REST/-GraphQL? (Capítulos §7 y §10).
- **PI**$_4$: ¿En qué medida es posible modelar los acuerdos de APIs GraphQL? (Capítulos §8 y §11).

A su vez, la pregunta de investigación **PI**$_1$ se pudo concretar en las siguientes preguntas:

- **PI**$_{1.1}$: ¿Se pueden implementar aplicaciones complejas con suficiente calidad usando el paradigma GraphQL? (Sección §3.5)
- **PI**$_{1.2}$: ¿Quiénes son los autores e instituciones que han investigado acerca de GraphQL? (Sección §5.3.1).
- **PI**$_{1.3}$: ¿Dónde se han publicado los trabajos de investigación acerca de GraphQL? (Sección §5.3.2).
- **PI**$_{1.4}$: ¿Qué tipos y métodos de investigación se han utilizado en las publicaciones acerca de GraphQL? (Sección §5.3.3).
- **PI**$_{1.5}$: ¿Cuándo se han publicado los trabajos de investigación acerca de GraphQL? (Sección §5.3.4).
- **PI**$_{1.6}$: ¿Por qué se ha estudiado el paradigma GraphQL? (Sección §5.3.5).
- **PI**$_{1.7}$: ¿Cómo se ha introducido GraphQL en la comunidad científica? (Sección §5.3.6).

Del mismo modo, la pregunta de investigación **PI**$_2$ se pudo derivar en las siguientes preguntas:

- **PI**$_{2.1}$: ¿Cuál es el efecto del paradigma de desarrollo en la calidad del software?.
- **PI**$_{2.2}$: ¿Cuál es la curva de aprendizaje de los paradigmas REST y GraphQL en el proceso de desarrollo de APIs?.

Por su parte, la pregunta de investigación **PI**$_3$ se deriva en las siguientes preguntas:

- **PI**$_{3.1}$: ¿Cuál es el efecto del desarrollo de una arquitectura híbrida REST/GraphQL en la calidad del software?.
- **PI**$_{3.2}$: ¿En qué escenarios es más adecuado usar las arquitecturas híbridas REST/-GraphQL?.

Así mismo, la pregunta de investigación **PI**$_4$ se concreta en las siguientes preguntas:

- **PI**$_{4.1}$: Viabilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar los acuerdos de nivel de servicio (SLA) de un caso en concreto?.
- **PI**$_{4.2}$: Flexibilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) en distintos casos concretos?.
- **PI**$_{4.3}$: Idoneidad Funcional. ¿En qué medida el estándar *SLA4GraphQL* proporciona funciones que cumplan con necesidades establecidas en condiciones específicas?.

1.4 Resumen de la Contribución

Para llevar a cabo esta investigación, por un lado, se utilizó la experimentación en la ingeniería de software como estrategia empírica [186], para comparar las condiciones adecuadas de uso de los paradigmas REST y GraphQL, como también de las arquitecturas híbridas REST/GraphQL. Por otro lado, se usó el enfoque metodológico *Ciencia del diseño* (DSR, por sus siglas en inglés de *Design Science Research*) que ayuda a crear y evaluar artefactos que resuelven problemas organizacionales identificados [175]. La figura §1.4 muestra el resumen de las aportaciones realizadas en este estudio.

Primeramente, basado en la guía propuesta por Wohlin *et al.* para la experimentación en ingeniería de la software, se definió las siguientes actividades [186]:

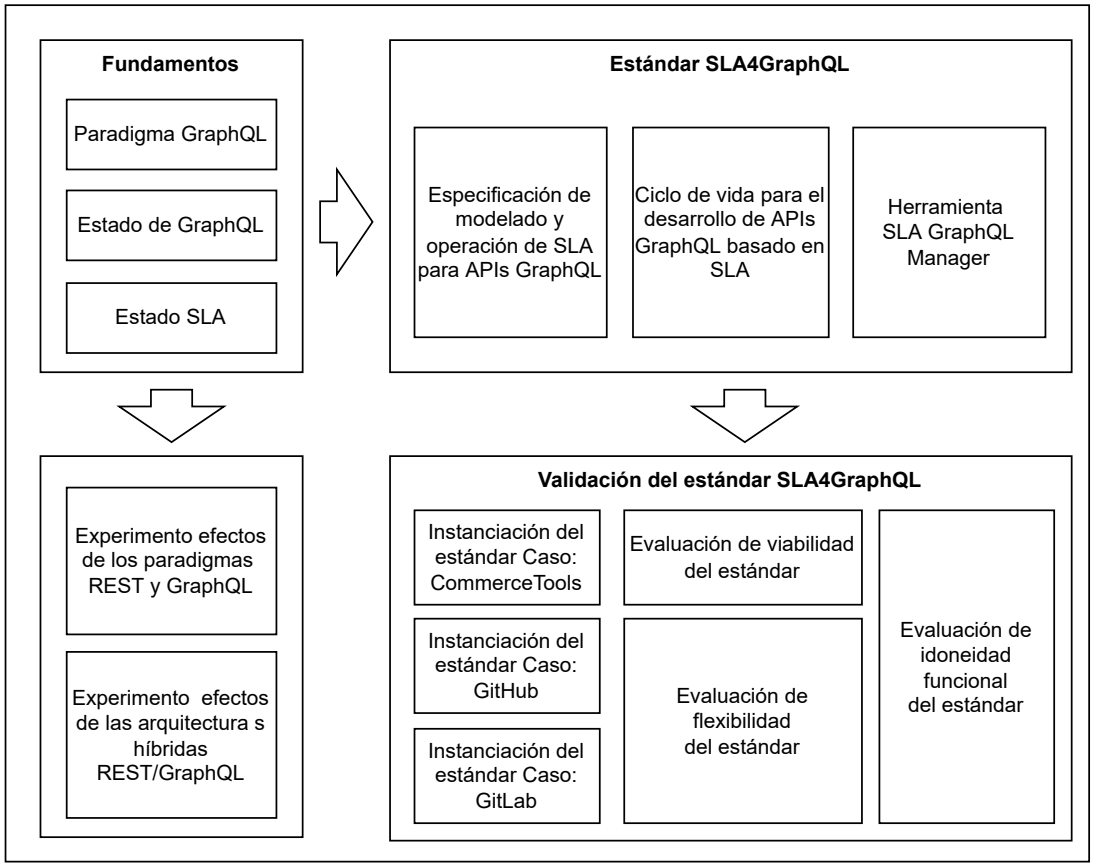


Figura 1.4: Resumen de contribuciones

1. *Entorno experimental*. Se define los siguientes elementos del entorno experimental de los experimentos para comparar las condiciones adecuadas de uso de los paradigmas REST y GraphQL, como también de las arquitecturas híbridas REST/-GraphQL: Objetivos, factores, tratamientos, variables, hipótesis, diseño, sujetos experimentales, y tareas experimentales.

2. *Operación*. Se define los siguientes elementos de la fase de operación de los experimentos para comparar las condiciones adecuadas de uso de los paradigmas REST y GraphQL, como también de las arquitecturas híbridas REST/GraphQL: Preparación, ejecución, recolección y validación de datos.

3. *Análisis e interpretación*. Se define los siguientes elementos de la fase de análisis e interpretación de datos de los experimentos para comparar las condiciones adecuadas de uso de los paradigmas REST y GraphQL, como también de las arquitecturas híbridas REST/GraphQL: Estadística descriptiva, depuración de datos, y/o test de hipótesis.

Luego, Basado en DSR se definió las siguientes actividades [119]:

1. *Identificación del problema y motivación*. Se define el problema que da lugar a una propuesta de investigación.

   En nuestro caso, se analizó los problemas encontrados en el manejo de los datos mediante APIs, concretamente utilizando el diseño arquitectónico REST y las alternativas que brinda el lenguaje de consultas GraphQL. Se identificó la necesidad de abordar un estudio de GraphQL como una alternativa a REST y el desarrollo de un modelo para describir contratos de niveles de servicio SLA para APIs GraphQL.

2. *Definición de los objetivos de la solución*. Es necesario conocer otras propuestas y especificar qué se espera del artefacto a desarrollar.

   Por lo cual, se revisó la literatura para proporcionar una visión general de la producción de la comunidad científica para encontrar tendencias y lagunas de conocimiento en torno al paradigma GraphQL mediante la realización de un estudio de mapeo sistemático (SMS, por sus siglas en inglés de *Systematic Mapping Study*).

   Las contribuciones en esta fase son:

   - *Análisis del estado general de GraphQL*. Tras seleccionar y revisar trabajos publicados entre 2015 y junio del 2021 que reporten estudios acerca de GraphQL, se identificó las principales instituciones, autores, y países destacados en la publicación acerca del tema. Se usó clasificadores de datos como *tablas*, *mapas de calor* y *diagramas de barra* para identificar los países clusters o grupos de autores y coautores destacados. Así mismo, se han clasificado los estudios según tipos de publicación, tipos y métodos de investigación utilizados para identificar las tendencias y vacíos de la investigación.
   - *Revisión del estado específico de GraphQL*. Tomando como base los trabajos identificados en la revisión anterior, se analizó el dominio y tipos de contribuciones técnicas realizadas con GraphQL, como también el detalle del uso, ejemplificación y mención de los componentes GraphQL en los estudios.

3. *Diseño y desarrollo*. Se da forma al artefacto.

   El artefacto propuesto es el *estándar SLA4GraphQL* para modelar y operativizar contratos de nivel de servicio (SLA) para APIs GraphQL. Una posible instancia-

ción del estándar son los *documentos SLA* que facilitan la descripción y operación de los SLA.

Las contribuciones en esta fase son:

- *Especificación para modelar SLA*. A partir de la propuesta SLA4OAI establecida por Gamez-Diaz [46, 71], se desarrolló un estándar que describe la *Especificación para modelar SLA* para APIs GraphQL, la cual, servirá como base para definir *documentos SLA* que describan contratos específicos de una API GraphQL.
- *Patrones lingüísticos*. Se ha definido el formato de serialización de datos *yalm* y el lenguaje *xpath* para modelar los acuerdos de nivel de servicio en *documentos SLA* para APIs GraphQL.
- *Ciclo de vida*. Se define un ciclo de vida para el desarrollo de APIs GraphQL basadas en SLA, y procesos principales para operativizar los SLA modelados.
- *Herramienta SLA GraphQL Manager*. Es una herramienta que se conforma por una librería *npm*[10] que valida los documentos SLA modelados con el estándar *SLA4GraphQL*, y un repositorio en la nube para registrar documentos SLA.

4. *Demostración*. Implica el uso del artefacto desarrollado en una o más instancias del problema analizando su aplicabilidad.

   Como primer paso de esta fase se instanció el estándar para especificar un *documento SLA* que modele los acuerdos de nivel de servicio a partir de las limitaciones de las APIs GraphQL de CommerceTools [25], GitHub [51], y GitLab [53] tomando en cuenta las limitaciones descritas en sus páginas oficiales.

5. *Evaluación*. Mediante evidencias empíricas, se analiza si el uso del artefacto cubre los objetivos esperados. Puede ser necesario realizar ajustes en el artefacto y volver, mediante un proceso iterativo, a la actividad de *diseño y desarrollo*.

   La contribución en esta fase es:

   - *Evaluación de viabilidad del estándar SLA4GraphQL*. Mediante la instanciación del estándar en el caso en concreto de la API de CommerceTools; para comprobar la cantidad de acuerdos que pudieron ser modelados a partir de las limitaciones documentadas de la API.

[10] https://docs.npmjs.com/about-npm

- *Evaluación de flexibilidad del estándar SLA4GraphQL*. Mediante la instanciación del estándar en los casos en concreto de las APIs de GitHub y GitLab; para comprobar la cantidad de acuerdos que pudieron ser modelados a partir de las limitaciones documentadas de la APIs.
- *Evaluación de la idoneidad funcional del estándar SLA4GraphQL*. Mediante la verificación del grado de funcionalidad que tiene el estándar SLA4GraphQL para modelar acuerdos de nivel de servicio, aplicado a los casos de: CommerceTools, GitHub, y GitLab.

6. *Comunicación*. Se publican los resultados de la investigación.

   Los resultados se reportarán mediante artículos científicos que se publiquen en conferencias internacionales, y revistas científicas relacionadas a la ingeniería del software de preferencia indexadas en JCR[11] o en clase 1 o 2 del Ranking GRIIN-SCIE[12].

La Tabla §1.1 muestra el detalle de las actividades correspondientes a las secciones de esta memoria.

1.5 Contexto de la Tesis

Esta tesis ha sido desarrollada en el contexto de las líneas de estudio del grupo de investigación de Ingeniería de Software Aplicado (ISA) de la Universidad de Sevilla.

Se encuadra en el ámbito de los siguientes proyectos de investigación:

- FEDER: Ministerio de Ciencia e Innovación – Agencia Estatal de Investigación under project HORATIO (RTI2018-101204-B-C21), and by Junta de Andalucía under projects APOLO (US-1264651).
- EKIPMENT-PLUS: Proyecto del Plan Andaluz de I+D+i, financiado por la Junta de Andalucía (P18-FR-2895, PAIDI 2020). Fecha de inicio: 01-01-2020. Fecha de finalización: 31-12-2023.
- UTN-FICA: Proyectos internos de investigación de la Facultad de Ingeniería en Ciencias Aplicadas de la Universidad Técnica del Norte, Ibarra, Ecuador. Fecha de inicio: 01-01-2019. Fecha de finalización: En ejecución.

[11] https://clarivate.com/webofsciencegroup/solutions/journal-citation-reports/
[12] http://scie.lcc.uma.es/

Tabla 1.1: Actividades de la metodología DSR y secciones correspondientes

Actividad	Descripción	Sección
Identificación del problema y motivación	Necesidad de realizar un estudio acerca de GraphQL para abordar la gobernanza de arquitecturas híbridas REST/GraphQL mediante contratos.	§1.1, §1.2
Definición de los objetivos de la solución	Facilitar la gobernanza de arquitecturas híbridas REST/-GraphQL mediante contratos de niveles de servicio para APIs GraphQL.	§1.3
Diseño y desarrollo	Se propone el estándar *SLA4GraphQL*. Una posible implementación es un *documento SLA* definida en el formato de serialización de datos *Yalm* y el lenguaje *Xpath* para modelar acuerdos de nivel de servicios para APIs GraphQL.	§8
Demostración	Se instancia la primera versión del estándar *SLA4GraphQL* en las APIs de CommerceTools, GitHub, y GitLab.	§11
Evaluación	Se valida el artefacto, mediante la evaluación de viabilidad, flexibilidad e idoneidad funcional de las instanciaciones del estándar *SLA4GraphQL*.	§11
Comunicación	Se publican los resultados de la investigación en esta memoria y en artículos científicos.	§12.2

1.6 Estructura del documento

La organización de la memoria es la siguiente:

Parte I. Introducción. Consta de dos capítulos; el presente capítulo §1 que plantea la motivación, relevancia del problema y la estructuración y dirección de la investigación. El capítulo §2 describe la metodología de investigación utilizada a lo largo de esta memoria.

Parte II. Estado de la cuestión. Se distinguen tres capítulos. El capítulo §3 introduce el marco conceptual denominado Paradigma GraphQL, que ilustra y ejemplifica la especificación formal de GraphQL. El capítulo §4 introduce la base conceptual de los acuerdos de nivel de servicio, en donde, se revisa sus principales conceptos y

propuestas existentes para gestionar SLA para APIs. Y en el capítulo §5 muestra una visión general y específica de GraphQL mediante el desarrollo de un *estudio de mapeo sistemático* para descubrir tendencias y lagunas de conocimiento del tema de estudio.

Parte III. Propuesta. Es el núcleo de la tesis, en donde se distinguen tres capítulos. El capítulo §6 propone un diseño experimental para analizar el proceso de desarrollo del software, con el propósito de comparar los efectos de los paradigmas de programación REST y GraphQL con respecto a la calidad del software. El capítulo §7 propone un diseño experimental para analizar el producto de software, con el propósito de comparar la eficiencia entre las arquitecturas híbridas REST/GraphQL y la arquitectura REST con respecto a la calidad del software. El capítulo §8 define un *estándar* para modelar y operativizar acuerdos de niveles de servicio (SLA) para APIs GraphQL.

Parte IV. Validación. Se distinguen tres capítulos. El capítulo §9 ejecuta el diseño experimental propuesto en el capítulo §6 para comparar los efectos entre los paradigmas de programación REST y GraphQL, luego se analiza estadísticamente los resultados y contesta las preguntas de investigación establecidas en el experimento. El capítulo §10 ejecuta el diseño experimental propuesto en el capítulo §7 para comparar la eficiencia entre las arquitecturas híbridas REST/GraphQL y la arquitectura REST, luego se analiza estadísticamente los resultados y contesta las preguntas de investigación establecidas en el experimento. El capítulo §11 valida el *estándar SLA4GraphQL* para modelar y operativizar acuerdos de niveles de servicio SLA propuestos en el capítulo §8 mediante la evaluación de viabilidad, flexibilidad e idoneidad funcional de la instanciación del estándar en tres APIs GraphQL públicas.

Parte V. Conclusiones. En el capítulo §12 se analiza el cumplimiento de los objetivos propuestos en esta memoria, se resumen las publicaciones derivadas de este trabajo, y se indican los principales retos existentes del tema como trabajo futuro.

Parte VI. Anexos. El anexo §A muestra los estudios primarios del mapeo sistemático de GraphQL realizado en el capítulo §5. El anexo §B muestra el paquete de laboratorio del experimento realizado en el capítulo §6.

Metodología de investigación

Una máquina puede hacer el trabajo de cincuenta hombres ordinarios.
Ninguna máquina puede hacer el trabajo de un hombre extraordinario.

Elbert Hubbard (1856 – 1915),
Filósofo, escritor, editor, y artista estadounidense

En este capítulo, se describe la metodología de investigación utilizada a lo largo del estudio doctoral, la cual, se reporta en esta memoria. El objetivo del capítulo es fundamentar conceptualmente los métodos de investigación aplicados en el desarrollo y documentación de los diferentes capítulos.

En la sección §2.1, se destaca una breve discusión del propósito de la utilización de los métodos de investigación en los diferentes capítulos de la memoria. En la sección §2.2 muestra la descripción del método de investigación "Estudio de mapeo sistemático". En la sección §2.3 muestra la descripción del método de investigación "Experimentación en la ingeniería de software". En la sección §2.4 muestra la descripción del enfoque de investigación "Ciencia del Diseño".

2.1 Introducción

A lo largo de este capítulo se describe los distintos enfoques y métodos de investigación utilizados en el desarrollo de la presente memoria. En donde, en el capítulo §3 se utilizó el método *Estudio de mapeo sistemático* para realizar una investigación sistemática de la literatura acerca de GraphQL; en los capítulos §6, §7, §9, §10 se utilizó el método de *experimentación en la ingeniería de software* para comprobar los efectos de las arquitecturas REST, GraphQL, e híbridas REST/GraphQL en la calidad del software; y en el capítulo §11 se utilizó el enfoque de la *Ciencia del diseño* para validar la propuesta *SLA4GraphQL* para modelar acuerdos de nivel de servicios (SLA) para APIs GraphQL, expuesta en el capítulo §8. Estos métodos se detallan a continuación.

2.2 Estudio de mapeo sistemático

El Estudio de Mapeo Sistemático (SMS, por sus siglas en inglés de Systematic Mapping Study), es un método para construir un esquema de clasificación de temas de estudio en un campo de interés. Contando el número de publicaciones para las categorías dentro de un esquema, se puede determinar la cobertura y la madurez del campo de investigación. Los resultados se presentan en mapas, gráficos o datos tabulados que muestran los estudios primarios (es decir, estudios inéditos que obtienen pruebas empíricas sobre un tema de interés [48]) existentes del tema de estudio en diferentes categorías. Los estudios de mapeo suelen abarcar una gama más amplia de publicaciones, ya que el análisis se centra en los resúmenes y los términos clave [120]. En la ingeniería de software, existen varias guías para realizar estudios de mapeos sistemáticos como por ejemplo las propuestas de Petersen *et al.* [122], Kitchenham [87], Kitchenham y Charters [90], Kitchenham y Brereton [89], Budgen *et al.* [15], Arksey and O'Malley [2], Biolchini *et al.* [8], y Petticrew y Roberts [123].

A continuación, se describe los pasos principales del proceso para llevar a cabo un estudio de mapeo sistemático [122]:

Planificación: Planificación del mapeo contiene: Identificación de la necesidad y alcance, Identificación de estudios, Extracción y clasificación de datos, Visualización, y Amenazas a la validez.

Realización: Al realizar el mapeo hay que aplicar el proceso definido durante la fase de planificación. Se recomienda registrar la información en todas las fases del

proceso.

Informe: Se recomienda seguir la siguiente estructura: introducción, trabajos relacionados, método de investigación, resultados, discusión, conclusiones, y apéndices.

Evaluación del proceso: Se define rúbricas sobre la base de la lista de comprobación de la calidad, se definen criterios para la identificación de la necesidad, la identificación del estudio, la extracción y clasificación de los datos y la discusión de la validez.

Divulgación: Por lo general los resultados de los estudios de mapeo sistemáticos se publican en revistas, más que en conferencias.

2.3 Experimentación en la ingeniería de software

Los experimentos investigan una hipótesis comprobable en la que se manipulan una o varias variables independientes para medir su efecto sobre una o varias variables dependientes [33]. Los experimentos se los realiza cuando se pretende controlar una situación y manipular el comportamiento de forma directa, precisa y sistemática; los experimentos pueden orientarse a las personas o a la tecnología, cuando se orienta a los seres humanos, éstos aplican diferentes tratamientos a los objetos; cuando son orientados a la tecnología, normalmente se aplican diferentes herramientas a diferentes objetos [186]. En la ingeniería de software, existen varias guías para realizar experimentos como por ejemplo las propuestas por Wohlin *et al.* [186], Jedlitschka *et al.* [78], Singer [148], Juristo y Moreno [82], y Kitchenham *et al.* [91].

Para la presente memoria, se utilizó la guía propuesta por Wohlin *et al.* [186] para aplicar la experimentación en varios estudios, en donde, se siguió la estructura de la guía que muestra en la Tabla §2.1 y el proceso que muestra la Figura §2.1.

Tabla 2.1: Estructura de la guía de Wohlin *et al.*

Fases	Actividades	Descripción

Continuación de la Tabla §2.1

Alcance	Alcance del experimento	Se define el objetivo y las metas del experimento: Objeto de estudio (¿qué se estudia?), Propósito (¿cuál es la intención?), Enfoque cualitativo (¿qué efecto se estudia?), Perspectiva (¿el punto de vista de quién?), y Contexto (¿dónde se realiza el estudio?).
Planificación	Selección del contexto Formulación de hipótesis Selección de variables Selección de sujetos Diseño del experimento Instrumentación Evaluación de la validez	Se determina en detalle el contexto del experimento. Se establece formalmente las hipótesis (nula y alternativa). Se determina variables (independiente(s) y dependiente(s)). Se identifican los sujetos del estudio. Se diseña el experimento, se prepara la instrumentación. Se considera la validez de los resultados (interna, externa, constructo y conclusión).
Operación	Preparación Ejecución Validación de datos	Se prepara a los sujetos, así como el material necesario. En la ejecución se debe asegurar que el experimento se lleva a cabo según el plan y el diseño del mismo. Procurar que los datos recogidos sean correctos.
Análisis e interpretación	Estadística descriptiva Reducción del set de datos Prueba de hipótesis	Se trata de entender los datos utilizando estadística descriptiva. Se debe considerar si el conjunto de datos debe reducirse. Luego realizar una prueba de hipótesis mediante pruebas estadísticas.
Presentación y empaquetado	Presentación y empaquetado	Es la documentación de los resultados que pueden realizarse mediante un artículo de investigación para su publicación, y un paquete de laboratorio para su replicación.

Fin de tabla

2.4 Ciencia del diseño

La Ciencia del Diseño (DSR, por sus siglas en inglés de Design Science Research) es fundamentalmente un paradigma de resolución de problemas, trata de crear innovaciones que definan las ideas, las prácticas, las capacidades técnicas y los productos mediante los cuales el análisis, el diseño, la implementación, la gestión y el uso de los sistemas de información puedan llevarse a cabo de forma eficaz y eficiente [67]. La figura §2.2 muestra el modelo del proceso para realizar investigaciones usando el enfoque de la ciencia del diseño.

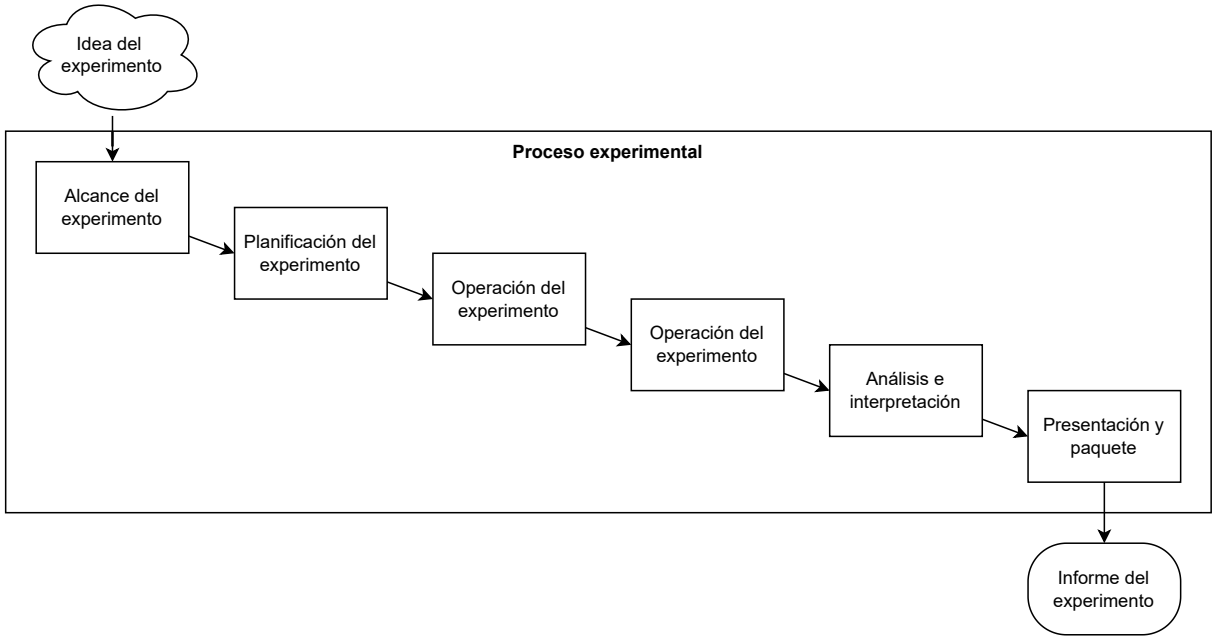


Figura 2.1: Resumen del proceso experimental

Los pasos principales del proceso de la ciencia del diseño son [119]:

Identificación del problema y motivación: Definir el problema de investigación específico para justificar el valor del desarrollo de una solución efectiva. Atomizar el problema conceptualmente puede ser útil para que la solución pueda captar la complejidad del problema.

Objetivos de una solución: Inferir los objetivos de una solución a partir de la definición del problema. Los objetivos pueden ser cuantitativos, cualitativos, o podrían deducirse racionalmente de la especificación del problema. Los recursos necesarios para ello son el conocimiento del estado de los problemas y de las soluciones actuales y su eficacia.

Diseño y desarrollo: Crear la solución de artefactos, estos deben ser potencialmente definidos de forma amplia, constructos, modelos, métodos o instancias. Se determina la funcionalidad deseada del artefacto y su arquitectura para luego crear el artefacto real. Los recursos necesarios para pasar de los objetivos al diseño y el desarrollo incluyen el conocimiento de la teoría que puede aportarse como solución.

Demostración: Demostrar la eficacia del artefacto para resolver el problema. Esto

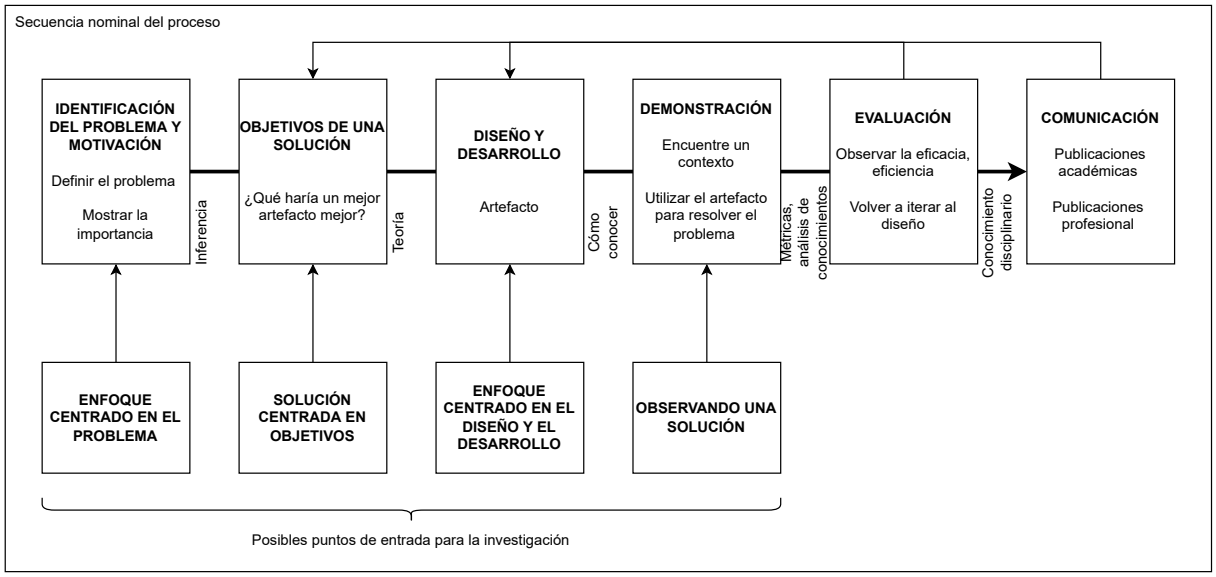


Figura 2.2: Modelo de proceso de investigación en ciencias del diseño

podría implicar su uso en la experimentación, la simulación, un estudio de caso, una prueba u otra actividad apropiada. Los recursos necesarios para la demostración incluyen el conocimiento efectivo de cómo utilizar el artefacto para resolver el problema.

Evaluación: Observar y medir la eficacia del artefacto para resolver el problema. Esta actividad consiste en comparar los objetivos de una solución con los resultados reales observados del uso del artefacto en la demostración, mediante el uso de métricas y técnicas de análisis pertinentes. Al final de esta actividad, los investigadores pueden decidir si volver al ”diseño y desarrollo” para intentar mejorar la eficacia del artefacto o continuar con la comunicación y dejar las mejoras para proyectos posteriores.

Comunicación: Comunicar el problema y su importancia, el artefacto, su utilidad y novedad, el rigor de su diseño y su eficacia, mediante publicaciones académicas o profesionales, los cuales, puede utilizar la estructura de este proceso para estructurar sus investigaciones empíricas (definición del problema, revisión de la literatura, desarrollo de la hipótesis, recogida de datos, análisis, resultados discusión y conclusión).

Parte II

Estado de la cuestión

Paradigma GraphQL

La inteligencia consiste no sólo en el conocimiento,
sino también en la destreza de aplicar los conocimientos en la práctica.

Aristóteles (384 a.C. – 322 a.C.),
Filósofo y erudito griego

En este capítulo, se introduce el marco conceptual denominado Paradigma GraphQL, *que ilustra y ejemplifica la especificación formal de GraphQL (versión junio del 2018). Además, se responde la pregunta de investigación* $\mathbf{PI}_1$. *¿Cuál es el estado actual de GraphQL?.*

En la sección §3.1, se destaca una breve discusión de los temas a tratar en este capítulo. En la sección §3.3, se presenta una caracterización de los principales componentes GraphQL. En la sección §3.4, se introduce la estructura conceptual y ejemplificación de los componentes GraphQL. En la sección §3.5, se valida el paradigma GraphQL mediante la creación y validación de un artefacto tecnológico basado en la implementación del paradigma. Por último, la sección §3.6, se muestra las principales conclusiones del capítulo.

3.1 INTRODUCCIÓN

En las secciones §3.3, y §3.4, se introduce el marco conceptual denominado *Paradigma GraphQL* que ilustra y ejemplifica la estructura conceptual y funcional de la especificación formal de GraphQL (versión junio del 2018) [162]. Además, en este capítulo se concreta la pregunta de investigación $\mathbf{PI}_1$: ¿Cuál es el estado actual de GraphQL? con la sub pregunta:

- $\mathbf{PI}_{1.1}$: ¿Se pueden implementar aplicaciones complejas con suficiente calidad usando el paradigma GraphQL?

Se responde a $\mathbf{PI}_{1.1}$ mediante el diseño y validación de un artefacto (implementación del *paradigma GraphQL*), basado en el enfoque de investigación *DSR*. También, es necesario aclarar que la pregunta de investigación $\mathbf{PI}_1$ se complementará con seis sub preguntas adicionales en el capítulo §5.

3.2 ANTECEDENTES

Antes de introducir el paradigma GraphQL, se describe el concepto de REST, y sus principales diferencias con GraphQL.

3.2.1 REST

REST, por sus siglas en inglés de REpresentational State Transfer, es un estilo arquitectónico para sistemas hipermedia distribuidos, creado por Roy Thomas Fielding en 2000 [42]. REST es un conjunto de conceptos de diseño basados en características y requisitos del software de red, que tiene como objetivo reducir la complejidad del desarrollo, mejorar la escalabilidad de los sistemas y reducir la carga de la comunicación [118, 134].

La arquitectura de servicios REST plantea varios criterios principales de diseño para su aplicación en la web: i) todas las cosas en Internet se abstraen como recursos; ii) cada recurso corresponde a un identificador de recurso único, es decir, una URI; iii) operar los recursos a través de la interfaz del conector genérico aplicando acciones estándar como GET, POST, PUT, DELETE, HEAD del protocolo HTTP; iv) varias operaciones sobre los recursos no cambiarán el identificador del recurso; y v) todas las

operaciones son sin estado [57]. Las limitaciones mencionadas han sido clave para el éxito de la arquitectura REST en la Web; sin embargo tal como se revisó en la sección §1.1, REST ha mostrado varios problemas, en donde, GraphQL se presenta como una alternativa para mejorar varios de estos problemas. Por tal razón, a continuación, se describe las diferencias entre las características de estos dos paradigmas.

3.2.2 Diferencias entre REST y GraphQL

Antes de mencionar las diferencias, se describe las similitudes entre estos patrones arquitectónicos. El sistema de tipos de GraphQL es similar a la definición de recursos en la arquitectura REST, en donde, no hay consultas ad-hoc, sino que los tipos del sistema se definen de antemano, especificando los posibles parámetros y el formato de salida. Tanto GraphQL como REST son independientes del lenguaje de programación subyacente y de los backends de bases de datos utilizados. La comunicación con el cliente se basa en un patrón de intercambio de mensajes petición-respuesta [113].

Entre las principales diferencias de REST con GraphQL se destacan las siguientes: REST es un estilo arquitectónico, y GraphQL es un lenguaje de consulta y tiempo de ejecución de APIs. La interfaz REST proporciona múltiples puntos finales (endpoints) de la API, uno para cada recurso; en el caso de GraphQL tiene un único punto final. REST utiliza generalmente las acciones POST, GET, PUT, y DELETE de http para manipular los datos; en cambio GraphQL utiliza consultas (queries) y mutaciones (mutations) con la acción POST sobre su único endpoint. REST admite el almacenamiento en caché y documentos parciales, estas características, así como la libertad de elección con respecto a los formatos de intercambio de datos utilizados, no son posibles con GraphQL por el momento [11, 113, 173]. En la Tabla §3.1 se muestra un resumen de las diferencias entre las propiedades de REST y GraphQL.

3.3 FUNDAMENTOS DE GRAPHQL

La especificación formal de *GraphQL* lo define como un lenguaje de consulta y un motor de ejecución para describir capacidades y requisitos de modelos de datos para aplicaciones cliente-servidor [162]. Para implementar servidores de aplicaciones GraphQL (*API GraphQL*), no es necesario utilizar un lenguaje de programación o un mecanismo de persistencia específicos. En su lugar, GraphQL codifica las capacidades del modelo de datos basado en un sistema de tipos utilizando un lenguaje uniforme

Tabla 3.1: Diferencias entre REST y GraphQL

Propiedad	REST	GraphQL
Tecnología	estilo arquitectónico	lenguaje de consulta
Manipulación de datos	http: POST, GET, PUT, DELETE	queries, mutations
Puntos finales de la API	múltiple	único
Formato de los mensajes para las consultas	basado en cadenas	cadenas
Formato de mensaje para las mutaciones	cualquiera	cadenas
Formato de mensaje para la respuesta	cualquiera, especialmente hipermedia	JSON
Sistema de tipos Características incorporadas	débilmente tipificado, sin metadatos legibles por la máquina cacheable	fuertemente tipado, metadatos para la validación de la consulta de introspección, introspección

que responde a los siguientes principios de diseño: centrado en el producto, jerárquico, tipificación fuerte, consultas especificadas por el cliente e introspectivo [39, 162].

Se comenzó a estructurar el paradigma mostrando la interacción entre los siguientes componentes de alto nivel de GraphQL: el lenguaje, su gramática, el sistema de tipos, la introspección, el motor de ejecución, y el motor de validación). La figura §3.1 muestra como esta interacción comienza en (1) las herramientas en tiempo de desarrollo que utilizan el lenguaje GraphQL, su gramática, y el lenguaje de descripción de interfaces (IDL, por sus siglas del inglés de Interface Description Language) para definir un sistema de tipos o una extensión del sistema de tipos de un Servicio GraphQL. (2) La generación del servicio del API GraphQL se complementa con la implementación de Resolvers en algún lenguaje de programación para devolver las propiedades definidas en el sistema de tipos.(3) Desde las herramientas de cliente, se crea un documento con definiciones de operaciones ejecutables (query, mutation, subscription) o fragments para enviar una petición de ejecución al servicio GraphQL. Antes de enviar la solicitud, las herramientas del cliente proporcionan una validación de la sintaxis utilizando la introspection del servicio GraphQL (representada con un círculo entrecortado con la palabra ”INT” en la sección ”Herramientas del cliente”). El servicio GraphQL recibe la petición en el motor de validación para validar que el documento sólo contenga definiciones ejecutables, y que la solicitud no contenga errores; luego

pasa la solicitud al motor de ejecución o reporta errores en la respuesta del servicio. (4) El servicio GraphQL ejecuta la solicitud y envía el resultado como un documento de respuesta (normalmente en formato JSON[1]) a las herramientas del cliente. El documento de respuesta también puede incluir mensajes de error si hay errores durante la ejecución.

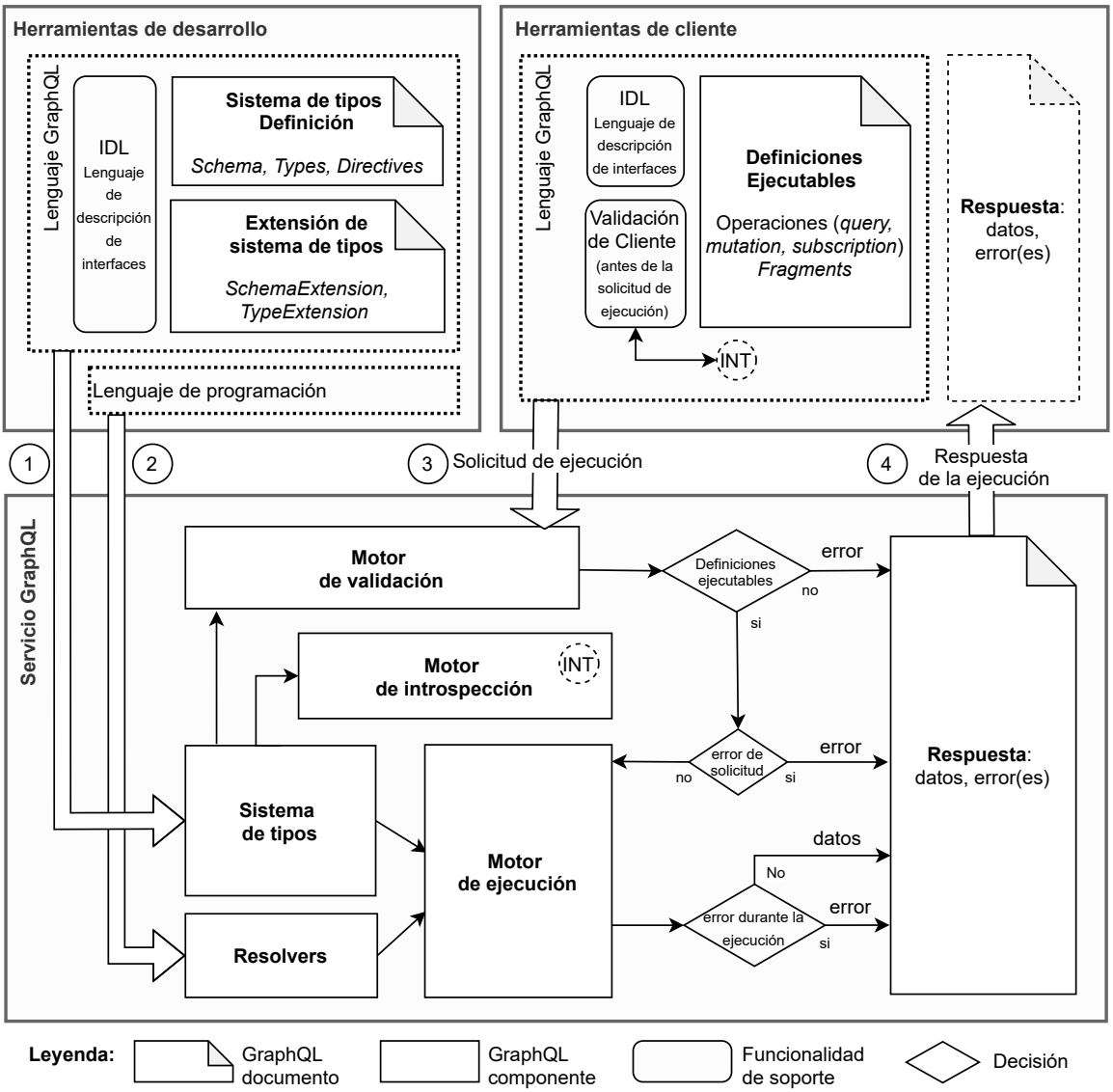


Figura 3.1: Componentes principales del paradigma GraphQL

Una vez explicado cómo interactúan los componentes de alto nivel, en la siguiente sección §3.4 se revisa su estructura y descripción técnica de los componentes específicos que conforman el paradigma GraphQL.

[1]Notación de Objetos de JavaScript (JSON, por sus siglas en inglés de JavaScript Object Notation): es un formato ligero de intercambio de datos [80].

3.4 SEMÁNTICA OPERACIONAL

El lenguaje GraphQL utiliza una gramática sintáctica para definir un sistema de tipos o consultar un servicio GraphQL mediante documentos. Un documento GraphQL puede contener diferentes instancias de definición (ejecutable, sistema de tipo, extensión de sistema de tipo) [162]. La figura §3.2 muestra los componentes principales de cada tipo de definición; los mismos que se detallarán y ejemplificarán a continuación.

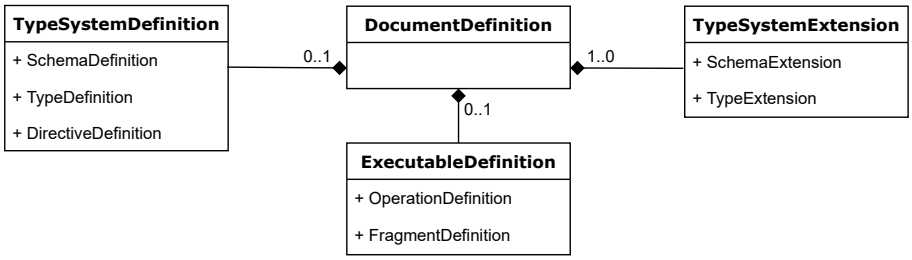


Figura 3.2: Definición de documentos GraphQL

Además, el lenguaje GraphQL incluye un lenguaje de descripción de interfaces (IDL por sus siglas en inglés de Interface Definition Language) para describir el sistema de tipos de un servicio GraphQL, que es usado por las herramientas de desarrollo o de cliente para proporcionar utilidades como la generación de código de cliente o el bootstrapping del servicio [162].

3.4.1 Definición del Sistema de Tipos *(Type System Definition)*.

Define las capacidades de datos, y los tipos de entrada de las variables de consulta del *Type System* de un servidor GraphQL. Un Documento GraphQL que contenga un *TypeSystemDefinition* no debe ser ejecutado por los servicios de ejecución GraphQL. En esta sección se describe los componentes y en la figura §3.3 se ilustra la estructura conceptual e interacción de los componentes de la definición del sistema de tipos.

Esquema *(Schema)*: conforma la colección de capacidades del sistema de tipo de un servicio GraphQL. Su definición se la realiza en términos de tipos, directivas, y tipos de operación: query, mutation, y subscription.

Tipos *(Types)*: son la definición concreta de los tipos, que pueden incluir tipos con nombre, abstractos o envolventes.

Los *tipos con nombre* son una unidad básica del esquema GraphQL, que puede definirse como:

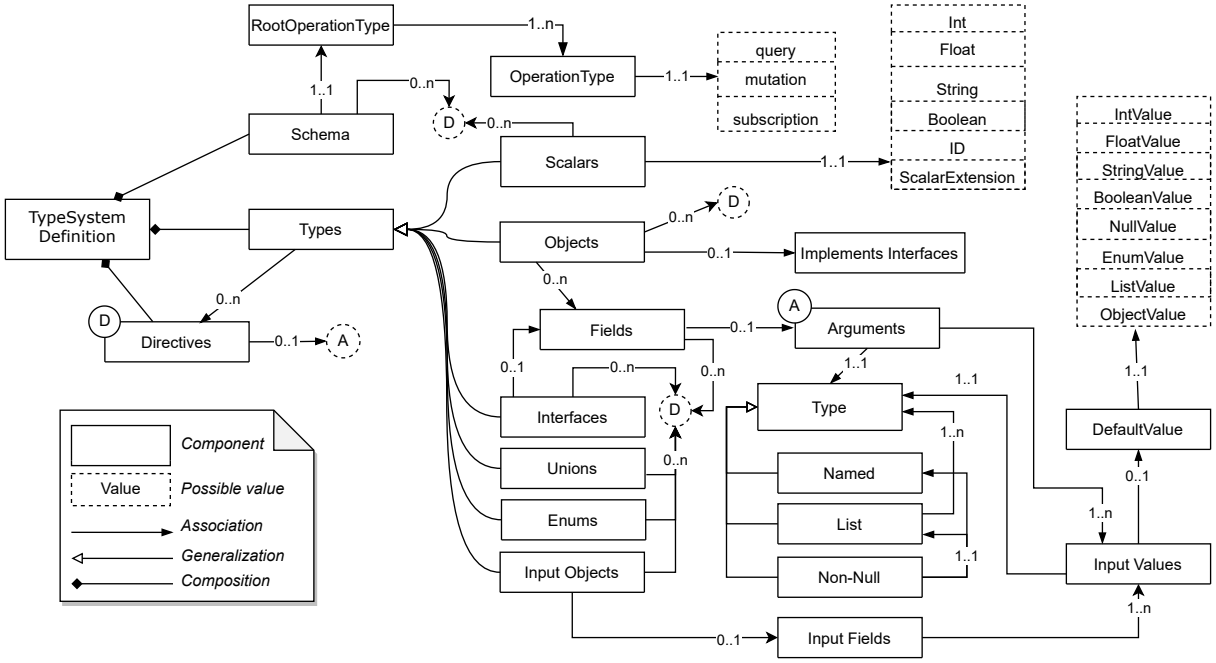


Figura 3.3: Definición del sistema de tipos *(TypeSystemDefinition)*

- Escalares *(Scalars)*: representan un valor primitivo como *string*, *integer*, *float*, *boolean*, o *ID* (identificador único).
- Enumeraciones *(Enums)*: describe un conjunto de posibles valores.
- Objetos *(Objects)*: definen un conjunto de campos *(fields)*, donde cada campo es de otro tipo en el sistema, lo que permite la definición arbitraria de jerarquías de tipos. Los campos son conceptualmente funciones que devuelven valores, y ocasionalmente aceptan argumentos *(arguments)* que alteran su comportamiento. Estos argumentos suelen asignarse directamente a una función dentro de la implementación de un servidor GraphQL
- Objetos de entrada *(Input objects)*: define un conjunto de campos de entrada; pueden ser escalares, enumeraciones, u otros objetos de entrada.

Los *tipos abstractos* deben ser primero resueltos a un objeto de "tipo con nombre":

- Interfaces *(Interfaces)*: definen una lista de campos.
- Uniones *(Unions)*: definen una lista de posibles tipos.

Por último, los *tipos envoltura* envuelven o contienen un "tipo con nombre":

- Lista *(List)*: describe a un campo como una lista de otros tipos.

- Non-null *(Non-null)*: devuelve otro tipo y denota que el valor resultante nunca será nulo.

Directivas *(Directives)*: proporcionan una manera de describir el comportamiento alternativo de ejecución y validación de tipos en un documento GraphQL. Las directivas pueden utilizarse para describir información adicional para tipos, campos, fragmentos y operaciones.

A continuación, y a lo largo de esta sección se complementa la conceptualización del paradigma GraphQL con ilustraciones y ejemplos de sus componentes. Para este propósito, utilizamos la implementación de un servicio GraphQL de Star Wars que muestra la estructura básica de las películas y sus personajes.

a)
```
schema {
  query: Query
  mutation: Mutation
}
```

b)
```
type Query {
  humans(id: Int): [Human]
  droid(id: ID!): Droid
}
```

c)
```
type Mutation {
  createHuman(human: HumanInput): Human
}
```

d)
```
enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}
```

e)
```
type Character {
  name: String!
  appearsIn: [Episode]!
}
```

f)
```
interface Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
}
```

g)
```
type Human implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  totalCredits: Int
}
```

h)
```
type Droid implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  primaryFunction: String
}
```

i)
```
input HumanInput {
  name: String!
  friends: [String]
  appearsIn: [String]!
  totalCredits: Int
}
```

j)
```
union SearchResult = Human | Droid
```

Figura 3.4: Ejemplo de un sistema de tipos de GraphQL

La figura §3.4 muestra algunos ejemplos que contienen los componentes esenciales de la definición de un sistema de tipos [153]: a) define el esquema de servicio GraphQL con las operaciones disponibles (consulta y mutación); b) define los tipos de consulta (humans, droid), que reciben argumentos de tipo escalar y devuelven una lista de objetos de tipo; c) define la mutación (createHuman), que recibe un argumento de tipo de entrada y devuelve un tipo de objeto; d) define el tipo enumeración (Episode), que especifica un conjunto de valores e) define el tipo objeto (Character), que contiene un conjunto de campos. El símbolo (!) en el campo appearsIn significa que no acepta valores nulos; f) define el tipo Interfaz (Carácter) que contiene campos adicionales al objeto de tipo Carácter; g) define la implementación (Human) de la interfaz (Character) que contiene los campos de la interfaz y un campo adicional (totalCredits); h) define la implementación (Droid) de la interfaz (Character) que contiene los campos de la interfaz

y un campo adicional (primaryFunction); i) define el objeto de entrada (HumanInput) que contiene un conjunto de campos utilizados en la mutación createHuman; y j) define SearchResult como un tipo de unión utilizado para buscar datos en los tipos Human o Droid.

3.4.2 Definición de Ejecutables *(Executable Definition).*

Se considera documentos ejecutables cuando son definidos por operaciones *OperationDefinition* o fragmentos *(fragments)* [162]. A continuación, se describen los componentes de las definiciones ejecutables; y en la figura §3.5 se muestra la relación y estructura conceptual de estos componentes.

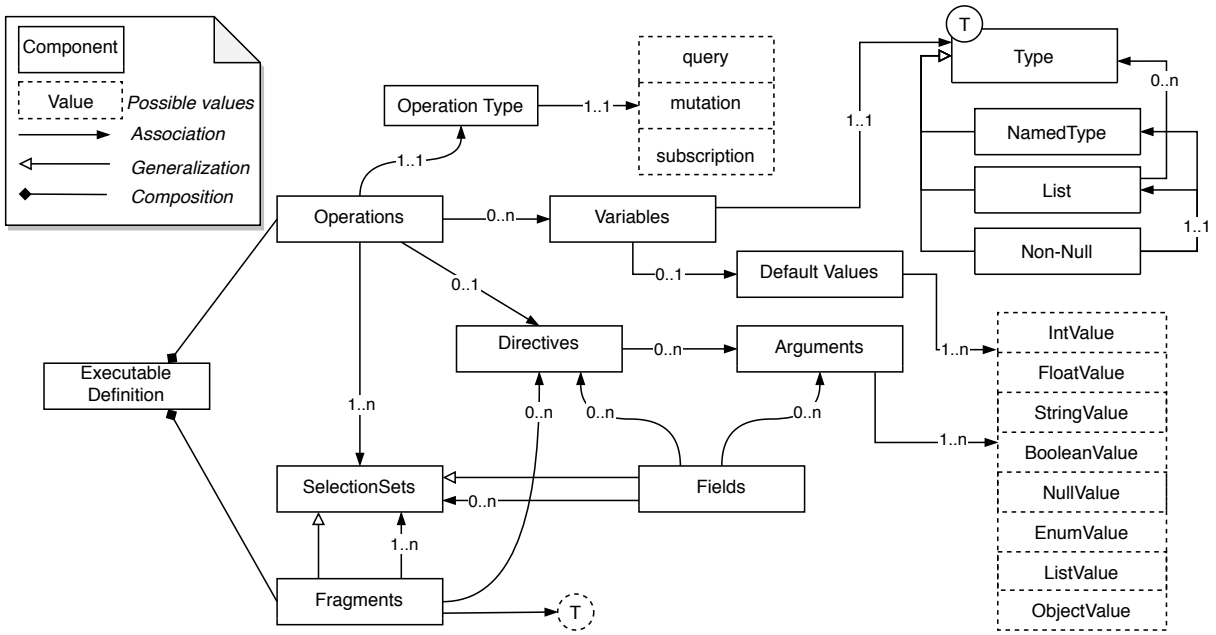


Figura 3.5: Definición de ejecutables en GraphQL

Operaciones *(Operations)*: son los elementos principales de las definiciones ejecutables, y pueden ser de tipo consulta *(query)*, que son de sólo lectura y obtienen datos; mutación *(mutation)* que primero escriben y luego obtienen datos; y suscripción *(subscription)*, que son peticiones de larga duración que obtienen datos en respuesta a eventos de la fuente.

Variables *(Variables)*: se introducen para maximizar la reutilización de las consultas GraphQL parametrizadas con variables, evitando la costosa construcción de cadenas en los clientes durante el tiempo de ejecución.

Valores de Entrada *(Input Values)*: pueden ser valores escalares, valores de enumeración, listas u objetos de entrada.

Directivas *(Directives)*: son una forma de describir comportamientos alternativos de ejecución y validación de tipos en un documento GraphQL; se utilizan de forma similar a descrita en las directivas del sistema de tipos.

Argumentos *(Arguments)*: alteran el comportamiento de los campos que ocasionalmente los aceptan. Los argumentos de las funciones se suelen asignar dentro de una implementación del servidor GraphQL.

Conjuntos de Selección *(Selection Sets)*: se componen principalmente de campos. Su propósito es restringir las solicitudes para seleccionar sólo el subconjunto necesario de la información, evitando el exceso, y falta de obtención de datos.

Campos *(Fields)*: describen la información disponible y pueden representar datos complejos o relaciones con otros datos; también se consideran funciones conceptuales que devuelven valores.

Fragmentos *(Fragments)*: es la unidad básica de composición en GraphQL que permite reutilizar selecciones de campos que se repiten habitualmente.

La figura §3.6 muestra ejemplos de las operaciones más utilizadas (consultas y mutaciones) de la definición ejecutable [153] relativa al sistema de tipos definido en la figura §3.4, en concreto: a) define la operación (NewHumans) que ejecuta la mutación (createHuman). También muestra tres formas de ejecutar el tipo de consulta (humans); b) ejecuta la consulta, pero sin definir el nombre de la petición; c) define la petición (queryOneHuman) que ejecuta la consulta acompañada del parámetro ”id” con el valor de uno; y d) define la petición (queryWithFragment) que ejecuta la consulta utilizando el fragmento (FragmentTwoFields) que hace referencia a un conjunto de campos específicos de la implementación (Human). Las herramientas de cliente se encargan de enviar estas solicitudes para su ejecución en el servicio GraphQL.

3.4.3 Respuesta *(Response)*.

La respuesta de una operación GraphQL es un mapa comúnmente en formato JSON, que contiene tres entradas: datos, errores, y/o extensiones. Los datos contienen el resultado de la ejecución de la operación solicitada. Los errores aparecen cuando la

a) Mutation example

```
mutation NewHumans{
    createHuman(name:"Leia Organa", friends:["Han Solo", "R2-D2"],
        appearsIn:["NEWHOPE", "EMPIRE", "JEDI"], totalCredits: 10000000)
        {   id
            name
            friends{ name }
        }
}
```

b) Query example 1

```
{ humans {
    id
    name
    friends{ name }
  }
}
```

c) Query example 2

```
query queryOneHuman{
  humans (id:1) {
    id
    name
    friends{ name }
  }
}
```

d) Query example 3

```
query queryWithFragment{
  humans (id:1) {
    ...fragmentTwoFields
    friends{ name }
  }
}
fragment fragmentTwoFields on Human {
  id
  name
}
```

Figura 3.6: Ejemplos de definición de ejecutables en GraphQL

ejecución ha encontrado un error. La extensión está reservada para que los implementadores puedan extender el protocolo sin restricciones de contenido [124, 153, 162]. La figura §3.7 muestra la respuesta a las solicitudes de ejecución de las operaciones de mutación y consulta realizadas en la figura §3.6.

```
{
    "data": {
        "humans": [
            {
                "id": 1,
                "name": "Leia Organa",
                "friends": [ "Han Solo", "R2-D2" ]
            }
        ]
    }
}
```

Figura 3.7: Ejemplo de respuesta en GraphQL

3.4.4 Introspección *(Introspection).*

Los servidores GraphQL soportan la introspección de su esquema mediante consultas al sistema de tipos utilizando el mismo lenguaje GraphQL [162]. La figura §3.8 muestra: a) una porción del sistema de tipos establecido en la figura §3.4; b) una consulta de introspección; y c) la respuesta de esa consulta.

3.4.5 Validación *(Validation).*

GraphQL con la validación verifica si una petición es sintácticamente correcta y se asegura de que es inequívoca y sin errores de acuerdo con un esquema GraphQL dado. La validación se realiza antes de la ejecución. Sin embargo, un servicio GraphQL puede ejecutar una petición sin validarla explícitamente si se sabe que esa misma petición ha

a) Type System

```
enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}

type Character {
  name: String!
  appearsIn: [Episode]!
}
```

b) Query to GraphQL Introspective

```
query queryToIntrospective {
  __type(name: "Character") {
    kind
    name
    fields {
      name
      type {
        ofType {
          name
          kind
          ofType {
            name
            enumValues {
              name
            }
          }
        }
      }
    }
  }
}
```

c) Response - Query to GraphQL Introspective

```
{
  "data": {
    "__type": {
      "kind": "OBJECT",
      "name": "Character",
      "fields": [
        { "name": "name",
          "type": {
            "ofType": {
              "name": "String",
              "kind": "SCALAR",
              "ofType": null } }
        },
        { "name": "appearsIn",
          "type": {
            "ofType": {
              "name": null,
              "kind": "LIST",
              "ofType": {
                "name": "Episode",
                "enumValues": [
                  { "name": "NEWHOPE" },
                  { "name": "EMPIRE" },
                  { "name": "JEDI" } ] } } }
        }
      ]
    }
  }
}
```

Figura 3.8: Ejemplo de introspección en GraphQL

sido validada anteriormente. Las definiciones del sistema de tipos y las extensiones no son ejecutables y por lo tanto no se consideran durante la ejecución [162].

3.4.6 Ejecución *(Execution).*

La ejecución de GraphQL sólo considerará las definiciones ejecutables de Operaciones y Fragmentos. Mediante la ejecución de una solicitud enviada desde las herramientas de cliente, genera una respuesta en el contexto del universo de datos disponibles en un sistema de tipos proporcionado por el servicio GraphQL. Sólo las peticiones que pasan las reglas de validación son realmente ejecutadas; si tiene errores de validación, serán reportados en la lista de ”errores” dentro de la respuesta, y la petición fallará sin ejecución [162]. Cabe mencionar que existen casos de peticiones que pasan la validación, pero durante la ejecución presentan errores de datos, en esos casos, la ejecución devuelve el resultado de los datos, pero también devuelve la descripción de los errores ocurridos.

3.4.7 Resolutores *(Resolvers)*.

Cada campo de cada tipo está respaldado por una función llamada ***resolver*** que el desarrollador implementa con algún lenguaje de programación en el servicio de GraphQL. Cuando se ejecuta un campo, se llama al resolver correspondiente para producir el valor de la respuesta [124, 162].

3.5 Validación del Paradigma GraphQL

3.5.1 Introducción

El objetivo de este apartado es validar el paradigma GraphQL introducido en las secciones §3.1, §3.3, y §3.4. Para lo cual, se utilizó la metodología del enfoque de investigación *Ciencia del diseño* (DSR) que indica como evaluar artefactos para resolver problemas organizacionales identificados [175]. En este sentido, el artefacto a validar será una *prueba de concepto* de la implementación del paradigma GraphQL.

A continuación, se resume las actividades propuestas en DSR [119], y aplicadas en el contexto de la prueba de concepto:

1. *Identificación del problema y motivación*. Se define el problema que da lugar a una propuesta de investigación.

   En este segmento, se hace referencia a la problemática presentada en el capítulo §1, la cual, motiva la validación del paradigma GraphQL y el planteamiento de una pregunta de investigación.

2. *Definición de los objetivos de la solución*. Es necesario conocer otras propuestas y especificar qué se espera del artefacto a desarrollar.

   El objetivo definido es contestar la pregunta de investigación mediante la creación y validación de un artefacto tecnológico. Para lo cual, se definió: *i)* El tema de implementación del artefacto; *ii)* Población y muestra del estudio; *iii)* Encuesta de aceptación del tema de implementación; *iv)* Propuestas existentes del tema de implementación; y *v)* Identificación de principales características y funcionalidades para el artefacto.

3. *Diseño y desarrollo*. Se da forma al artefacto.

En el diseño del artefacto se definió: Requerimientos, Diagrama de procesos, arquitectura de software, diagrama inicial de la base de datos. En el desarrollo se implementó el artefacto diseñado siguiendo el marco de trabajo SCRUM[2].

4. *Demostración*. Implica el uso del artefacto desarrollado en una o más instancias del problema analizando su aplicabilidad.

   En este caso, se desplegó el artefacto desarrollado en la nube, para un fácil acceso, y así constatar su aplicabilidad.

5. *Evaluación*. Mediante evidencias empíricas, se analiza si el uso del artefacto cubre los objetivos esperados. Puede ser necesario realizar ajustes en el artefacto y volver, mediante un proceso iterativo, a la actividad de *diseño y desarrollo*.

   En este punto, se realiza la evaluación de la calidad en uso del artefacto usando las normas ISO/IEC 25010 [75], 25022 [76], y 25040 [73]. Para recabar las evidencias empíricas se realizó un taller práctico y una encuesta de satisfacción del uso de la aplicación.

6. *Comunicación*. Se publican los resultados de la investigación.

   La comunicación se realiza a través de esta memoria, y del artículo de conferencia *Quality in use evaluation of a GraphQL implementation* [129] presentado en el *XVI Multidisciplinary International Congress on Science and Technology - CIT 2021*[3] realizado del 14 al 18 de junio 2021 de manera virtual.

3.5.2 Identificación del problema y motivación

El problema se lo identificó y describió en el capítulo §1, el cual, a su vez motiva a la validación del paradigma GraphQL introducido en las secciones anteriores §3.1, §3.3, y §3.4 para contestar la sub pregunta de investigación:

- **PI**$_{1.1}$: ¿Se pueden implementar aplicaciones complejas con suficiente calidad usando el paradigma GraphQL?

3.5.3 Definición de los objetivos de la solución

El objetivo de la solución es contestar la pregunta de investigación **PI**$_{1.1}$, mediante la creación y validación de un artefacto tecnológico; que en este caso consiste en rea-

[2] https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf
[3] https://cit-conferences.org

lizar una prueba de concepto de la implementación del paradigma GraphQL. Como tema de implementación, se consideró la automatización de la gestión de estudios de mapeos sistemáticos de la literatura (SMS).

Población y muestra: Debido al contexto de la investigación, el estudio se aplicó en una *Población* conformada por veintidós profesores y veinte y cinco estudiantes de octavo nivel del Departamento de Software de la Universidad Técnica del Norte, que tenían los conocimientos adecuados para validar este trabajo. De los cuales, mediante la aplicación de la técnica de muestreo no probabilístico por conveniencia, la *muestra* se conformó por *cuarenta* sujetos (15 profesores y 25 estudiantes).

Encuesta de aceptación: Para validar la viabilidad del tema de implementación se realizó una encuesta de aceptación. La encuesta fue dirigida a la población de profesores y se conformó de 6 preguntas validadas por tres investigadores expertos en Ingeniería de Software antes de su ejecución. Los resultados indicaron que el 45,45 % de los encuestados conocían acerca de métodos de revisión de la literatura (SLR[4]/SMS), pero nadie conocía una herramienta para llevarlos a cabo; además, el *100 %* indicó que les gustaría recibir una breve formación para utilizar una herramienta que gestione SMSs, lo que confirmó la motivación para realizar la automatización de la gestión de SMSs como prueba de concepto de la implementación de GraphQL.

Propuestas existentes: Una vez establecido el tema de implementación, se realizó una búsqueda bibliográfica de herramientas que automaticen métodos de estudios sistemáticos de la literatura. La búsqueda se realizó en las bases de datos científicas Scopus, DBLP, y Springer, en donde, luego de un análisis de resultados se encontró los estudios de Knutas *et al.* [92] y Kohl et al. [93] que exponen herramientas para gestionar estudios de revisión sistemática de la literatura (SLR) y SMS.

Por un lado, Knutas [92] propone una aplicación llamada *NAILS*, que se conecta a la base de datos científica "web of science" para elegir una colección de datos de estudios y exportar en archivos al servidor de análisis basado en la web *HAMMER* para mostrar un análisis estadístico de los datos introducidos. También ofrecen la opción de descargar una aplicación basada en el lenguaje R para el estudio de la colección

[4]Revisión sistemática de la literatura (SLR, por sus siglas en inglés de Systematic Literature Review), sirve para identificar, analizar e interpretar evidencia reportada relacionada a un conjunto de preguntas de investigación específicas [88].

de datos. Por otro lado, Kohl [93] realiza un revisión de herramientas en línea que apoyan la realización y el informe de revisiones y mapeos sistemáticos; muestra 22 aplicaciones, de las cuales dos aplicaciones *Colandr* y *CADIMA* de acceso libre que apoyan la automatización de SMSs.

Identificación de principales características y funcionalidades: Se empezó revisando las funcionalidades de las herramientas existentes, en donde, con *Colandr* se pudo realizar las fases básicas para llevar a cabo SMSs, además, se distinguió las siguientes funcionalidades especiales: Planificación de trabajos, Citation Screening, Full-text Screening, Extracción de Datos, e importación de colecciones de datos a través de archivos planos. Además, se revisó las funcionalidades de la herramienta *CADIMA*, que presentaba documentación detallada en cada paso del proceso; sin embargo, fue difícil de utilizar, lo que sugería tener conocimientos previos de la aplicación para usarla correctamente.

Como resultado de la revisión de las herramientas *NAILS*, *Colandr*, y *CADIMA*, se logró identificar varias características y funcionalidades principales que se debería implementar en la *prueba de concepto*, estas funcionalidades son: Automatización de las fases básicas de los estudios de mapeo sistemático; Importación de estudios primarios; Colaboración con herramientas de gestión bibliográfica; Exportar resultados de los estudios realizados; Documentación integrada en la herramienta de todo el proceso del SMS; y que sea de fácil acceso y uso.

3.5.4 Diseño y desarrollo

Diseño

El artefacto se diseñó de acuerdo con el backlog de requerimientos establecidos para la automatización de la gestión de estudios de mapeos sistemáticos. Los requerimientos se conformaron por 15 historias de usuario, basadas en el proceso (ver Figura §5.1) propuesto en la guía de Petersen *et al.* [122] y las características y funcionalidades identificadas en la sección anterior §3.5.3. Además, se tomó como base la arquitectura orientada a microservicios para diseñar una aplicación cliente-servidor; utilizando como proveedor una API-GraphQL, y como cliente, una aplicación web integrada con la API-Mendeley para la gestión de referencias bibliográficas. La arquitectura de software de la aplicación se muestra en la Figura §3.9.

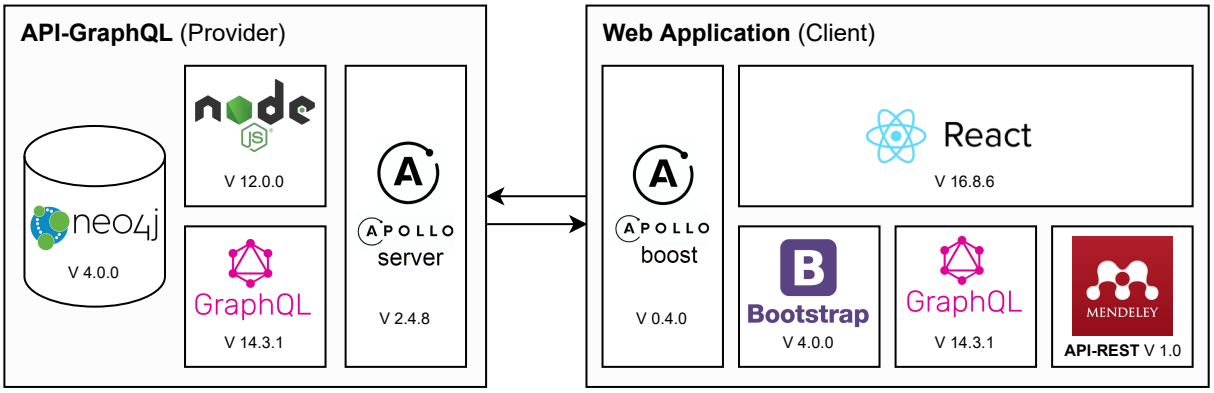


Figura 3.9: Arquitectura de software del artefacto

Desarrollo

El desarrollo de la aplicación se realizó desde el 03/06/2019 hasta el 06/12/2019 (220 horas) utilizando el marco de trabajo SCRUM. La aplicación denominada *"SMS-Online"* se desarrolló conforme a los requerimientos establecidos en el diseño. El equipo de trabajo estuvo formado por dos stakeholders, un product owner, un scrum master y un desarrollador. A continuación se muestra un resumen de las fases y artefactos utilizados en el desarrollo del artefacto de software, véase Tabla §3.2.

Tabla 3.2: Resumen del desarrollo e implementación del artefacto

Fase Scrum	Sprint (duración)	Entregables
Prejuego	Sprint 0 (20 horas)	Requerimientos: - 1 Backlog del producto - 15 historias de usuario Diseño: - Diagrama de procesos (Figura §5.2.2) - Arquitectura de software - Diagrama inicial de la base de datos
Juego	Sprint 1 (42 horas) Sprint 2 (42 horas) Sprint 3 (42 horas) Sprint 4 (42 horas)	- Spring Backlog - Incremento del producto de software
Post-Juego	Sprint 5 (32 horas)	- 15 Pruebas de Aceptación - Despliegue del producto de software - Acta de entrega-recepción

En esta fase, se destaca el uso del paquete *neo4j-graphql.js*[5] que facilitó la manipulación de datos utilizando en conjunto las tecnologías GraphQL y Neo4j [111]. En concreto, el paquete *neo4j-graphql.js* traduce las consultas GraphQL a un único Cypher[6], eliminando la necesidad de escribir resolutores de consultas GraphQL o de realizar consultas por lotes [56]. Además, se verificó los componentes GraphQL que se implementaron en el desarrollo de la aplicación *SMS-Online*. La Tabla §3.3 muestra los componentes implementados manualmente y generados por el paquete *neo4j-graphql.js*.

Tabla 3.3: Componentes GraphQL implementados

Componente clasificación	Componente	Implementado	Proveedor		Cliente
			Generado	Desarrollado	componentes
Executable Definition	Query	✓	12	0	27
	Mutation	✓	62	1	40
	Subscription	X	0	0	0
Type System Definition	Schema	✓	0	1	0
	Scalars	✓	0	132	132
	Objects	✓	0	12	12
	Interfaces	✓	0	1	1
	Unions	✓	0	1	1
	Enums	✓	0	2	2
	Input Object	✓	63	0	0
	Directives	✓	0	14	0

3.5.5 Demostración

El artefacto (Aplicación Web) desarrollado se desplegó como un servicio en la nube para constatar su aplicabilidad. El artefacto se nombró como *SMS-Online* y esta disponible en: https://appsms.utn.edu.ec:3000

3.5.6 Evaluación

En esta sección, se evaluó el uso del artefacto desarrollado en la sección anterior. Para lo cual, se evaluó la calidad en uso de la aplicación *SMS-Online*, utilizando los estándares ISO/IEC 25010, 25022, y 25040, en donde, se realizó tres pasos: 1) definir un modelo de calidad; 2) medir la calidad en uso de la aplicación *SMS-Online* de acuerdo

[5] https://www.npmjs.com/package/neo4j-graphql-js
[6] Cypher es el lenguaje de consulta de grafos de Neo4j.

a las métricas de las características establecidas en el modelo de calidad; y 3) evaluar los resultados obtenidos en el paso anterior.

Definición del modelo de calidad de uso

El modelo de calidad en uso fue definido por el Product Owner y el Scrum Master del proyecto, en donde, escogieron y valoraron (porcentaje del total) las características y sub características de calidad establecidas en la ISO/IEC 25010 [75], de acuerdo a las necesidades del contexto del estudio, ver Tabla §3.4.

Tabla 3.4: Modelo de calidad en uso

Características	Subcaracterísticas	Porcentaje subcaracterísticas	Porcentaje características
Efectividad	Tareas completadas	16%	42%
	Objetivos alcanzados	16%	
	Errores en una tarea	16%	
Eficiencia	Tiempo de la tarea	16%	32%
	Eficiencia del tiempo	16%	
Satisfacción	Utilidad	8%	26%
	Confianza	8%	
	Experiencia de usuario	8%	

Medición del modelo de calidad

En esta sección, se recaba la evidencia empírica del uso del artefacto. En primer lugar, la Tabla §3.5 se describe las funciones y elementos de medición que establece la norma ISO/IEC 25022 [76] asociadas a las sub características del modelo de calidad definido en la tabla §3.4.

Luego, para recabar las evidencias empíricas se utilizó dos instrumentos orientados a obtener los elementos de medida establecidos en la Tabla §3.5. Las cuales, se aplicaron a la muestra del estudio (40 sujetos) definida en el apartado §3.5.2.

- *Taller práctico* del uso de la aplicación *SMS-Online*, para obtener los elementos necesarios para medir las características de calidad en uso de eficacia y eficiencia.
- *Encuesta de satisfacción* del uso de la aplicación *SMS-Online*, luego de realizar el taller práctico. La encuesta se basó en la escala de usabilidad (SUS, por sus siglas en inglés de System Usability Scale), que proporciona una herramienta fiable y rápida para medir la usabilidad de una aplicación [12].

Tabla 3.5: Métricas de la calidad en uso

Característica	Sub característica	Funciones de medida	Elementos de medida
Efectividad	Tareas completadas	X=A/B	A=Número de tareas únicas completadas. B=Número total de tareas únicas intentadas.
	Objetivos alcanzados	$X = 1 - \sum Ai \mid X \geq 0$	Ai = Valor proporcional de cada objetivo faltante o incorrecto en la salida de la tarea (valor máximo = 1).
	Errores en una tarea	X = A	A = Número de errores cometidos por el usuario durante una tarea.
Eficiencia	Tiempo de la tarea	X = T	T = Tiempo de la tarea.
	Eficiencia del tiempo	X = A/T	A = Número de objetivos alcanzados. T = Tiempo.
Satisfacción	Utilidad	X = N/T	N = Número de usuarios satisfechos. T = Número de usuarios encuestados.
	Confianza	C = A / T, X=1-C	X = % Quejas. C = % Confianza. A = Número de quejas presentadas. T = Número total de encuestados.
	Experiencia de usuario	X=A	A = Valor de la escala psicométrica de un cuestionario sobre el interés (Escala de Likert)

Luego de aplicar los instrumentos de medición, se realizó una evaluación estadística de estos instrumentos para verificar su fiabilidad, y fundamentar la validez del estudio.

Evaluación estadística de los instrumentos de medición: Para la evaluación estadística de los instrumentos, se tomó en cuenta el tipo de variables y resultados obtenidos luego de medición aplicada a los sujetos de estudio. El análisis se realizó utilizando la herramienta RStudio[7].

La evaluación estadística del *Taller práctico*, se la realizó mediante el método estadístico del coeficiente Alfa de Cronbach; la estructura de la matriz de análisis contiene los siguientes elementos de medición: tareas intentadas, tareas completadas, objetivos completados, errores en una tarea, tiempo de las tareas, objetivos alcanzados, número de quejas y confianza. En donde, el resultado obtenido es de *0,89*, lo que indica

[7]RStudio es un entorno de desarrollo integrado (IDE por su siglas en inglés de Integrated Development Environment) para R, un lenguaje de programación para el cálculo estadístico y gráficos [107].

que la fiabilidad del taller es aceptable según [147] [81], ver Figura §3.10.

```
Reliability analysis
Call: psych::alpha(x = setValidacion)

 raw_alpha std.alpha G6(smc) average_r S/N   ase mean   sd median_r
      0.89      0.89    0.81      0.81 8.5 0.033 0.45 0.16     0.81
```

Figura 3.10: Evaluación estadística del taller práctico (Alfa de Cronbach)

La evaluación estadística de la *Encuesta de Satisfacción*, se la realizó mediante la técnica de Análisis Factorial Confirmatorio (AFC); la estructura factorial contiene los factores: utilidad (ML1) y comodidad (ML2), ver Figura §3.11

```
       ML1   ML2
P1    0.73 -0.08
P2   -0.05  0.46
P3    0.77 -0.07
P4    0.02  0.70
P5    0.74  0.02
P6    0.11  0.52
P7    0.87  0.07
P8    0.39  0.47
P9    0.73 -0.03
P10 -0.11  0.75
```

Figura 3.11: Evaluación estadística de la encuesta de aceptación (cálculo AFC)

En base a los resultados de la Figura §3.11, se analiza la correlación entre el diseño inicial de la encuesta y el cálculo del AFC de los resultados de la ejecución de la encuesta. El diseño inicial de ML1 contiene las preguntas Q1, Q6 y Q9, de las cuales se descartó la pregunta Q6 por tener un valor de correlación menor a 0.3, que significa que es un valor muy bajo para mantener una encuesta viable. El diseño inicial de ML2 contiene las preguntas: P2, P3, P4, P5, P7, P8 y P9, de las cuales se descartaron las preguntas: P3, P5, P7, P9 por tener una baja correlación. Después de analizar y elegir las preguntas correlacionadas, se calculó el índice de ajuste comparativo (CFI) y el índice de Tucker y Lewis (TLI) con el mismo algoritmo AFC, en donde, se obtuvo un CFI=0.94 y un TLI=0.90; estos valores se consideran aceptables [164], y confirman que la *encuesta es válida y fiable*.

Evaluación del modelo de calidad

La evaluación del modelo de calidad en uso, se obtuvo en base a los valores esperados establecidos en el modelo de calidad y los resultados obtenidos en la medición del modelo, ver Tabla §3.6.

Tabla 3.6: Evaluación del modelo de calidad de uso

Características	Sub características	Medición	Esperado	Logrado	Completado
Efectividad	Tareas completadas	0.97	16%	15.50%	96.62%
	Objetivos alcanzados	0.97	16%	15.50%	
	Errores en una tarea	0.96	16%	9.58%	
Eficiencia	Tiempo de la tarea	0.78	16%	12.51%	78.90%
	Eficiencia del tiempo	0.80	16%	12.74%	
Satisfacción	Utilidad	0.71	8%	7.05%	70.26%
	Confianza	0.70	8%	5.60%	
	Experiencia de usuario	0.70	8%	5.62%	
			Total	84.11%	

El resultado obtenido de la evaluación del modelo de calidad en uso es ***84,11%***, que se considera una ***oportunidad*** para mejorar los resultados económicos, sanitarios o medioambientales, según los riesgos y oportunidades asociados al nivel de calidad [73, 76].

3.6 Conclusiones

Como conclusiones de este capítulo, se puede decir que mediante la extracción conceptual de la especificación formal de GraphQL, se logró ilustrar y ejemplificar sus componentes, los cuales, se introducen a la comunidad académica y científica mediante el marco conceptual denominado *Paradigma GraphQL*.

El *Paradigma GraphQL* se puedo validar utilizando el enfoque de investigación *DSR*, en donde, surgió la sub pregunta de investigación $\mathbf{PI}_{1.1}$: ¿Se pueden implementar aplicaciones complejas con suficiente calidad usando el paradigma GraphQL?. Esta pregunta se responde mediante la creación y validación de un artefacto tecnológico. La creación del artefacto consistió en implementar GraphQL en una aplicación web cliente-servidor basada en la arquitectura de microservicios. La validación del artefacto consistió en la evaluación de la calidad en uso de la aplicación desarrollada.

Como resultado del desarrollo se logró implementar todos los componentes de GraphQL (excepto la operación *"Subscription"*); y como resultado de la validación se logró evaluar la calidad en uso del artefacto obteniendo el 96.62% de Efectividad, 78.89% de Eficiencia, y 70.26% de Satisfacción, que dan como resultado general el 84.11% del valor de la calidad esperada; lo que en el contexto de la evaluación sig-

nifica que el artefacto se considera como una oportunidad para mejorar los resultados económicos, sanitarios o medioambientales, según los riesgos y oportunidades asociados al nivel de calidad. En este sentido, se dio respuesta a la sub pregunta $\mathbf{PI}_{1.1}$ y se concluye que las implementaciones de GraphQL si funcionan, y en consecuencia, el *paradigma GraphQL* queda validado.

Como trabajo futuro de este capítulo se propone validar la implementación de la operación *"Subscription"*, debido a que este componente GraphQL quedó fuera de este estudio. Además, se sugiere realizar más estudios que muestren evidencia empírica acerca de la implementación de GraphQL.

Acuerdos de nivel de servicio

Si decides hacer solo las cosas que sabes que van a funcionar,
dejarás un montón de oportunidades encima de la mesa.

Jeff Bezos (1964 –),
Presidente Ejecutivo de Amazon

En este capítulo, se introduce la base conceptual de los Acuerdos de nivel de servicio, *en donde, se revisará sus principales conceptos; se analizará las principales propuestas existentes para gestionar SLA para APIs; y se describirá la propuesta más adecuada, que servirá como base teórica para desarrollar la propuesta expuesta en el capítulo §8.*

En la sección §4.1, se destaca una breve discusión de los temas a tratar en este capítulo. En la sección §4.2, se presenta una comparativa de propuestas existentes para gestionar los acuerdos de nivel de servicios (SLA) para APIs. En la sección §4.3, se describe la propuesta más adecuada de acuerdo a la comparativa realizada en la sección §4.2.

4.1 Introducción

Los Acuerdos de Nivel de Servicio (SLA, por sus siglas en inglés de Service-Level Agreements) se definen como la relación entre un proveedor de servicios y sus clientes. El acuerdo describe los productos y/o servicios que el cliente ha de recibir, las responsabilidades de cada parte, las condiciones económicas, cómo será medido el servicio y el esquema de reporte. Su objetivo es presentar de manera clara, concisa y medible lo que el proveedor hace para el cliente [172].

A medida que el diseño de la arquitectura del software evoluciona hacia un paradigma de microservicios, las APIs se han convertido en un punto de inicio para construir aplicaciones, en donde, los proveedores de APIs ofrecen diferentes niveles de servicio con limitaciones a medida, normalmente basadas en el coste [46]. En este sentido, el objetivo de este capítulo es realizar una comparativa entre varias propuestas existentes que gestionen acuerdos de nivel de servicios para APIs, para determinar la propuesta más adecuada, la cual, se describirá con más detalle y será utilizada como base teórica para el desarrollo del capítulo §8 que pretende gestionar los acuerdos de nivel de servicio (SLA) para APIs GraphQL.

4.2 Propuestas existentes de SLA para APIs

En esta sección, se realiza un análisis comparativo entre seis propuestas destacadas en los campos de la academia y la industria que pretenden gestionar las limitaciones y alcance de servicios web y APIs mediante la especificación de limitaciones de los servicios web y gestión de acuerdos de nivel de servicios (SLA). Las principales características identificadas para la comparación son: C1: Dominio objetivo son los servicios web; C2: Modelar diferentes operaciones de un servicio; C3: Modelamiento de modelos jerárquicos; C4: Modelamiento de limitaciones temporales; y C5: Herramienta de soporte para modelar SLA. La Tabla §4.1 muestra la comparativa entre las principales propuestas existentes para gestionar SLA en APIs.

Por otro lado, basados en los estudios empíricos [43, 112] que analizaron un conjunto de 69 y 500 APIs industriales respectivamente, muestran entre sus principales conclusiones que: i) la mayoría de las APIs proporcionan diferentes capacidades y precios dependiendo del nivel o plan de consumo. ii) Las limitaciones de uso es un aspecto común en las ofertas que describen las APIs. iii) Las limitaciones más comunes

Tabla 4.1: Comparativa entre propuestas para gestionar SLA

Propuesta	C1	C2	C3	C4	C5	C6
ysla [37]	✓	✓	✓	✓	✓	✓
SLA4OAI [44]	✓	✓	✓	✓	✓	✓
L-USDL Ag. [47]	✓	✓				✓
CSLA [94]		✓			✓	
rSLA [161]	✓		✓	✓		
SLAC [169]						✓

son cuotas sobre periodos de tiempo estáticos (e.g. diariamente), o tiempos dinámicos (e.g. por segundo) en las solicitudes de la API. iv) Las ofertas pueden incluir un amplio número de métricas sobre otros aspectos de la API que pueden ser dependientes o independientes del dominio, como por ejemplo el número de resultados devueltos o el consumo de CPU durante el procesamiento de la solicitud.

A partir de las conclusiones de los estudios empíricos [43, 112] y del resultado de la comparativa de las principales propuestas existentes para gestionar SLA en APIs mostrada en la tabla §4.1.Se ha identificado que el marco conceptual de la propuesta *SLA4OAI* expuesta en los trabajos [44, 45, 46, 71] es la más adecuada para fundamentar teóricamente la propuesta para describir SLA para APIs GraphQL que se desarrollará en el capítulo §8; debido a que *SLA4OAI* brinda un soporte no funcional del ciclo de vida del desarrollo de la API y que tiene un alto nivel de expresividad de las ofertas de la API. Por lo cual, en la siguiente sección se describirá a detalle la propuesta *SLA4OAI*.

4.3 SLA4OAI: Estándar para describir SLA para APIs REST

SLA4OAI es un estándar de código abierto propuesto por Gamez-Diaz *et al.* [44, 45, 46, 71]; se basa en los estándares de (OAI, por sus siglas en inglés de OpenAPI-Specification) para describir SLA para APIs REST de una manera tecnológica neutral.

4.3.1 Ciclo de vida del desarrollo de APIs impulsado por SLA

En esta sección basándose en Gamez-Diaz *et al.* [44, 46], primero se identifica un conjunto mínimo de etapas y actividades generales para abordar el ciclo de vida de

APIs, a la cual luego se le incorpora las actividades necesarias para gestionar acuerdos de nivel de servicio.

Por un lado, se identifica que las actividades el ciclo de vida básico de APIs comienzan por el desarrollo y prueba de la lógica de las funcionalidades que proveerá el servicio del API. Luego, se realiza la actividad de despliegue en la que el artefacto desarrollado se configura para ser ejecutado en una infraestructura determinada. Por último, una vez que la API está en funcionamiento, se inicia la operación en la que se pueden aceptar las solicitudes de los clientes. Este proceso es una simplificación que puede evolucionar para añadir pasos intermedios como pruebas o para incluir un modelo de ciclos evolutivos en el que se desplieguen progresivamente diferentes versiones.

Por otro lado, para incorporar los acuerdos de nivel de servicio al proceso básico del ciclo de vida, tanto el proveedor como el cliente de la API deben interactúan como se detalla en la Figura §4.1.

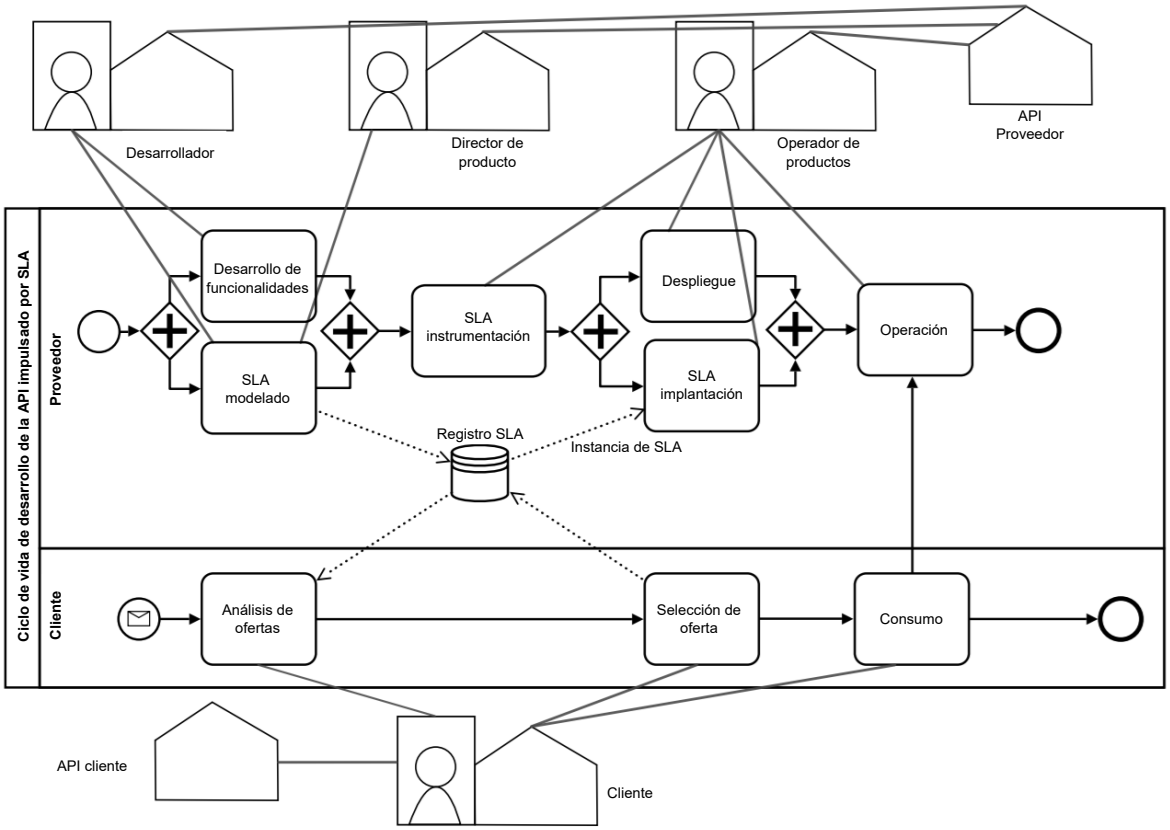


Figura 4.1: Ciclo de vida del desarrollo de APIs impulsado por SLA

En la figura §4.1 se muestra que desde la perspectiva del proveedor, las actividades comienzan por el desarrollo funcional de las APIs, las cuales pueden implementarse en paralelo con un modelado de los acuerdos de nivel de servicio (SLA), en donde, se describe en documentos los planes de la oferta real de la API en términos de cuotas y tarifas, que luego se registran para que puedan ser accedidos por los clientes. Una vez que el desarrollo funcional como el modelado de los planes SLA hayan concluido, se debe llevar a cabo la instrumentación del SLA, en donde, las herramientas y/o artefactos desarrollados se deben parametrizar para que ajusten su comportamiento en función a los planes SLA establecidos; estas a su vez deben proporcionar métricas adecuadas para poder analizar su estado en función al SLA establecido. Luego, mientras se realiza el despliegue de la API, se desarrolla una actividad paralela que implementa el SLA, el cual se debe configurar la infraestructura de despliegue para hacer cumplir el SLA antes de que la API llegue a la actividad de operación. Complementariamente, desde la perspectiva del cliente, una vez que el proveedor ha publicado la oferta de SLA (es decir, los planes) en el Registro de SLA inicia el análisis de la oferta para seleccionar la opción más adecuada y para crear y registrar su SLA real; finalmente, el consumo de la API se lleva a cabo mientras la actividad de operación y su regulación se basa en los términos (como cuotas o tarifas) definidos en el SLA.

4.3.2 Roles en el ciclo de vida del desarrollo de APIs impulsado por SLA

Los roles inmersos en el ciclo de vida de APIs impulsadas por SLA son [46]:

Desarrollador: Este rol está compuesto por el equipo responsable del desarrollo de una determinada API y de ponerla a disposición de otros equipos. Sus casos de uso están relacionados con la definición de objetivos de nivel de servicio (SLO, por sus siglas en inglés de Service Level Objectives).

Director de producto: Este rol está compuesto por personas de negocios, alineadas con los objetivos de la empresa. Sus casos de uso tienen como objetivo satisfacer las necesidades del cliente y conocer el panorama general de las dependencias entre los servicios. En concreto, conocer los SLO de las dependencias descendentes para poder crear productos que satisfagan las necesidades de los clientes.

Operador de productos: Esta función está compuesta por personas de administración de sistemas, que se encargan de supervisar e informar el rendimiento del servicio

en los SLO. Sus casos de uso tienen como objetivo notificar cualquier alerta o incidente y tomar medidas correctivas.

Cliente: Este rol está compuesto por el conjunto de clientes de la API. Sus casos de uso pretenden ser informados de los diferentes niveles de servicio y reclamar si los niveles de servicio no se están cumpliendo.

4.3.3 Componentes de los acuerdos de nivel de servicio para APIs

Los conceptos que se describen a continuación, serán la base de los componentes del documento SLA que se revisará en la sección §4.3.4:

Componentes generales del SLA

El contexto (Context): describe aspectos como la versión, las partes interesadas o el periodo de validez.

Las métricas (Metrics): son elementos que se recogen y computan.

El indicador de nivel de servicio (SLI, Service Level Indicator): es un caso particular de métrica que se utiliza para evaluar un aspecto clave del sistema.

El objetivo de nivel de servicio (SLO, Service Level Objective): es un objetivo numérico preciso (a menudo un ratio) para uno o más SLI, que describe la fiabilidad o el rendimiento mínimo aceptable de un sistema.

Los términos de garantía (Guarantee terms): describen los compromisos sobre determinados SLI.

El Acuerdo de Nivel de Servicio (SLA, Service Level Agreement): es, por tanto, un contrato firmado con un usuario. En particular, los SLI y los SLO son construcciones técnicas mientras que los SLA son construcciones empresariales.

Las propiedades del servicio (Service properties): (o configuración) son las restricciones de atributos que se utilizan para dirigir el comportamiento de la API.

Restricciones de la API

Las cuotas (Quotas): describen las limitaciones de uso durante un periodo de tiempo determinado/estático de tiempo.

Las tarifas (Rates): describen las limitaciones de uso durante un periodo de tiempo dinámico.

Limitación de tiempo (Time constraint): Algunas APIs pueden ofrecer una serie de limitaciones en cuanto al tiempo en el que se solicita.

La autenticación (Authentication): es la verificación de las credenciales de la solicitud.

Monetización de la API

La fijación de precios (Pricing): es la forma de monetizar las API.

Los planes (Plans): son un enfoque para una amplia gama de necesidades empresariales mediante organizar los precios en un conjunto de niveles de planes.

La medición (Metering): es el registro del uso de la API con suficiente detalle para realizar la tarificación.

La tarificación (Rating): es la conversión de los registros del uso de la API en una cantidad de dinero.

La facturación (Billing): es la presentación a un usuario de la API de un informe de los importes adeudados, teniendo en cuenta cualquier descuento, crédito de servicio impuestos y reparto de ingresos.

El cobro (Collection): es la forma de recibir y registrar los pagos de cantidades adeudadas por los usuarios de las APIs.

Cumplimiento de la ley (Enforcement): consiste en impedir que un usuario utilice una API luego de agotarse su crédito de servicio prepagado o ha alcanzado un límite de crédito.

4.3.4 Estructura del documento SLA para APIs REST

Esta sección muestra la especificación para describir los componentes de los documentos SLA, los cuales contendrán la configuración y los acuerdos de nivel de servicio de una API específica. La tabla §4.2 muestra el esquema del documento SLA usando los conceptos repasados en la sección §4.3.3.

Tabla 4.2: Esquema del documento SLA

Comienzo de tabla					
Componente	Requerido	Campo(s)	Requerido	Subcampo(s)	Requerido
context	✓	id	✓	-	-
		version	✓	-	-
		api	✓	-	-
		type	✓	-	-
		provider	•	-	-
		consumer	•	-	-
		validity	•	effectiveDate	✓
				expirationDate	•
infrastructure	✓	supervisor	✓	-	-
		monitor	✓	-	-
		name	•	-	-
pricing	•	cost	•	-	-
		custom	•	-	-
		currency	•	-	-
		billing	•	-	-
metrics	✓	{name}	•	type	✓
				format	•
				description	•
				unit	•
				resolution	•
plans	•	{planName}	•	configuration	•
				availability	•
				pricing	•
				quotas	•
				rates	•
				guarantees	•
quotas	•	{pathName}: {methodName}: {metricName}	•	max	•
				custom	•
				period	•
				scope	•
rates	•	{pathName}: {methodName}: {metricName}	•	max	•
				custom	•
				period	•
				scope	•
guarantees	•	{pathName}: {methodName}	•	objective	✓
				period	•
				window	•

Continuación de la Tabla §4.2					
Campo	Requerido	Campo(s)	Requerido	Subcampo(s)	Requerido
				scope	•
configuration	•	name	•	-	-
Fin de tabla					

Leyendas: ✓ Campo requerido; • Campo opcional; - No aplica

La documentación completa de la propuesta *SLA for Open API Initiative Specification (Research Version)*, se puede acceder en el siguiente enlace: `https://sla4oai.specs.governify.io/Specification.html`

Estudio de mapeo sistemático de GraphQL

El mayor riesgo es no correr ningún riesgo

Mark Zuckerberg (1984 –),
Director General de Facebook

En este capítulo, se responde las sub preguntas de la pregunta de investigación $\mathbf{PI}_1$ *acerca del estado del paradigma GraphQL. Asimismo, se identifican vacíos de investigación, tanto en temas generales como específicos del paradigma GraphQL.*

En la sección §5.1, se presenta una introducción de los temas que se tratarán en el capítulo. En la sección §5.2, se define las fases del proceso del estudio de mapeo sistemático y las preguntas de investigación que se responderán. En la sección §5.3, se contestan las preguntas de investigación referente al estudio de mapeo sistemático. La sección §5.4 presenta las conclusiones del estudio. Por último, la sección §5.5 resume el capítulo.

5.1 Introducción

Este capítulo tiene como objetivo completar la respuesta a la pregunta de investigación **PI**$_1$: ¿Cuál es el estado actual de GraphQL?, por lo cual, complementando a la pregunta **PI**$_{1.1}$ planteada y respondida en la sección §3.5, se plantea las siguientes sub preguntas:

- **PI**$_{1.2}$: ¿Quiénes son los autores e instituciones que han investigado acerca de GraphQL? (Sección §5.3.1).
- **PI**$_{1.3}$: ¿Dónde se han publicado los trabajos de investigación acerca de GraphQL? (Sección §5.3.2).
- **PI**$_{1.4}$: ¿Qué tipos y métodos de investigación se han utilizado en las publicaciones acerca de GraphQL? (Sección §5.3.3).
- **PI**$_{1.5}$: ¿Cuándo se han publicado los trabajos de investigación acerca de GraphQL? (Sección §5.3.4).
- **PI**$_{1.6}$: ¿Por qué se ha estudiado el paradigma GraphQL? (Sección §5.3.5).
- **PI**$_{1.7}$: ¿Cómo se ha introducido GraphQL en la comunidad científica? (Sección §5.3.6).

En este sentido, se responde a las preguntas planteadas mediante el desarrollo del *Estudio de Mapeo Sistemático*, para tener una visión del campo general y campos específicos de los estudios que se han realizado acerca de GraphQL en la comunidad científica.

5.2 GraphQL: Estudio de mapeo sistemático

En este apartado, se define las fases del proceso para llevar a cabo un estudio de mapeo sistemático (SMS, por sus siglas en inglés de *Systematic Mapping Study*) que sirve para establecer una visión general y específica acerca de GraphQL mediante la clasificación de estudios primarios que sirve para estructurar y descubrir el campo de interés de la comunidad científica en el tema.

5.2.1 Definición del proceso del SMS

El proceso para la realización del SMS se estableció en base a las directrices propuestas por Petersen *et al.* en 2008 [120], Petersen *et al.* 2015 [122], y Kuhrmann 2017 [96], en la figura §5.1 se muestra las fases básicas de los SMS.

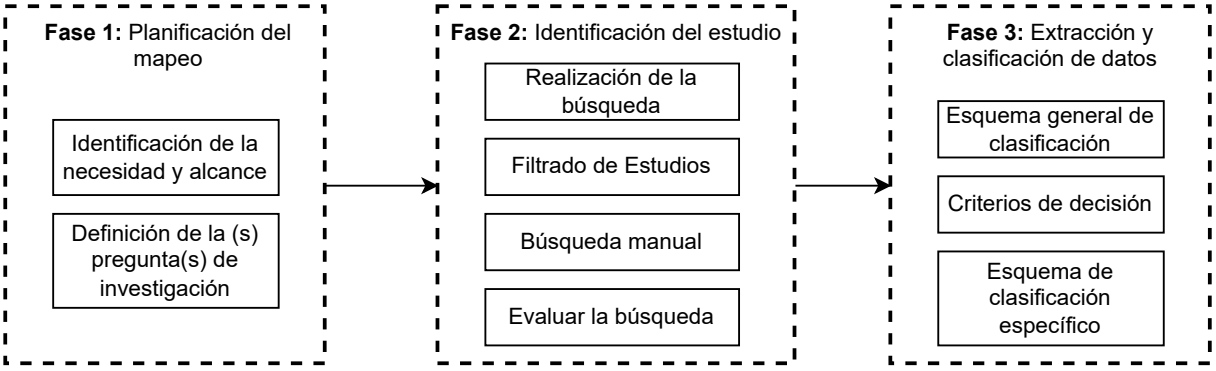


Figura 5.1: Definición del proceso SMS

5.2.2 Fase 1: planificación del mapeo

En esta sección se establece las actividades específicas del proceso del mapeo, las cuales servirán como guía durante el SMS o para que este sea reproducido:

Identificación de la necesidad y el alcance: El estudio de mapeo sistemático tiene como objetivo proporcionar una visión general mediante un inventario de estudios categorizados y clasificados acerca del paradigma GraphQL. El SMS sirve para esclarecer lagunas de investigación que existen en torno a este tema, las cuales servirán como una guía para inicio de futuras investigaciones. La clasificación se realiza con dos enfoques, la primera una clasificación específica del tema de estudio, y la otra una clasificación general e independiente del tema de estudio que muestre los tipos de publicaciones que ha realizado la comunidad científica, la madurez de la evidencia empírica, su tendencia, y evolución a lo largo del tiempo [3, 120, 122].

5.2.3 Fase 2: Identificación de estudios

Esta sección detalla cómo se identificó el conjunto de estudios primarios utilizados para el mapeo, ver figura §5.2.

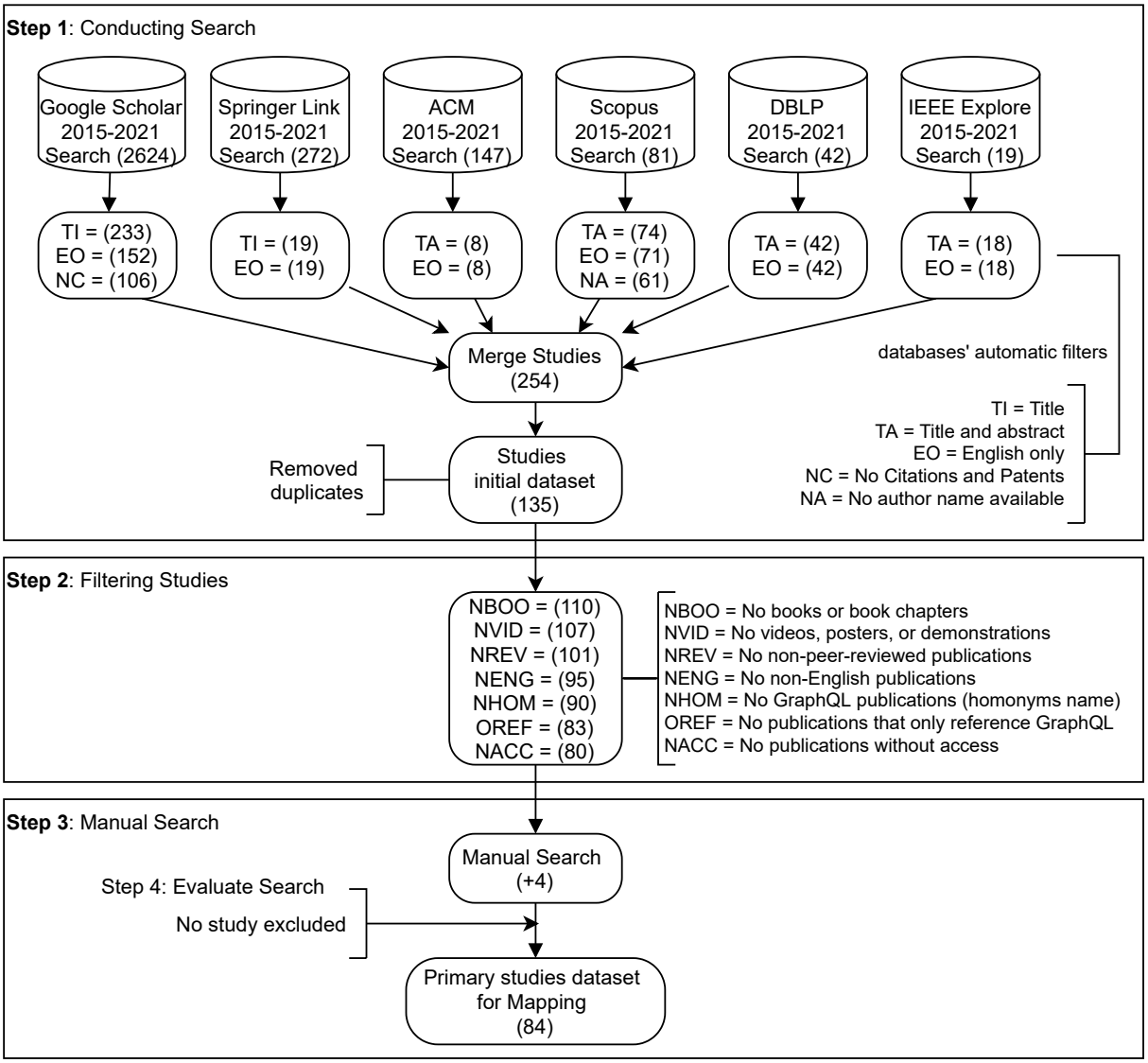


Figura 5.2: Identificación de estudios

Estrategias de búsqueda: En este punto se eligió dos estrategias: la primera una búsqueda automática en las bases de datos científicas, la que se complementa con una segunda estrategia que consiste en una búsqueda manual para añadir estudios relevantes que no aparecieron en la primera búsqueda. Los pasos establecidos para la búsqueda de estudios son: 1) Realizar la búsqueda de estudios primarios mediante una cadena de búsqueda en las bases de datos científicas; luego se aplicó filtros automáticos de las bases de datos de acuerdo con los criterios de inclusión y exclusión definidos para el estudio. Después, se utilizó las herramientas de gestión de referencias bibliográficas

Zotero[1] y Mendeley[2], para gestionar y fusionar las referencias de los estudios en una estructura homogénea. Se finalizó este paso, eliminando los estudios duplicados. 2) Se filtró de manera manual los estudios aplicando criterios de inclusión y exclusión. 3) Se pretendió mejorar la calidad del conjunto de estudios primarios, aplicando las recomendaciones de Wohlin [184], mediante una búsqueda manual para añadir estudios relevantes al conjunto de datos. 4) Por último, se realizó una evaluación a la búsqueda de estudios.

Conducción de la búsqueda: La búsqueda se empezó con establecer la cadena de búsqueda usando el enfoque de mejora interactiva a partir de la identificación de palabras claves de artículos conocidos, en la donde, se decidió definir a la cadena de búsqueda con la palabra *"graphql"* para cubrir todo lo reportado en la comunidad científica desde 2015, año de liberación de GraphQL a la comunidad, hasta el 30 de junio de 2021. Se identificó que una población científica apropiada para consultar estudios primarios son las siguientes bases de datos de literatura científica comúnmente utilizadas en la ingeniería de software: ACM Digital Library[3], SpringerLink[4], IEEE Xplore Library[5], Scopus[6], Google Scholar[7], and DBLP[8] [32, 88, 96, 122]. Se empezó realizando una búsqueda automática en las bases de datos y se obtuvo un total de *3185* estudios. Luego, se realizó una segunda búsqueda, pero esta vez aplicando filtros automáticos disponibles en las herramientas de búsqueda de las bases de datos; los filtros aplicados están basado en los criterios de inclusión y exclusión definidos para el estudio, en donde, se obtuvo *254* estudios. Posteriormente, se eliminó los estudios duplicados, obteniendo así un conjunto inicial de *135* estudios, ver figura §5.2.

Filtrado de estudios: En esta sección, se define y aplica los criterios de inclusión y exclusión sobre el conjunto inicial de estudio para establecer la relevancia del tema de investigación [98, 122].

Criterios de inclusión:

[1] https://www.zotero.org/
[2] https://www.mendeley.com/
[3] https://dl.acm.org
[4] https://link.springer.com
[5] https://ieeexplore.ieee.org
[6] https://www.scopus.com/
[7] https://scholar.google.com
[8] https://dblp.org

- Publicaciones en inglés que hayan sido revisadas por pares en revistas, conferencias o talleres.
- Literatura Gris (pruebas no controladas por editoriales comerciales) puede incluir trabajos académicos como tesis de licenciatura, maestría y doctorado [116].
- Publicaciones de 2015 a 2021.
- Publicaciones que se centran en el uso del paradigma GraphQL en el título y el resumen.
- Publicaciones que contengan un contexto, objetivos y método de investigación razonables.

Criterios de exclusión:

- No publicaciones que sólo hagan referencia a GraphQL como una cita externa y no utilizan o amplían el paradigma (sección §3.3) en sus estudios.
- No publicaciones que hagan referencia a GraphQL (nombre homónimo) propuesto por Huahai He y Ambuj Singh en 2008 [64], el cual, se trata de un lenguaje de consulta de grafos que permite la manipulación flexible de estructuras de grafos.
- No publicaciones, libros, capítulos de libros, vídeos, posters o demostraciones que no hayan sido revisados por pares.
- No publicaciones que no sean en inglés.
- No publicaciones sin acceso.

Tras aplicar los criterios de inclusión y exclusión, se obtuvo un conjunto de *ochenta* estudios.

Búsqueda manual: Se completó la estrategia de búsqueda con el método de búsqueda manual en las bases de datos antes mencionadas. Se recuperó *cuatro* estudios que no se identificaron en la primera búsqueda, lo que dio lugar a un conjunto de *ochenta y cuatro* estudios primarios.

Evaluación de la búsqueda: Petersen *et al.* [122] y Kitchenham [88] recomiendan mantener un bajo nivel de exigencia en la evaluación de la calidad de los estudios primarios porque los SMS tienen como objetivo proporcionar una visión general del área temática de la investigación. Por este motivo, se decidió no excluir ningún estudio primario en esta sección, lo que dio como resultado un conjunto de *ochenta y cuatro* estudios primarios para el mapeo (ver Anexo §A).

5.2.4 Fase 3: Extracción de datos y clasificación

En esta fase, se extrajo los datos necesarios de los estudios primarios para realizar una clasificación general que no depende del tema investigado (clasificación independiente del tema) y otra específica sobre el paradigma GraphQL (clasificación específica del tema) [122].

Esquema de clasificación independiente del tema: A continuación se describen los campos de extracción de datos basados en las *facetas*, *sedes*, *tipos de investigación*, *métodos de investigación*, y *enfoques de estudio* propuestos en Petersen *et al.* [122]:

- Autores: autor principal (primer autor) y coautores (el resto de los autores) de la publicación.
- Afiliación de los autores: primera afiliación de los autores de la publicación.
- Países: país de la afiliación de los autores.
- Año: año de la publicación.
- Sede: nombre de la sede de la publicación.
- Tipos de sede: tipos de sedes revisados por pares como revistas, conferencias y talleres [122]. Además, se añadió la literatura gris, como artículos académicos, tesis y disertaciones, para reducir un posible sesgo de publicación y proporcionar una visión más equilibrada de las pruebas recogidas para el estudio [116].
- Índice de sede: factor de impacto de los tipos de sede medido con *Scimago Journal & Country Rank* (SJR)[9], *Journal Citation Reports* (JCR)[10], y *GII-GRIN-SCIE* (GGS)[11].

[9] https://www.scimagojr.com
[10] https://jcr.clarivate.com
[11] http://gii-grin-scie-rating.scie.es

SJR calcula el factor de impacto Scimago Journal Rank (SJR) de las revistas y los países a partir de la información de la base de datos Scopus (Elsevier B.V.) [143], mide las citas ponderadas de una revista según el área científica y la relevancia de las revistas citadas [144]. Las subdisciplinas de las revistas se clasifican en cuatro cuartiles (Q1 a Q4) utilizando el Índice de Citación SJR [36]. El JCR es un producto de *Web of Science* que proporciona una rica gama de métricas de citación como el *Journal Impact Factor*™ (JIF) [23]. La clasificación por cuartiles de una revista (Q1 a Q4) se determina comparando una revista con otras de su categoría JCR basándose en el JIF [24]. GGS es una iniciativa que proporciona una clasificación unificada de las conferencias de ciencias de la computación [50], véase la tabla §5.1.

Tabla 5.1: Calificación de Conferencias GGS

Clase	Clasificación	Descripción
1	A++, A+	conferencias de primera clase
2	A, A-	eventos de muy alta calidad
3	B, B-	eventos de buena calidad
-	Trabajo en progreso	Trabajo en progreso

- Tipos de publicación: esta clasificación está relacionada con los tipos de sede revisados por pares las cuales son: artículos de revistas, artículos de conferencias y artículos de talleres [122]. Para la literatura gris, de acuerdo con la clasificación de tesis del Ministerio de Educación Finlandés[12] se utilizó: Tesis de licenciatura, tesis de máster, y tesis de doctorado.

- Tipo de investigación: se basa en la clasificación de tipos de investigación propuesta por Wieringa [180] y se la complementó con una tabla de decisión para desambiguar la clasificación de los estudios mediante la comprobación de una serie de condiciones propuestas por Petersen et al. [122], véase la tabla §5.2.

  A continuación, se muestra y ordena los tipos de investigación según la madurez expuesta por los estudios [180, 185]:

  - *Investigación de evaluación*, evalúa empíricamente la investigación de un problema o la aplicación de una técnica en la práctica de la ingeniería.
  - *Investigación de validación*, desarrolla una solución novedosa y se evalúa empíricamente en un entorno de laboratorio (es decir, no se utiliza en la práctica).

[12] https://aaltodoc2.org.aalto.fi/doc_public/ohjeet/publicationclassification2010.pdf

- *Propuesta de solución*, este tipo de estudio propone una solución a un problema de investigación, y los beneficios se discuten pero no se evalúan.
- *Documentos filosóficos*, son estudios que estructuran una área en forma de taxonomía o marco conceptual, por lo que proporciona una nueva forma de ver las cosas existentes.
- *Documentos de experiencia*, son estudios que incluyen la experiencia del autor sobre qué y cómo ocurrió algo en la práctica.
- *Documentos de opinión*, estos estudios contienen la opinión del autor sobre un tema concreto sin basarse en trabajos de investigación y metodologías relacionadas.

Tabla 5.2: Clasificación de los tipos de investigación

Tipo de investigación \ Condiciones	*Usado en la práctica*	*Solución novedosa*	*Evaluación empírica*	*Marco conceptual*	*Opinión sobre algo*	*Experiencia de los autores*
Investigación de evaluación	V	•	V	•	F	•
Investigación de validación	F	•	V	•	F	•
Propuesta de solución	•	V	F	•	F	•
Documentos filosóficos	F	F	F	V	F	F
Documentos de experiencia	V	F	F	•	F	V
Documentos de opinión	F	F	F	F	V	F

Leyenda si aplica: V=Verdadero, F=Falso, y •= irrelevante o no aplicable.

- Métodos de investigación: en este apartado basados en Petersen *et al.* [122] se identifica que los tipos de "investigación de evaluación" e "investigación de validación" necesitan una evaluación empírica mediante el uso de métodos de investigación frecuentemente utilizados en la ingeniería de software [33, 180, 186]; en este sentido, analizamos y utilizamos el siguiente subconjunto de métodos de investigación propuestos por [122]:

- *Encuesta*, utilizada para identificar las características de una población amplia de individuos, se asocia estrechamente con el uso de cuestionarios para la recogida de datos [33].
- *Estudio de caso*, utilizado para investigar una entidad o fenómeno en su contexto de la vida real dentro de un tiempo y espacio específico [186].
- *Experimento controlado*, utilizado para investigar una hipótesis comprobable en la que se manipulan una o más variables independientes para medir su efecto sobre una o más variables dependientes [33].
- *Simulación*, utilizado como una potente técnica para manejar la complejidad del modelo (cientos de variables dinámicas y vínculos causales). El uso de la simulación permite a los gestores evaluar de forma rápida y segura las implicaciones de una política prevista antes de aplicarla [105].
- *Prototipos*, es un enfoque basado en una visión evolutiva del desarrollo de software y un impacto en el proceso de desarrollo. Implica la producción de las primeras versiones de trabajo (prototipos) del sistema de aplicación de características y la experimentación con ellos [14].
- *Análisis matemático*, abarca las técnicas básicas para identificar un conjunto de reglas de razonamiento de forma precisa en el sistema estudiado [9].

- Ámbito de estudio: se refiere al contexto en el que se realizó el estudio, estos pueden ser: ámbito académico, industrial o gubernamental [26]. En este sentido, establecimos la siguiente semántica para definir los ámbitos mencionados:
  - *Académico*: estudios realizados en entornos sintéticos que no implementan soluciones de uso público.
  - *"Industria" y "Gobierno"*: estudios realizados en entornos de práctica de la industria/gobierno o que exponen soluciones de servicio público.

Esquema de clasificación específico del tema: A continuación, se describen los campos de extracción de datos para la clasificación específica del paradigma GraphQL:

- Áreas de conocimiento: se consideró las áreas de conocimiento establecidas en la Guía del Cuerpo de Conocimiento de la Ingeniería del Software (SWEBOK, por sus siglas en inglés de Software Engineering Body of Knowledge) versión 3.0 por la *IEEE Computer Society*[13] [9]. Las áreas son: Requisitos de software, Diseño

[13] https://www.computer.org

de software, Construcción de software, Pruebas de software, Mantenimiento de software, Gestión de la Configuración, Gestión de la Ingeniería, Proceso de Ingeniería, Modelos y Métodos de Ingeniería, Calidad, Práctica Profesional de la Ingeniería, Economía de la Ingeniería, Fundamentos de la Computación, Fundamentos Matemáticos y Fundamentos de la Ingeniería.

- Dominio de contribución: se refiere al dominio en donde se aplica la contribución técnica del estudio, de acuerdo a la estructura de los principales componentes del paradigma GraphQL establecido en la figura §3.1, en donde, se identifica los dominios de *Proveedor* y *Cliente* del servicio GraphQL. Se considera dominio Proveedor a cualquier contribución implementada en el *servicio GraphQL*, y el dominio Cliente son las contribuciones que consumen el *servicio GraphQL*.

- Tipo de contribución: es una clasificación de alto nivel de las contribuciones técnicas utilizando el paradigma GraphQL, para lo cual, se definió la siguiente semántica para identificar y clasificar los tipos de contribución:

  - *Implementación,* se refiere a los estudios que utilizan el paradigma GraphQL para implementar software.
  - *Análisis,* se refiere a los estudios que aportan un análisis comparativo o una revisión teórica para identificar formas de aplicar el paradigma GraphQL.
  - *Integración,* se refiere a los estudios que presentan la integración de GraphQL con otras tecnologías para diseñar o construir soluciones.
  - *Extensión,* se refiere a los estudios que proponen un nuevo componente o funcionalidad en el paradigma GraphQL.
  - *Aplicaciones,* se refiere a estudios que proponen enfoques de soluciones aplicando el paradigma GraphQL pero sin llegar a su implementación.

- Componentes de GraphQL: en base a las secciones §3.3 y §3.4, se estableció la siguiente clasificación:

  - *Componentes de Servicio*: Ejecución, Introspección, Resolutores, Sistema de Tipos, y Validación.
  - *Componentes de Sistema de Tipos*: Directivas, Enumeraciones, Objetos de Entrada, Interfaces, Objetos, Escalares, Esquemas, y Uniones.
  - *Componentes de Ejecución*: Fragmentos, Mutaciones, Consultas y Suscripciones.

- API pública: se refiere a las APIs públicas de la vida real utilizadas en el estudio como un caso de uso o un sistema de apoyo.

Por lo antes expuesto, en la tabla §5.3 se muestra la estructura de campos necesarios para la extracción de datos, que tiene como objeto responder a las preguntas de investigación establecidas para este estudio de mapeo sistemático. Los campos se derivan de la clasificación independiente y específica del tema de investigación, y de las estructuras de datos mínimas recomendadas por Kuhrmann *et al.* [96] y Petersen *et al.* [122].

Tabla 5.3: Estructura de extracción de datos

No.	Campo	Relación	Descripción	PI
1	No.	1	Número de la publicación	
2	Título	1	Título de la publicación	
3	Autores	1..n	Autores de la publicación	$PI_{1.1}$
4	Afiliación de los autores	1..n	Institución de afiliación de los autores	$PI_{1.1}$
5	Países	1..n	País de las instituciones de los autores	$PI_{1.1}$
6	Sede	1	Nombre de la sede de publicación	$PI_{1.2}$
7	Tipo de sede	1	Clasificación de las sedes de publicación	$PI_{1.2}$
8	Tipo de publicación	1	Tipos de publicación de las sedes	$PI_{1.2}$
9	Acrónimo de sede	1	Acrónimo del nombre de la sede	$PI_{1.2}$
10	Tasa de interés de la sede	1	Tasa de interés de publicación de la sede	$PI_{1.2}$
11	Tipo de investigación	1	Tipo de investigación del estudio	$PI_{1.3}$
12	Método de investigación	1	Método de investigación usado en el estudio	$PI_{1.3}$
13	Año	1	Año de publicación	$PI_{1.4}$
14	Entorno del estudio	1	Contexto en el que se aplica el estudio	$PI_{1.5}$
15	Área de conocimiento	1	Áreas de conocimiento del SWEBOK	$PI_{1.5}$
16	Dominio de contribución	1	Dominio en el que se aplica la contribución	$PI_{1.6}$
17	Tipo de contribución	1	Tipo de contribución que aporta la publicación	$PI_{1.6}$
18	Contribución	1..n	Contribución técnica de la publicación	$PI_{1.6}$
19	Componentes GraphQL	1..n	Componentes de GraphQL nombrados, ejemplificados, o utilizados en la publicación	$PI_{1.6}$
20	API pública	1..n	APIs públicas utilizadas en la publicación	$PI_{1.6}$

5.2.5 Evaluación de la validez

En la realización del SMS se ha intentado ser lo más metodológico posible, por lo que se ha evaluado la validez en base a las directrices de [121, 122], utilizando las siguientes consideraciones:

Validez descriptiva es la descripción con precisión y objetividad. Se intento minimizar esta amenaza introduciendo formularios de criterios de decisión para la categorización y clasificación general de estudios basados en la guía de Petersen *et al.* [122] (véase sección §5.2.4). Además, se introdujo criterios de clasificación específicos basados en los componentes del paradigma GraphQL (véanse secciones §3.3 y §3.4).

Validez teórica determina la capacidad de capturar lo que pretendemos capturar [122]. Para minimizar esta amenaza, en la "Identificación del Estudio", se estableció la cadena de búsqueda sólo con la palabra "graphql" para obtener una muestra adecuada respecto a la población objetivo. También se eligió dos estrategias para la búsqueda: se comenzó con una búsqueda automática a las bases de datos; y se la complementó con la estrategia de búsqueda manual, en donde se recuperó cuatro estudios los que se añadió al conjunto de datos del estudio inicial. Para minimizar el riesgo de sesgo en la extracción y clasificación de los datos, se realizó un mapeo individual con el primer investigador, seguido de la revisión de un segundo investigador. Las diferencias encontradas se validaron mediante una lectura conjunta del texto completo de la publicación y luego se discutieron hasta llegar a un consenso basado en las reglas de clasificación establecidas en la sección §5.2.4.

Generalización, según Petersen y Gencel [121], se debe considerar la generalización interna (dentro de una población) y la generalización externa (entre diferentes poblaciones). La validez interna asegura que el diseño experimental de un investigador sigue de cerca el principio de causa y efecto; por lo tanto, en este estudio, se siguió la mayoría de las recomendaciones metodológicas expuestas en la guía de Petersen *et al.* [122]. Sin embargo, la existencia de clasificaciones manuales en el proceso, expone que los investigadores introduzcan algunos errores de selección de estudios. Por este motivo, se incluyó a este estudio una amplia gama de trabajos (incluyendo la literatura gris) y se introdujo métodos de criterios de decisión en las clasificaciones para minimizar este impacto. La validez externa mide la aplicabilidad de los resultados del estudio a otras situaciones, grupos o eventos (Generalización). En este aspecto se intentó minimizar esta amenaza utilizando la guía de Petersen *et al.* [122], que es popular para la realización de SMS en la ingeniería del software. Además, se realizó la clasificación específica de los componentes de GraphQL utilizando el paradigma expuesto en el capítulo §3, el cual a su vez se basó en el sitio oficial de la especificación formal de GraphQL.

Validez interpretativa es cuando las conclusiones se extraen razonablemente, dados los datos. En este apartado, se minimizó la amenaza de sesgo en la interpretación de

las conclusiones de los resultados descritos por el primer autor, luego revisados por separado por los tres coautores, y después discutidos conjuntamente en reuniones de consenso para unificar las revisiones e interpretaciones.

Repetibilidad requiere información detallada sobre el proceso de investigación. En este estudio se informa detalladamente del proceso de SMS, se exponen los archivos del trabajo en repositorios digitales y se explican las medidas adoptadas para reducir las posibles amenazas a la validez.

5.3 Resultados

En esta sección, se responde a las preguntas de investigación definidas en la sección §5.1, mediante los resultados del proceso de extracción de datos realizado en la sección §5.2.4.

5.3.1 $PI_{1.2}$ ¿Quiénes son los autores e instituciones que investigan sobre el paradigma GraphQL?

En esta pregunta, se descubre a los investigadores e instituciones más activos en realizar estudios acerca del paradigma GraphQL.

Por un lado, se muestra la información de los autores que utilizaron el paradigma GraphQL en sus publicaciones, para lo cual, se consideró al primer autor de la publicación como autor principal y a los demás autores como coautores, obteniendo 298 autorías en las publicaciones (84 autorías principales y 214 coautorías) correspondientes a 270 investigadores. La figura §5.3 muestra un resumen de los autores por país y continente; en donde, se observa que los autores afiliados a instituciones europeas lideran el número de contribuciones con un 60,74 %; sin embargo, Estados Unidos es el país con más autores, seguido de Alemania, donde destaca el número de autorías de los autores principales.

La tabla §5.4 muestra los autores con mayor número de publicaciones.

Por otro lado, La figura §5.4 muestra las instituciones de afiliación de los autores; se agrupo las instituciones que tienen una publicación en la categoría "Otras". Se encontró 86 instituciones de 33 países, en donde la Universidad de Linkoping, de Suecia, es la que aportó más publicaciones, seguida de la Universidad Aalto, de Finlandia,

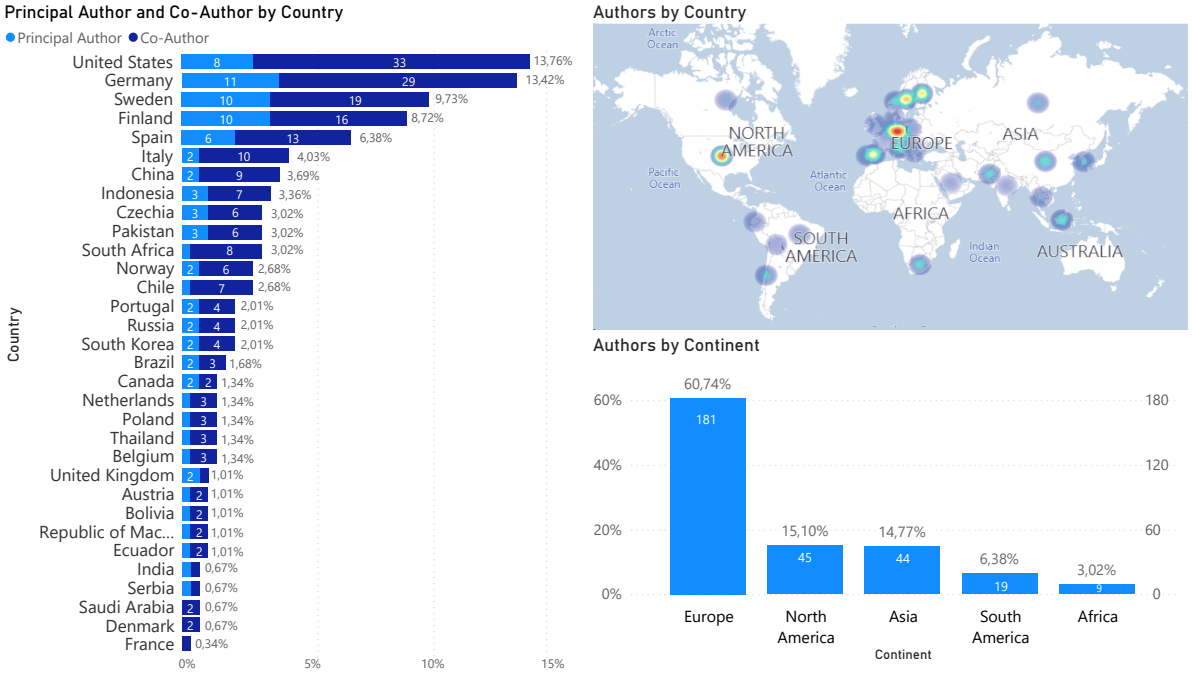


Figura 5.3: Autores por continente y por país

IBM Research, de Estados Unidos, y la Universidad RWTH Aachen, de Alemania.

5.3.2 $PI_{1.3}$ ¿Dónde se publicaron los trabajos de investigación?

En esta pregunta se identificó las sedes, tipos de sede, tipos de publicación donde se han publicado los estudios de GraphQL y su impacto. En este sentido, primero se identificó y ordenó los tipos de sede de acuerdo al rigor de la revisión de sus publicaciones; cabe mencionar que se consideró la literatura gris para verificar el interés de estudio de GraphQL en la academia. Luego, en las sedes se identificó y ordenó los siguientes tipos de publicación por su impacto: tesis de licenciatura, tesis de maestría, artículo de taller, artículo de conferencia y artículo de revista. Sin embargo, se identificó que las conferencias y talleres tienen dos formas de publicar sus estudios; la primera es publicar los estudios en las actas de las sedes, y la segunda es publicar las actas de la sede en un volumen de alguna revista, en donde el impacto de estos estudios se cataloga por el impacto de la revista; en este sentido, se subdividió estos tipos de publicación en: artículos de conferencia (revista) y artículos de taller (revista).

La figura §5.5 muestra las publicaciones por tipo de publicación y por tipo de sede. Se observó que la mayor parte de las publicaciones (46,07 %) son en conferencias, en

Tabla 5.4: Los principales autores por publicaciones

Autor	Publicaciones como autor principal	Publicaciones como coautor	Total Publicaciones
Hartig Olaf	3	3	6
Wittern Erik	2	1	3
Cha Lan	1	2	3
Taelman Ruben	2	-	2
Brito Gleison	2	-	2

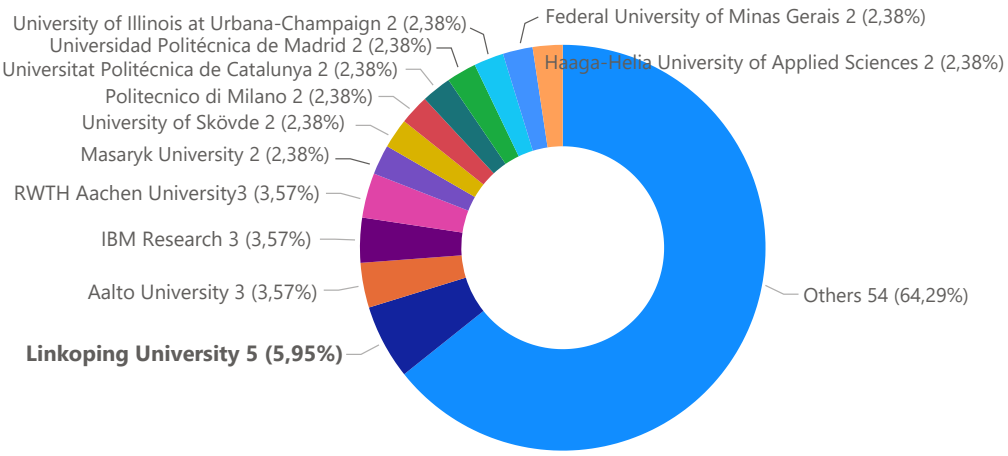


Figura 5.4: Publicaciones por afiliación

donde a su vez los estudios publicados en sedes revisadas por pares (revistas, conferencias, y talleres) son la mayoría 69,05 %, y la revisión académica es el 30,95 %.

A continuación, se muestra el impacto de las publicaciones medido por: GII-GRIN-SCIE (GGS), Scimago Journal & Country Rank (SJR), y Journal Citation Reports (JCR). En esta sección, se aclara que el impacto de la literatura gris no es medible; por lo tanto, solo se analizarán los papers de talleres, conferencias y revistas. La tabla §5.5 muestra los papers de los talleres y su impacto medido en GGS y SJR. En donde, el "Workshop on (Constraint) Logic Programming (WLP)", publicado en la revista Electronic Proceedings in Theoretical Computer Science, se destaca con un SJR: 0,33.

La tabla §5.6 muestra el impacto de los artículos de conferencia medido con GGS y SJR.

Según la métrica SJR, por un lado, las conferencias indexan sus actas directamente a la base científica *Scopus*, la cual asigna un índice SJR, pero no un cuartil; en este

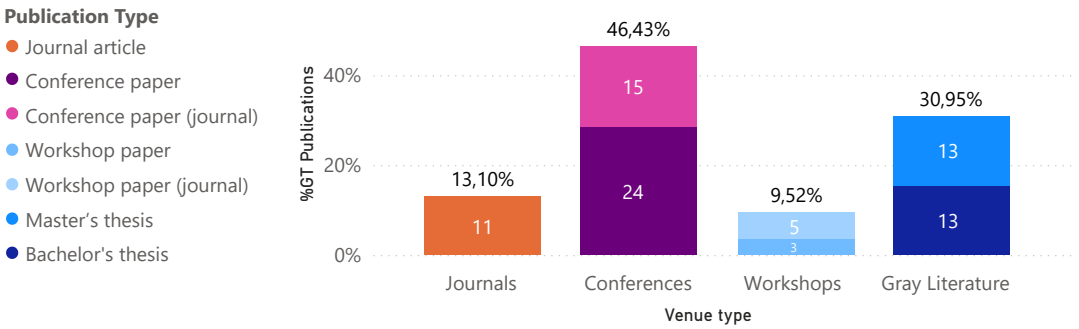


Figura 5.5: Publicaciones por tipo de sede y por tipo de publicación

Tabla 5.5: Artículos de taller

Tipo de Publicación	Clase GGS	Cuartil SJR	Año SJR	SJR	Número de Publicaciones	Publicaciones
Artículos de taller	-	-	-	-	3	[60, 109, 159]
Artículos de taller (revista)	Trabajo en progreso	-	2019	0.18	3	[61, 86, 179]
	-	-	2020	0.18	1	[54]
	-	-	2019	0.33	1	[113]

segmento, el congreso líder es SANER con un SJR: 0,29. Por otro lado, existe conferencias que publican sus actas en volúmenes de revistas indexadas en Scopus, las cuales suelen tener asignadas el índice SJR y cuartil; en este segmento, las conferencias más destacadas son CAISE, ICSOC, ICWE y MEDI, publicadas en la revista "Lecture Notes in Computer Science", que tiene un cuartil 2 (Q2) y SJR: 0,43. Según la métrica GGS, 19 de los 39 artículos de conferencias (48,72 %) tienen categorización en esta métrica; en este segmento, las mejores conferencias en el nivel 1 son WWW (Clase 1: A++) y ESEC/FSE (Clase 1: A+).

La tabla §5.7 muestra el impacto de los artículos de las revistas medido con las métricas JIF de JCR y SJR. Las revistas de mayor cuartil 1 (Q1) son Journal of Molecular Biology (JIF: 5,47, SJR: 3,19), Journal of Cheminformatics (JIF: 5,32, SJR: 1,43) e IT Professional (JIF: 3,70, SJR: 0,63).

5.3.3 $PI_{1.4}$ ¿Qué tipos y métodos de investigación se utilizaron en las publicaciones sobre el paradigma GraphQL?

En esta pregunta, se identifica la madurez de las publicaciones por tipo de investigación y el método de investigación utilizado para recoger pruebas empíricas sobre el paradigma GraphQL. La tabla §5.8 y la figura §5.6 muestran la clasificación de las

Tabla 5.6: Artículos de conferencia

Tipo de Publicación	Clase GGS	Cuartil SJR	Año SJR	SJR	Número de Publicaciones	Publicaciones
Artículos de Conferencia	Clase 1	-	-	-	3	[19, 62, 97]
	Clase 3	-	-	-	3	[126, 146, 177]
	Trabajo en progreso	-	-	-	6	[13, 79, 84, 102, 145, 149]
	-	-	2020	0.29	1	[10]
	-	-	-	-	11	[11, 18, 20, 29, 63, 66, 68, 130, 137, 157, 173]
Artículos de Conferencia (revista)	Clase 2	Q2	2019	0.43	3	[27, 174, 182]
	Clase 3	Q2	2019	0.43	2	[136, 183]
	Trabajo en progreso	Q3	2020	0.25	1	[142]
	Trabajo en progreso	Q3	2019	0.18	1	[117]
	-	Q2	2019	0.43	1	[40]
	-	Q3	2019	0.19	2	[16, 141]
	-	Q4	2020	0.16	1	[154]
	-	-	2019	0.20	2	[57, 171]
	-	-	2019	0.34	1	[110]
	-	-	-	-	1	[158]

publicaciones por tipo de investigación.

La figura §5.6 muestra, por un lado, que la mayoría de las publicaciones tienen un fuerte enfoque empírico, ya que se clasifican como investigación de ”evaluación” o ”validación” (61,91 %). Por otro lado, las publicaciones restantes muestran estudios sin evaluación empírica (es decir, no aplicaron los métodos de investigación descritos en la sección §5.2.4), clasificados como propuestas de solución, artículos de experiencia y artículos de opinión. Cabe destacar que no hubo publicaciones de tipo filosófico.

La figura §5.7 muestra los métodos de investigación utilizados en las investigaciones de ”validación” o ”evaluación”; para analizar el enfoque empírico de los estudios, como se sugiere en la sección §5.2.4. En concreto, se puede observar que el *estudio de caso* es el método de investigación más utilizado en la práctica de la ingeniería de software (es decir, la investigación de evaluación). En cambio, en la investigación de validación, los métodos más utilizados son los *experimentos* y *prototipos*.

Tabla 5.7: Artículos de revista

Revista	ISSN	Año JCR-SJR	Cuartil JIF (JCR)	Cuartil SJR	Publicaciones
Journal of Molecular Biology	0022-2836	2020	Q1-5.47	Q1-3.19	[138]
Journal of Cheminformatics	1758-2946	2019	Q1-5.32	Q1-1.43	[104]
IT Professional	1520-9202	2019	Q1-3.70	Q1-0.63	[132]
Molecules	1420-3049	2020	Q1-4.41	Q1-0.78	[70]
PLoS ONE	1932-6203	2020	Q2-3.24	Q1-0.99	[140]
Electronics	2079-9292	2019	Q2-2.41	Q2-0.30	[85]
BMC Medical Informatics and Decision Making	1472-6947	2019	Q3-2.31	Q1-0.91	[168]
Software and Systems Modeling	1619-1366	2020	Q3-1.91	Q2-0.42	[5]
Software Engineering and Knowledge Engineering	0218-1940	2019	Q4-0.89	Q3-0.25	[22]
Journal of Object Technology	1660-1769	2019	-	Q3-0.24	[156]
Theoretical And Applied Science	2308-4944	-	-	-	[95]

Tabla 5.8: Publicaciones por tipos de investigación

Tipo de investigación	**Publicaciones**
Investigación de evaluación	[4, 20, 41, 55, 63, 84, 97, 102, 104, 127, 130, 160, 168, 171, 174]
Investigación de validación	[1, 5, 10, 11, 18, 19, 22, 29, 34, 38, 40, 49, 54, 57, 60, 62, 65, 79, 85, 100, 106, 109, 110, 132, 137, 140, 141, 145, 146, 155, 156, 181, 182, 183]
Propuestas de solución	[59, 117, 126, 136, 142, 176, 177]
Artículos de experiencias	[13, 66, 70, 99, 138, 149, 150, 173]
Artículos de opinión	[16, 17, 31, 68, 95, 114, 135, 151, 154, 157, 158, 165, 166, 179]

5.3.4 $PI_{1.5}$ ¿Cuándo se publicaron los trabajos de investigación?

Esta pregunta muestra la evolución de las publicaciones de GraphQL desde 2015 hasta el 30 de junio de 2021. Se consideró los estudios desde 2015 porque es el año en que GraphQL se lanzó a la comunidad.

La figura §5.8 muestra el número de estudios por tipo de investigación y por año de publicación. De nuevo, se observa que el crecimiento de las publicaciones es constante en el tiempo, siendo 2019 el año con mayor crecimiento (17,86 %) respecto a 2018. Nótese que las publicaciones de los tipos de investigación que presentan evidencia empírica también aumentaron a lo largo de los años (tipos de investigación de ”validación” y ”evaluación”). Por lo tanto, se puede notar que, aunque GraphQL es un tema de investigación joven que mantiene una tendencia creciente, la comunidad científica

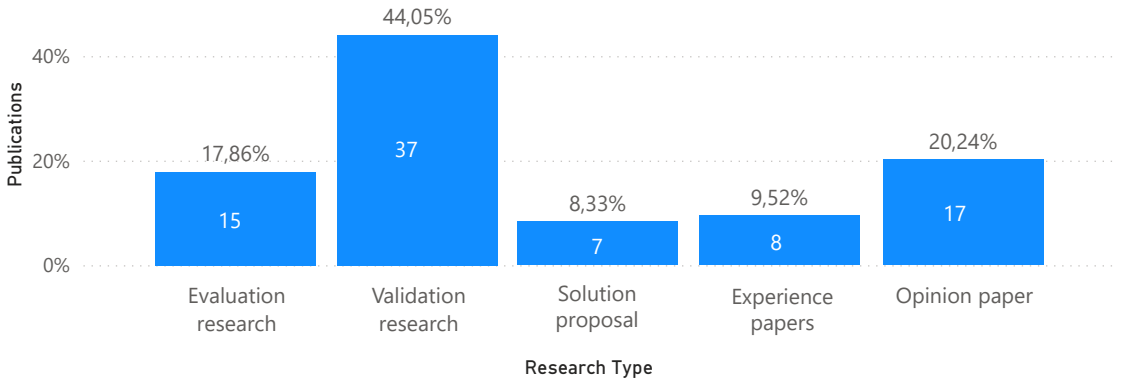


Figura 5.6: Publicaciones por tipo de investigación

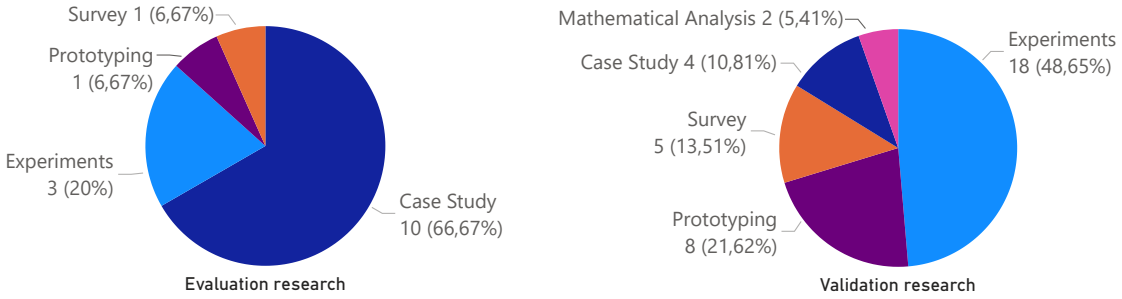


Figura 5.7: Publicaciones por métodos de investigación en los tipos de investigación de validación y evaluación

debe generar más estudios con evidencia empírica para mejorar y madurar la base de conocimiento sobre GraphQL.

5.3.5 $PI_{1.6}$ ¿Por qué se estudió el paradigma GraphQL?

En esta pregunta, se ha intentado determinar por qué los investigadores realizan estudios sobre GraphQL; en este sentido, acogemos las recomendaciones de Petersen [122] para responder la pregunta con las siguientes métricas: entorno de estudio, dominio de aplicación y áreas de conocimiento de ingeniería del software basadas en el SWEBOK.

La figura §5.9 muestra las publicaciones por entorno de estudio y dominio de aplicación. La mayoría de las publicaciones (71,43 %) se sitúan en el contexto académico y el resto en el contexto industrial; no se encontró estudios aplicados en el contexto gubernamental. También se analizó que los dominios de aplicación predominantes fueron el desarrollo (52,38 %) y la informática (21,43 %); esto indica que el mundo académico

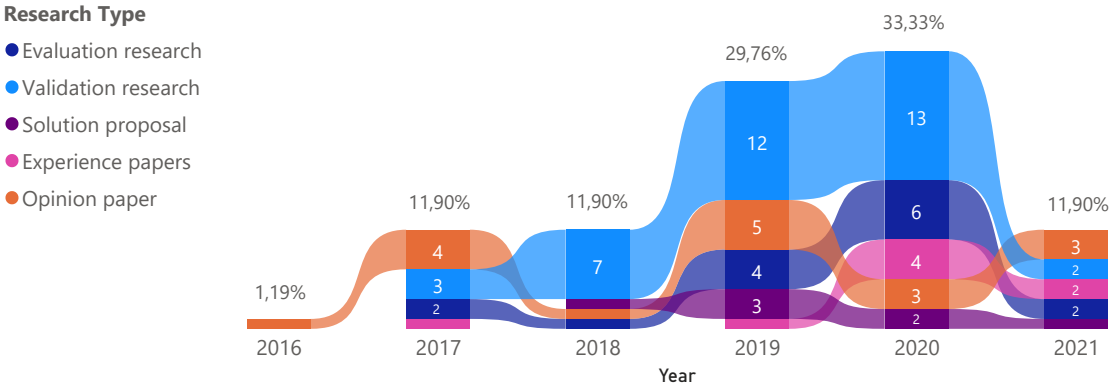


Figura 5.8: Publicaciones por tipo de investigación y por año

comenzó a estudiar los componentes técnicos del paradigma GraphQL antes de aplicarlos a otras disciplinas o ciencias.

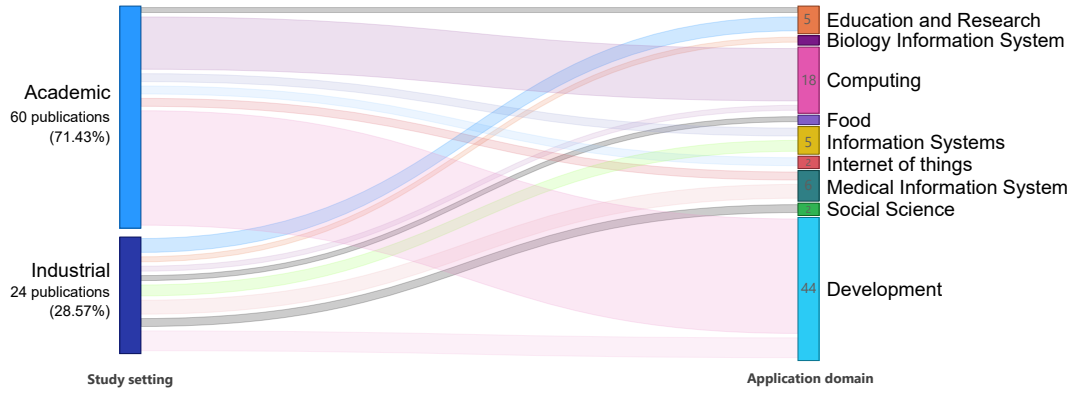


Figura 5.9: Publicaciones por ámbito de estudio y por dominio de aplicación

La figura §5.10 muestra las áreas de conocimiento de la Ingeniería del Software basadas en SWEBOK (ver Sec. §5.2.4) utilizadas en las publicaciones. Obsérvese que el área de conocimiento "construcción" es la más aplicada (55,95 %), seguida de lejos por "fundamentos informáticos". Estos resultados de las áreas de conocimiento coinciden y se complementan con los resultados de los dominios de aplicación, ratificando el interés de los investigadores en el estudio y aplicación técnica del paradigma GraphQL.

5.3.6 $PI_{1.7}$ ¿Cómo se introdujo el paradigma GraphQL en la comunidad científica?

Esta pregunta, se responde mediante la identificación de clasificaciones específicas del uso y aplicación del paradigma GraphQL en los estudios (basadas en las seccio-

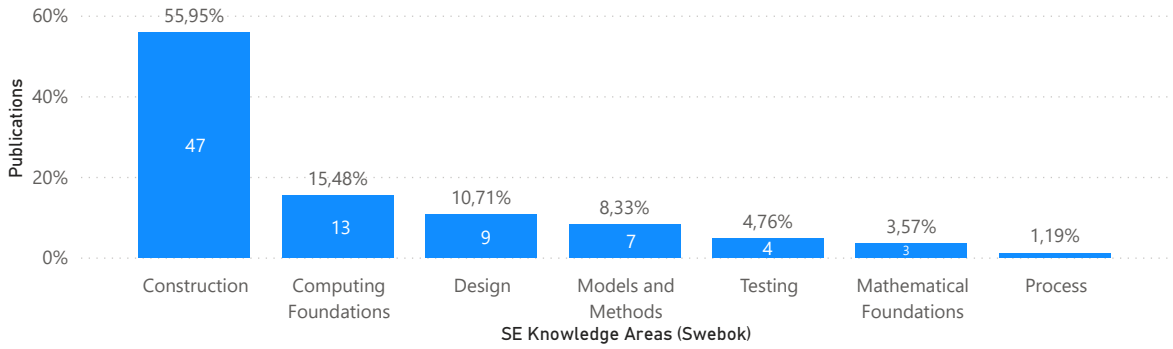


Figura 5.10: Publicaciones por áreas de conocimiento de la ingeniería del software

nes §3.3, §3.4, y §5.2.4), los cuales son: dominio de contribución, tipo de contribución, contribución, componentes GraphQL, y APIs públicas utilizadas.

Dominio de contribución: Establece el dominio de aplicación de la contribución técnica del servicio GraphQL en las publicaciones; basándose en la sección §3.3, se identificó los siguientes dominios: *Proveedor* y *Cliente* del servicio GraphQL. En el proceso de extracción de datos, se encontró 26 publicaciones que contribuyen a los dos dominios. Por lo tanto, se obtuvo 110 contribuciones de 84 publicaciones. La figura §5.11 muestra que la mayoría de las contribuciones en las publicaciones (72,73 %) se encontraban en el dominio del proveedor, es decir, la investigación se centró en el comportamiento del servicio GraphQL, por lo que evidencia la necesidad de reforzar los estudios que contribuyen al dominio del cliente del servicio GraphQL.

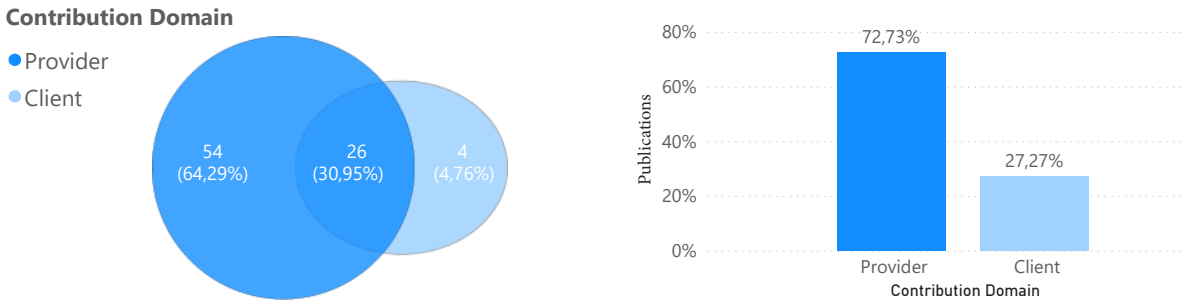


Figura 5.11: Publicaciones por dominios de contribución

Tipo de contribución: Esta es una clasificación de alto nivel de las contribuciones técnicas de GraphQL en las publicaciones. Al igual que en el segmento anterior, se en-

contró publicaciones que proporcionaban múltiples tipos de contribución; por lo tanto, se clasificó 127 contribuciones técnicas en 84 publicaciones. La figura §5.12 muestra que los tipos de contribución más frecuentes son la implementación y el análisis de GraphQL.

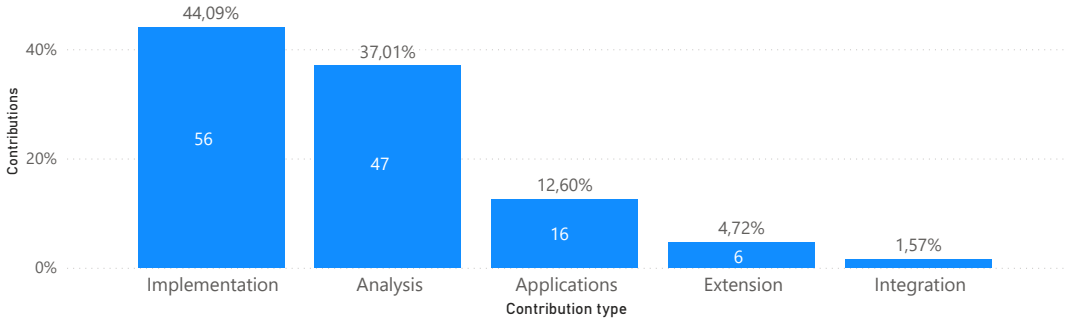


Figura 5.12: Tipos de contribución en las publicaciones

Contribuciones: Son las contribuciones técnicas específicas que utilizan el paradigma GraphQL en las publicaciones. La figura §5.12 muestra que la mayoría de las contribuciones son de tipo "Implementación" y "Análisis", por lo que a continuación se analizará estas dos contribuciones.

Por un lado, las contribuciones de tipo "Implementación" realizan soluciones como wrappers, migraciones, y sobre todo (43 de 56 contribuciones) implementaciones de APIs GraphQL. La figura §5.13 muestra que la mayoría de las implementaciones (72,37%) fueron en el dominio del proveedor; esto implica de nuevo una oportunidad de investigación en el consumo del servicio GraphQL.

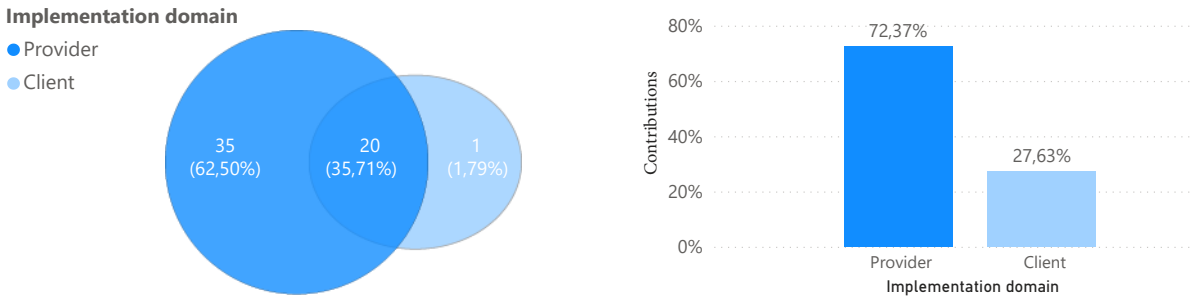


Figura 5.13: Tipo de contribución "Implementación" en las publicaciones

Por otro lado, las contribuciones de tipo "Análisis" presentaron trabajos como revisiones teóricas, análisis del tema, y las más destacadas (35 de 47 contribuciones) reali-

zaron comparaciones de GraphQL con otras tecnologías. En este sentido, la tabla §5.9 muestra las diferentes comparaciones realizadas en los estudios; nótese que la mayoría de ellos comparan GraphQL con REST, en donde, los principales resultados y conclusiones reportados mencionan que al utilizar GraphQL, lograron reducir el número de campos, el tiempo y el tamaño de las respuestas en las APIs implementadas; en consecuencia, se concluye que los investigadores analizan GraphQL como una alternativa al paradigma tradicional REST para la implementación de APIs.

Tabla 5.9: Comparaciones de GraphQL

Camparaciones	**Publicaciones**	**Porcentaje**
Entre REST y GraphQL	[1, 10, 11, 17, 35, 38, 41, 55, 57, 59, 63, 65, 85, 95, 102, 110, 113, 114, 135, 141, 149, 158, 160, 171, 173, 174, 181]	77.14 %
Entre esquemas de APIs GraphQL	[86, 182]	5.71 %
Otras comparaciones con GraphQL	[68, 100, 146, 155, 156, 179]	17.15 %
		100 %

Componentes de GraphQL : En esta sección, se mapeó los componentes GraphQL mencionados, utilizados y ejemplificados en las publicaciones para entender la profundidad cómo la comunidad científica estudió el paradigma GraphQL. La semántica establecida de las clasificaciones para los componentes es (i) *componentes mencionados*, se refiere a los componentes mencionados en las publicaciones pero no necesariamente utilizados; y (ii) *componentes ejemplificados*, se refiere a los componentes ejemplificados pero no necesariamente utilizados en las publicaciones; (iii) *componentes utilizados*, se identifica los componentes utilizados como parte de la contribución técnica de GraphQL en las publicaciones. La figura §5.14 muestra el mapeo específico de los componentes de GraphQL, sus frecuencias y el porcentaje sobre el número total de publicaciones.

Por un lado, los componentes más mencionados, ejemplificados y utilizados en las publicaciones son las consultas, los esquemas, los objetos, los escalares y las mutaciones, lo que confirma que estos componentes son fundamentales para construir APIs GraphQL. Por otro lado, los componentes menos estudiados son las directivas, la validación y las suscripciones, lo que revela lagunas de investigación en estos componentes. La falta de estudios sobre las suscripciones llama la atención ya que se trata una de las tres operaciones básicas del servicio GraphQL.

Classification	Component	Mentioned	%	Exemplified	%	Used	%
Service	Execution	42	50,00 %	26	30,95 %	40	47,62 %
	Introspection	28	33,33 %	8	9,52 %	19	22,62 %
	Resolvers	39	46,43 %	20	23,81 %	37	44,05 %
	Type System	36	42,86 %	12	14,29 %	27	32,14 %
	Validation	18	21,43 %	4	4,76 %	10	11,90 %
Type System Definition	Directives	17	20,24 %	10	11,90 %	10	11,90 %
	Enums	26	30,95 %	13	15,48 %	13	15,48 %
	Input Object	28	33,33 %	7	8,33 %	15	17,86 %
	Interfaces	31	36,90 %	11	13,10 %	13	15,48 %
	Objects	59	70,24 %	50	59,52 %	62	73,81 %
	Scalars	39	46,43 %	47	55,95 %	52	61,90 %
	Schemas	73	86,90 %	46	54,76 %	66	78,57 %
	Unions	26	30,95 %	8	9,52 %	8	9,52 %
Executable Definition	Fragments	20	23,81 %	13	15,48 %	9	10,71 %
	Mutations	63	75,00 %	24	28,57 %	42	50,00 %
	Query	80	95,24 %	65	77,38 %	75	89,29 %
	Subscription	31	36,90 %	6	7,14 %	5	5,95 %

Figura 5.14: Componentes GraphQL en publicaciones

APIs públicas utilizadas : Por último, se identificó la tendencia a utilizar APIs públicas en las publicaciones; en este sentido, se encontró en uso de 15 APIs GraphQL o REST públicas utilizadas en 15 publicaciones (17,86 %). Las API GraphQL públicas son GitHub en [10, 11, 19, 62, 97, 142, 182], Apollo Demo y Smalltalk GraphQL Demo en [109], y Yelp en [19, 109]. Las APIs REST utilizadas son GitHub en [11, 86, 146]; APIs.guru en [86, 97, 183]; arXiv en [10]; Kentico Cloud's Delivery en [17]; IBM Watson Language Translator en [183]; JAX-RS, jBPM, y KIE en [59]; Libraries. io en [86]; Toggl y Toggl Report en [151]; y Spotify en [18]. Se observó que las API REST y API GraphQL de GitHub son las más utilizadas, seguidas de APIs.guru; las cuales, se han hecho populares en la comunidad científica como fuente de información para estudios empíricos.

5.4 Conclusiones

En esta sección se presentan varias conclusiones obtenidas como respuesta a las preguntas de investigación planteadas en el estudio de mapeo sistemático; además se complementa con el análisis de algunas relaciones obtenidas entre los resultados.

- **PI**$_{1.2}$. En la sección §5.3.1 se observó que las instituciones de Europa son líderes en el estudio acerca de GraphQL y tienen la mayoría de publicaciones (60,74 %), seguidas con una distancia significativa por los otros continentes. Sin embargo, se notó que aún *no existe una comunidad científica especializada* en este tema, aunque se destaca las aportaciones de la Universidad de Linkoping.

- **PI**$_{1.3}$. En la sección §5.3.2 se identificó que el 46,43 % de publicaciones son artículos de conferencias, seguidos de la literatura gris, artículos de revistas y artículos de talleres. Además, en la figura §5.15 se muestra la relación con los resultados de PI$_{1.\ 4}$ y PI$_{1.\ 6}$ para analizar que tipos de investigación y en que contexto se realiza los estudios por las sedes; se observó que incluir la literatura gris en el estudio de mapeo sistemático fue una decisión adecuada ya que su producción aportó el 53,84 % de investigación de evaluación y validación.

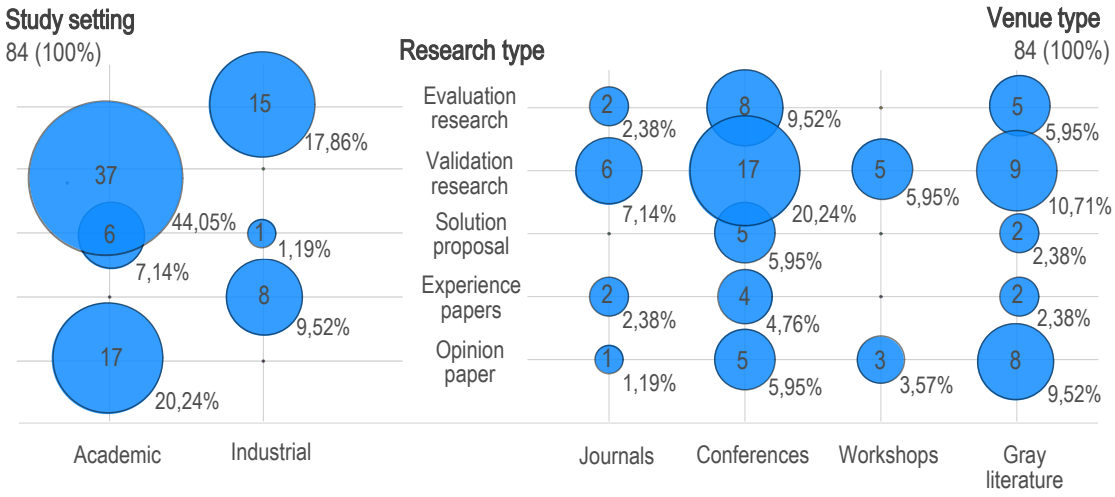


Figura 5.15: Publicaciones por contexto de estudio, por tipo de investigación y por tipo de sede

Aunque GraphQL es un tema joven, se destaca publicaciones en conferencias relevantes como WWW (Clase 1: A++) y ESEC/FSE (Clase 1: A+) y en revistas del cuartil 1 como "Molecular Biology, Cheminformatics" e "IT Professional". Sin

embargo, se concluye que los estudios acerca de GraphQL se debería aumentar y mejorar la calidad de sus pruebas empíricas.

- **PI**$_{1.4}$. En la sección §5.3.3 se identifica que la mayoría de investigaciones (61,91 %) son de tipo validación o evaluación, lo que significa que realizaron una evaluación empírica en sus estudios. Sin embargo, en estos estudios, sólo el 17,86 % se realizaron en la práctica de la industria de la ingeniería de software (es decir, investigación de evaluación). Además, se analizó los tipos de sede relativos a la investigación de "validación" y "evaluación", en donde, sorpresiva mente la literatura gris muestra un 16,66 % de estudios con evaluaciones empíricas incluso aplicadas en el entorno de la industria; esto indica que las universidades contribuyeron con estudios académicos relevantes en el estudio de GraphQL (véase figura §5.15).

  En este sentido, surge la siguiente pregunta ¿es posible mejorar la calidad de la evidencia empírica de los estudios GraphQL? Con la información recolectada en este estudio, se recomienda a los investigadores que para mejorar la calidad de la evidencia empírica, realicen sus estudios utilizando los métodos descritos en la sección §5.2.4.

- **PI**$_{1.5}$. En la sección §5.3.4 se encontró a lo largo del tiempo un crecimiento de estudios en cantidad y calidad. La figura §5.16 muestra la relación entre los resultados de las preguntas PI$_{1.\ 3}$, PI$_{1.\ 4}$, y PI$_{1.\ 5}$; en donde, se observa que a partir de 2019 aparecen trabajos de revistas que generalmente tienen más rigor para su publicación, así como la calidad de la evidencia empírica expuesta en investigaciones de "validación" y "evaluación". Estos resultados corroboran que la motivación para realizar el presente SMS fue acertada.

- **PI**$_{1.6}$. En la sección §5.3.5, por un lado, se analizó el entorno de estudio en relación con el tipo de investigación, en donde, se observó que la mayoría de publicaciones son de tipo de investigación de validación en el ámbito académico (véase la figura §5.15); esto sugiere que los investigadores realizaron sus estudios en contextos sintéticos sin aplicarlos en la práctica de la ingeniería del software. En este sentido, se encontró que otro punto clave para mejorar la calidad de la investigación de GraphQL es aplicar los estudios en contextos industriales y gubernamentales.

  Por otro lado, se observó que los estudios en las publicaciones utilizaron 7 de las 15 áreas de conocimiento de SWEBOK (véase figura §5.10), abriendo una oportunidad de investigación en el resto de las áreas de conocimiento descritas en la

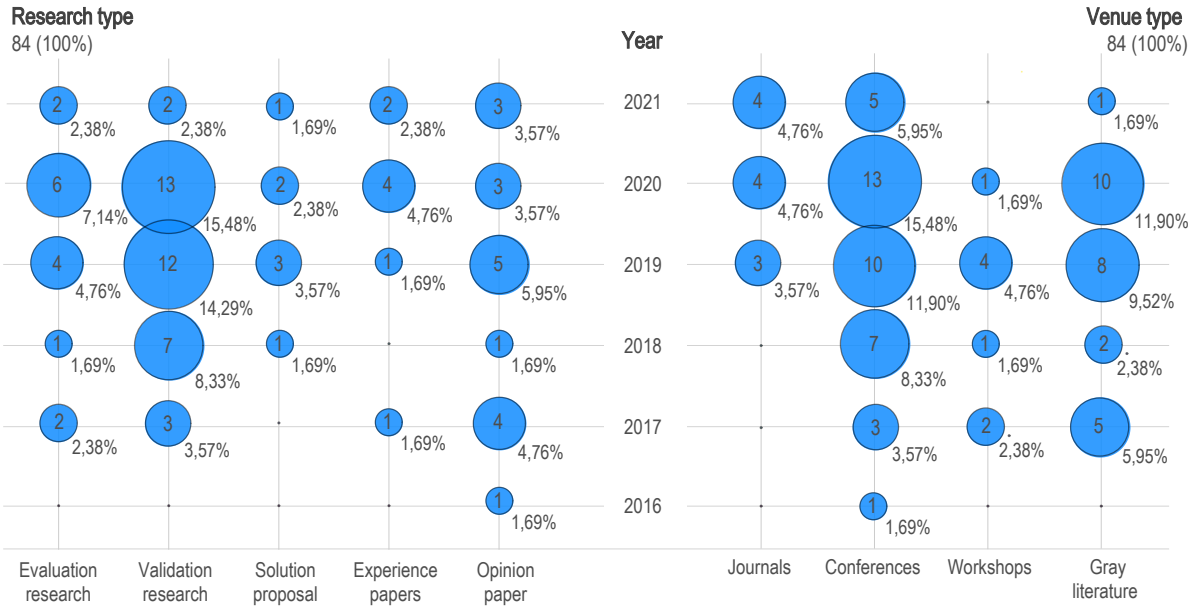


Figura 5.16: Publicaciones por tipo de investigación, por año y por tipo de sede

Sección §5.2.4.

- **PI**$_{1.7}$. En la sección §5.3.6 se responde cómo la comunidad científica estudió el paradigma GraphQL a través de las siguientes métricas de clasificación: contribuciones técnicas, sus tipos y dominios, componentes GraphQL y APIs públicas utilizadas en las publicaciones. La figura §5.17 muestra la relación entre PI$_{1.4}$ y PI$_{1.7}$, en donde, se nota que las contribuciones de tipo "implementación" en las conferencias son los estudios más concurrentes, dando a los investigadores una guía para entender lo que se está investigando y lo que falta investigar acerca de GraphQL y en donde publicarlo.

Las conclusiones encontradas en esta sección son en *primer lugar*, la identificación de una brecha de investigación en el dominio del cliente del servicio GraphQL, ya que el 72,73 % de las contribuciones de las publicaciones se realizaron en el dominio del proveedor (véase la figura §5.11). En *segundo lugar*, los tipos de contribución más frecuentes de los estudios son la implementación y el análisis de GraphQL (véase la figura §5.17) publicados con una frecuencia similar en artículos de conferencias y literatura gris; en este sentido, se observa que las implementaciones pueden mejorar su calidad mediante la validación o evaluación de sus estudios utilizando los métodos de investigación descritos en la sección §5.2.4. En *tercer lugar*, en el tipo de contribución

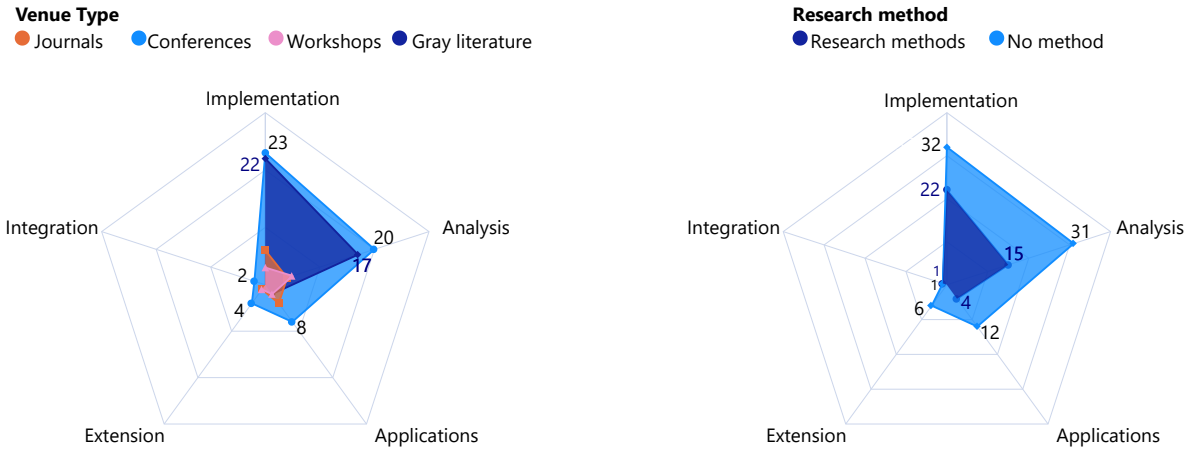


Figura 5.17: Publicaciones por tipo de contribución, por tipo de sede y por método de investigación

"análisis", se destacan las comparaciones entre REST y GraphQL (véase la Tabla §5.9), mostrando claramente que la comunidad estudió GraphQL como una alternativa a REST para implementar APIs web. En *cuarto lugar*, se logró mapear los componentes del paradigma GraphQL que fueron mencionados, utilizados y ejemplificados en las publicaciones. De manera general, se observa que los componentes más estudiados son las consultas, los esquemas, los objetos y los tipos escalares; concluimos que este comportamiento se debe a que estos componentes son fundamentales y necesarios para implementar APIs GraphQL (ver figura §5.14). En este sentido, se plantea otra pregunta de discusión ¿hay componentes de GraphQL que deberían ser estudiados más a profundidad? Esta pregunta nos sorprendió porque la mayoría de los componentes no son tratados en detalle, incluyendo componentes esenciales como las mutaciones y las suscripciones, evidenciando otra laguna en los estudios. Por *último*, se identificó que la API pública de GitHub es la más utilizada en las publicaciones, y también se identificó una cierta tendencia en implementar APIs REST públicas y sus equivalentes en APIs GraphQL.

5.5 Resumen

GraphQL es un enfoque novedoso para el desarrollo de APIs que está ganando adeptos, por lo cual, el foco principal de este apartado fue realizar un estudio de mapeo sistemático (SMS) de la literatura para mostrar una visión general de la investigación

e identificar las tendencias y lagunas en el campo de GraphQL.

El estudio de mapeo sistemático respondió a seis preguntas de investigación y clasificó los estudios en áreas generales y específicas sobre GraphQL. En particular, se centró en averiguar quién, dónde, cuándo, qué y por qué se investigó GraphQL, así como en contextualizar las áreas específicas de investigación con respecto al paradigma de GraphQL.

Como conclusiones generales, se observó que el estudio del paradigma GraphQL tiene una aceptación constante y creciente desde su aparición. A pesar de que la aceptación de GraphQL como un enfoque novedoso para el desarrollo de APIs es cada vez mayor entre los profesionales, todavía no existe una comunidad científica ampliamente establecida en torno a este tema.

En cuanto a los estudios analizados, se encontró una tendencia de publicación en sedes de mayor calidad, aunque es necesario mejorar la calidad de la evidencia empírica de los estudios mediante la aplicación de métodos de investigación ampliamente usados en la ingeniería de software, también se recomienda realizar los estudios en entornos industriales y gubernamentales para fortalecer la base de conocimiento de GraphQL.

Se encontró que los componentes establecidos en el paradigma GraphQL necesitan ser validados y evaluados con mayor rigor. Se entiende que el tema es muy joven y necesita madurar en algunos aspectos, por lo que este estudio ayudará a los investigadores a obtener una amplia visión de lo que se ha investigado y de los posibles temas de investigación de GraphQL en el futuro.

Parte III

Propuesta

Efectos de las arquitecturas REST y GraphQL: Diseño Experimental

El único modo de hacer un gran trabajo es amar lo que haces.

Steve Jobs (1955 – 2011),
Empresario y magnate estadounidense

En este capítulo, se propone un diseño experimental para analizar el proceso de desarrollo del software, con el propósito de comparar del efecto de los paradigmas de programación GraphQL y REST con respecto a la calidad del software, desde un punto de vista de los investigadores en un contexto académico.

En la sección §6.1, se destaca una breve discusión de antecedentes y motivaciones que impulsan el desarrollo de la propuesta del presente capítulo. En la sección §6.2, se detalla el diseño del entorno experimental, que se compone de los siguientes elementos: Objetivo del experimento; Factores y tratamientos; Variables de investigación; Hipótesis; Diseño; Sujetos experimentales; Tareas experimentales; Instrumentación; Recolección de datos; y Análisis. Por último, la sección §6.3 resume el capítulo.

6.1 Introducción

La motivación de este capítulo es contestar la pregunta de investigación **PI**$_2$: ¿Qué condiciones y circunstancias hacen más recomendable seguir el paradigma GraphQL frente al paradigma REST?.

Por lo cual, se propone realizar un experimento controlado que compare los efectos del uso del paradigma REST frente al paradigma GraphQL en la calidad del software. En donde, para caracterizar las condiciones y circunstancias del uso de los paradigmas, se utilizó las subcaracterísticas *Facilidad de Aprendizaje* de la calidad interna, y *Eficiencia* de la calidad en uso del producto de software basada en la norma ISO/IEC 25010 [75]. Para complementar las circunstancias del uso de los estilos REST y GraphQL se realizará el estudio de la curva de aprendizaje de estos paradigmas.

Por lo tanto, la pregunta de investigación **PI**$_2$ se concreta en las siguientes preguntas:

- **PI**$_{2.1}$: ¿Cuál es el efecto del paradigma de desarrollo en la calidad del software?.
- **PI**$_{2.2}$: ¿Cuál es la curva de aprendizaje de los paradigmas REST y GraphQL en el proceso de desarrollo de APIs?.

6.2 Entorno experimental

6.2.1 Objetivos de la investigación

Para definir el objetivo de este experimento controlado, se utilizó el enfoque *Objetivo-Pregunta-Métrica* (GQM, por sus siglas en inglés de *Goal Question Metric*) de Basili [6], en donde, se consideró:

- **Analizar** el proceso de desarrollo del software
- **Con el propósito de** comparar el efecto de los paradigmas de programación
- **Con respecto** a la calidad del software
- **Desde el punto de vista** del investigador
- **En el contexto** académico

6.2.2 Factores y tratamientos

El factor que se investiga es el paradigma de desarrollo de APIs, concretamente en el desarrollo del servicio (Backend).

Los *tratamientos* que se aplicarán a este factor son:

- Paradigma GraphQL para el desarrollo de APIs.
- Paradigma REST para el desarrollo de APIs (considerada como el nivel de control).

El Paradigma GraphQL, es un lenguaje de consulta y tiempo de ejecución de APIs; el cuál se introdujo en el Capítulo §3.

El Paradigma REST, es un estilo arquitectónico para el desarrollo de APIs, descrito en la Sección §3.2.1.

6.2.3 Variables

Por un lado, se definió como *variable independiente* al "Paradigma de desarrollo de APIs", aplicado con dos tratamientos: paradigma GraphQL y paradigma REST. Esta variable se operativiza principalmente en las tareas experimentales y de formación. En donde, los requisitos de cada tarea se especifican en lenguaje natural complementado con un diagrama relacional, una copia de seguridad de la base de datos (incluidos datos de prueba), y un instrumento para registrar la evidencia y tiempo de solución de cada requisito. Las tareas fueron diseñadas por los experimentadores, para que se desarrollen tanto en el paradigma REST, como GraphQL.

Por otro lado, se definió como *variable dependiente* a la "Calidad del Software"; la cual, se conceptualiza como la capacidad del producto de software de satisfacer las necesidades manifestadas e implícitas al ser usadas bajo condiciones específicas [74]. Esta variable, primero se operativiza desde el punto de vista de las subcaracterísticas de calidad del producto de software: *i) Facilidad de Aprendizaje* de la calidad interna [75], medida con la métrica "Completitud de la guía del usuario" establecida en la ISO/IEC 25023 [77]; y *ii) Eficacia* de la calidad en uso del producto de software [75], medida con la métrica "Tareas completadas" establecida en la ISO/IEC 25022 [76], en donde, estas subcaracterísticas servirán para contestar la pregunta $\mathbf{PI}_{2.1}$. Luego, se

complementa el análisis de la operativización de la variable con el cálculo de *iii) La Curva de aprendizaje* de los paradigmas de programación, medido por el "Tiempo de solución" de los requisitos de las tareas. En donde, la conceptualización de la curva de aprendizaje aplicada a personas conocida también como la generación de aprendizaje individual, menciona que es la mejora que se obtiene cuando las personas repiten un proceso y adquieren habilidad o eficiencia en razón de su propia experiencia [21], esta métrica se utilizará para contestar la pregunta $\mathbf{PI}_{2.2}$.

A continuación, se detalla las funciones de las métricas establecidas para medir la variable dependiente:

- *La completitud de la guía de usuario*, se centra en establecer la pregunta ¿Qué proporción de funciones se explica con suficiente detalle en la documentación del usuario o en la función de ayuda para permitir al usuario aplicar las funciones? [77]. La función de medición es:

$$x = \frac{(nFunc)}{nFuncImplemented} \tag{6.1}$$

  Donde, *nFunc* es el número de funciones descritas en la documentación del usuario o en la función de ayuda según sea requerido; y *nFuncImplemented* es el número de funciones implementadas que se requieren sean documentadas.

- *Las tareas completadas*, se centra en medir el rendimiento del usuario mediante la proporción de las tareas que se completa correctamente sin ayuda [76]. La función de medición es:

$$x = \frac{(nTasksCompleted)}{totalTasksAttempted} \tag{6.2}$$

  Donde, *nTasksCompleted* es el número de tareas únicas completadas; y *totalTasksAttempted* es el número total de tareas únicas intentadas.

- *La curva de aprendizaje* se define como [21]:

$$x = (timeFirstTask * nTimesDiscover)^{\frac{LOG10(rateLearning)}{LOG10(2)}} \tag{6.3}$$

  Donde, *timeFirstTask* es el tiempo de la primera tarea; *nTimesDiscover* es la unidad de veces que deseamos descubrir; y *rateLearning* es el índice de aprendizaje.

6.2.4 Hipótesis

Basado en la pregunta de investigación $\mathbf{PI}_{2.1}$, se ha definido las siguientes *hipótesis* para el experimento:

- H_0 (*Hipótesis nula*): No hay diferencia en los efectos de los paradigmas de programación en la calidad del software.
- H_1 (*Hipótesis alternativa 1*): Existe diferencia entre los efectos del paradigma de programación en la calidad del software. La calidad en la facilidad de aprendizaje del desarrollo de APIs que produce el paradigma GraphQL, es superior a la calidad que produce el paradigma REST.
- H_2 (*Hipótesis alternativa 2*): Existe diferencia entre los efectos del paradigma de programación en la calidad del software. La calidad en la facilidad de aprendizaje del desarrollo de APIs que produce el paradigma REST, es superior a la calidad que produce el paradigma GraphQL.

6.2.5 Diseño

Para este estudio de investigación se considera las fases del proceso experimental propuesto por Jedlitschka *et al.* [78] que son: la planificación, ejecución y análisis; esta decisión se tomó debido a que el ámbito de estudio se centra en experimentos controlados con humanos de acuerdo a la clasificación de Zelkowitz *et al.* [187]. En este sentido, las tareas que se pueden llevar a cabo son de: *Formación*, *Experimentales*, y *Análisis*, respectivamente a las fases del proceso experimental establecido [78].

El factor para el experimento será examinado en dos niveles utilizando un diseño de *medidas repetidas* debido a su solidez frente a la variación entre los sujetos experimentales [131]. Todos los sujetos deberán recibir ambos tratamientos experimentales, es decir, se les entregará requisitos para realizar una tarea de desarrollo de una API con el paradigma REST (*PR*) y otra tarea de desarrollo de una API con el paradigma GraphQL *(PG)*; en consecuencia, el efecto del tratamiento se podrá observar en todos los participantes. En este sentido, para evitar que surja algún efecto involuntario en los resultados debido a que los participantes reciban los dos niveles del tratamiento en un orden específico; se definió un *diseño cruzado* [83]. El diseño cruzado, es un diseño de medidas repetidas en donde los sujetos son asignados aleatoriamente a diferentes secuencias de los tratamientos [131]. Como hay dos niveles para el tratamiento, las secuencias posibles son: *PR-PG* (secuencia 1) y *PG-PR* (secuencia 2). Los participantes serán asignados al azar a una de estas secuencias.

La Tabla §6.1 muestra el diseño del experimento que se realizará en dos periodos de trabajo; cada período tendrá una duración de 2 horas para realizar una tarea experimental de desarrollo de una API. En el primer periodo se asignará los requisitos

para el desarrollo de una API, de uno de los siguientes casos:*eventos*, *facturas*, *notas*. El segundo periodo se asignará los requisitos para el desarrollo de una API con diferente paradigma y caso del asignado en el primer periodo. Cabe mencionar que los tres casos tienen el mismo nivel de complejidad; las mismos que se describen con más detalle en la Sección §6.2.7. La asignación de los grupos de participantes a las secuencias de tareas se determinará de forma aleatoria. Además, la justificación y validez del diseño se analizará en detalle en la Sección §9.4.

Tabla 6.1: Diseño del Experimento

Periodo / Secuencia	Periodo 1	Periodo 2
Grupo I: PR-PG	Paradigma REST	Paradigma GraphQL
Grupo II: PG-PR	Paradigma GraphQL	Paradigma REST

6.2.6 Sujetos experimentales

Los sujetos experimentales también llamados *participantes* se escogieron mediante un muestreo por conveniencia; tomando en cuenta que tuvieran los conocimientos de programación necesarios para comenzar un curso de desarrollo de APIs. Los participantes escogidos fueron 24 estudiantes de quinto nivel de la Carrera de Ingeniería en Software de la Universidad Técnica del Norte (Ibarra - Ecuador) matriculados en el periodo de abril del 2021.

La caracterización de los participantes se realizó al inicio del periodo académico mediante una breve encuesta demográfica y diagnóstica antes que estos reciban un periodo de formación en desarrollo de APIs. En la encuesta se consulta información acerca de la percepción del nivel de facilidad del uso, consumo y aprendizaje de los paradigmas de desarrollo REST y GraphQL. Para lo cual, se usó una escala ordinal de 5 puntos (muy fácil, fácil, regular, difícil, y muy difícil), en donde, se evidenció que 7 de los 24 participantes conocían acerca de los paradigmas de desarrollo. La Figura §6.1 muestra las respuestas de la percepción del nivel de facilidad de los siete participantes que tenían conocimiento de los paradigmas de desarrollo de APIs.

Tareas de formación

La formación en el desarrollo de APIs fue parte de las clases regulares impartidas en la Carrera de Software, que duró 3 semanas (compuestas por cinco horas síncronas y tres horas autónomas *(ha)* por semana) en modalidad virtual; en donde, para moti-

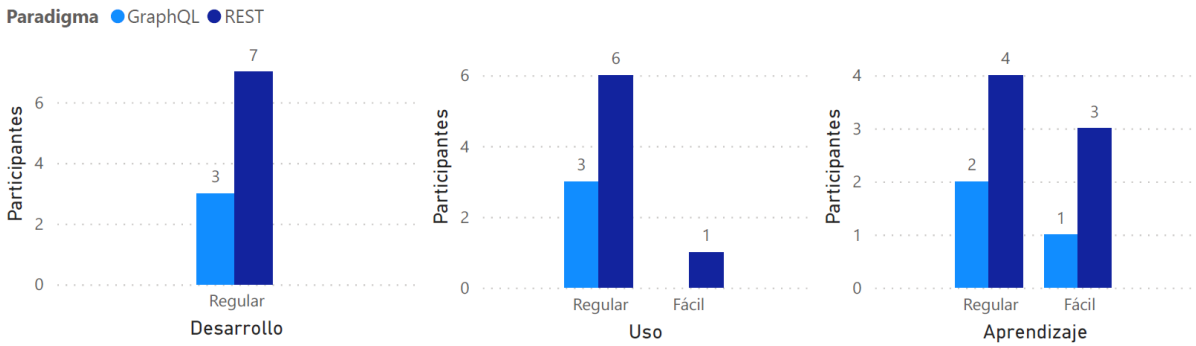


Figura 6.1: Respuestas de la encuesta diagnóstica

var a los participantes a realizar las tareas de formación, estas fueron calificadas. Las tareas fueron las siguientes: *i) Proyecto de desarrollo*, el objetivo fue instalar el ambiente de desarrollo (ver Sección §6.2.8), crear un proyecto de programación, configurar la conexión a la base de datos, y realizar tareas de consulta de datos desde el proyecto creado. *ii) Desarrollo API REST*, el objetivo es desarrollar una API REST, utilizando como base el proyecto creado en el apartado anterior, y consumido mediante la aplicación Postman[1]. *iii) Desarrollo API GraphQL*: el objetivo es desarrollar una API GraphQL, utilizando como base el proyecto creado en el primer apartado, y consumido mediante la aplicación GraphiQL[2]. Los casos utilizados en las tareas de formación fueron *Pizzas, Libros, y Blogs*, en donde, los artefactos utilizados fueron: un documento de requisitos y registro de solución, un diagrama relacional, una copia de seguridad y un script de creación de la base de datos. Los casos tienen diferente dificultad que básicamente se presenta en la estructura y manejo de consultas de datos (ver Anexo §B.1). La Tabla §6.2 muestra las tareas, casos y actividades de formación previas a la ejecución del experimento.

Durante la formación, los participantes asistieron a un total de seis clases síncronas en tres semanas. Cada semana se conformó por dos clases síncronas (3 horas y 2 horas), en donde en cada actividad, se realiza una introducción teórica, una demostración práctica, un taller práctico de desarrollo de una API asistido por un experimentador, y como paso final se envío un deber para afianzar los conocimientos.

Inicialmente, se empezó con 24 estudiantes como participantes, de los cuales uno se retiró en el transcurso de la formación, quedando finalmente 23 participantes para el experimento.

[1]Ver https://www.postman.com
[2]Ver https://github.com/graphql/graphiql

Tabla 6.2: Planificación de las tareas de formación

Tareas	Caso	Actividades	Duración (horas)
Proyecto de desarrollo	-	Introducción teórica a la arquitectura de software	0.5
	Pizzas	Ejemplo de creación de proyecto de desarrollo, conexión y consulta de base de datos	2.5
	Libros	Taller práctico de creación de proyecto y consulta de datos	2
	Blogs	Deber de creación de proyecto y consulta de datos	3 *(ha)*
Desarrollo API REST	-	Introducción teórica al paradigma REST	1
	Pizzas	Ejemplo de desarrollo de un API REST	2
	Libros	Taller práctico de creación de un API REST	2
	Blogs	Deber de creación de un API REST	3 *(ha)*
Desarrollo API GraphQL	-	Introducción teórica al paradigma GraphQL	1
	Pizzas	Ejemplo de desarrollo de un API GraphQL	2
	Libros	Taller práctico de creación de un API GraphQL	2
	Blogs	Deber de creación de un API GraphQL	3 *(ha)*
Leyenda: *ha* - horas autónomas		Total	24

6.2.7 Tareas experimentales

En esta sección, se diseñó tres tareas experimentales (eventos, facturas, notas) que tienen el mismo grado de complejidad entre ellas; y un grado de complejidad promedio comparado con las tareas de formación. Cada tarea tiene como objetivo implementar un API que realice operaciones de datos (*CRUD*, por su acrónimo en inglés de Create, Read, Update, and Delete); pero no implica la creación de interfaces de usuario.

Previo a la realización de la tarea, se entregó a los participantes los siguientes artefactos: una copia de seguridad de la base de datos; un script en lenguaje SQL de la estructura de la base de datos y datos de prueba (en el caso que el participante tenga inconvenientes en la carga de la copia de seguridad); un diagrama relacional de la base de datos; un archivo Microsoft Excel con los requisitos de las operaciones que deben implementar, y una sección para que puedan registrar los tiempos y evidencias de resolución de cada requisito; y por último una plantilla de código fuente del proyecto (basado en el ejemplo revisado en la etapa de formación).

A continuación, se detalla los requisitos de las tareas experimentales:

Tarea experimental 1: Gestión de Eventos

El objetivo de esta tarea es implementar una API que contenga las operaciones para la gestión de eventos, de acuerdo a la estructura de datos que se presenta en el diagrama de la Figura §6.2.

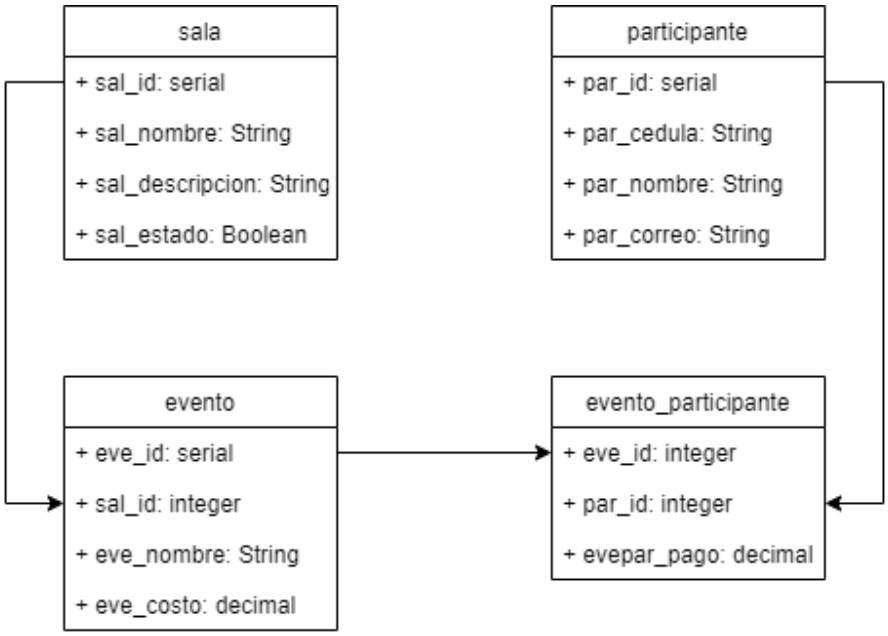


Figura 6.2: Diagrama relacional de la base de datos de eventos

Los requisitos de operaciones de datos que debe proveer la API son:

- Insertar un registro de evento
- Actualizar un registro de evento
- Eliminar un registro de evento
- Consultar los registros de los eventos
- Consultar el registro de un evento, filtrado por el campo: *eve_id*
- Asignar un participante a un evento
- Eliminar un participante de un evento
- Consulta compuesta: Consultar las salas y sus eventos
  - tablas: sala, evento
  - campos: sal_id, sal_nombre, eve_nombre, y eve_costo
- Consulta compuesta: Consultar un evento y sus participantes
  - tablas: evento, evento_participante, participante

  - campos: eve_nombre, par_cedula, par_nombre, y evepar_pago
  - parámetro: eve_id

- Consulta compuesta: Consultar las salas y eventos de un participante

  - tablas: sala, evento, evento_participante, participante
  - campos: sal_nombre, eve_nombre, eve_costo, par_nombre, y evepar_pago
  - parámetro: par_id

Tarea experimental 2: Gestión de Facturas

El objetivo de esta tarea es implementar una API que contenga las operaciones para la gestión de facturas, de acuerdo a la estructura de datos que se presenta en el diagrama de la Figura §6.3.

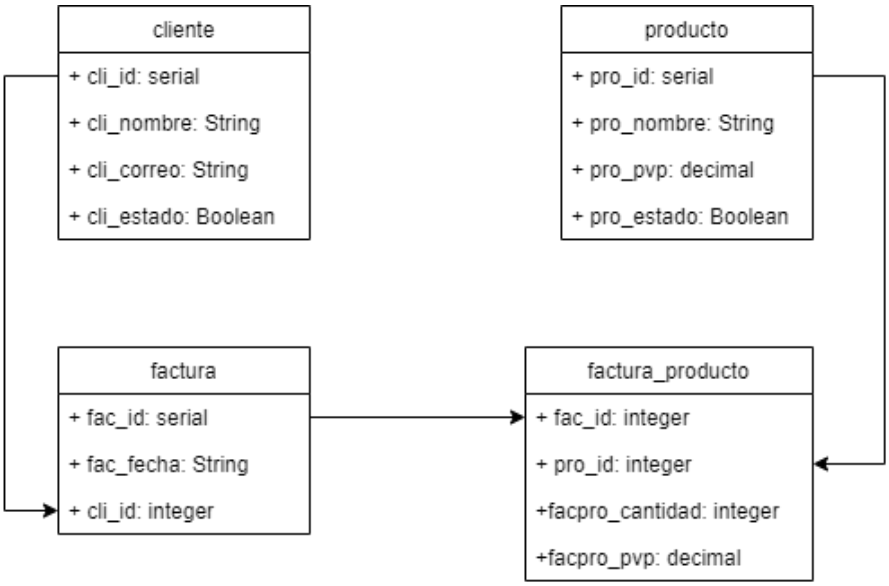


Figura 6.3: Diagrama relacional de la base de datos de facturas

Los requisitos de operaciones de datos que debe proveer la API son:

- Insertar un registro de factura
- Actualizar un registro de factura
- Eliminar un registro de factura
- Consultar los registros de las facturas
- Consultar el registro de una factura, filtrado por el campo: *fac_id*
- Asignar un producto a una factura
- Eliminar un producto de una factura

- Consulta compuesta: Consultar los clientes y sus facturas
  - tablas: cliente, factura
  - campos: cli_id, cli_nombre, fac_id, y fac_fecha
- Consulta compuesta: Consultar una factura y sus productos
  - tablas: factura, factura_producto, producto
  - campos: fac_id, pro_id, pro_nombre, y facpro_cantidad
  - parámetro: fac_id
- Consulta compuesta: Consultar las facturas con sus productos, de un cliente
  - tablas: cliente, factura, factura_producto, producto
  - campos: cli_nombre, fac_id, pro_nombre, pro_pvp, y facpro_cantidad
  - parámetro: cli_id

Tarea experimental 3: Gestión de Notas

El objetivo de esta tarea es implementar una API que contenga las operaciones para la gestión de notas de alumnos, de acuerdo a la estructura de datos que se presenta en el diagrama de la Figura §6.4.

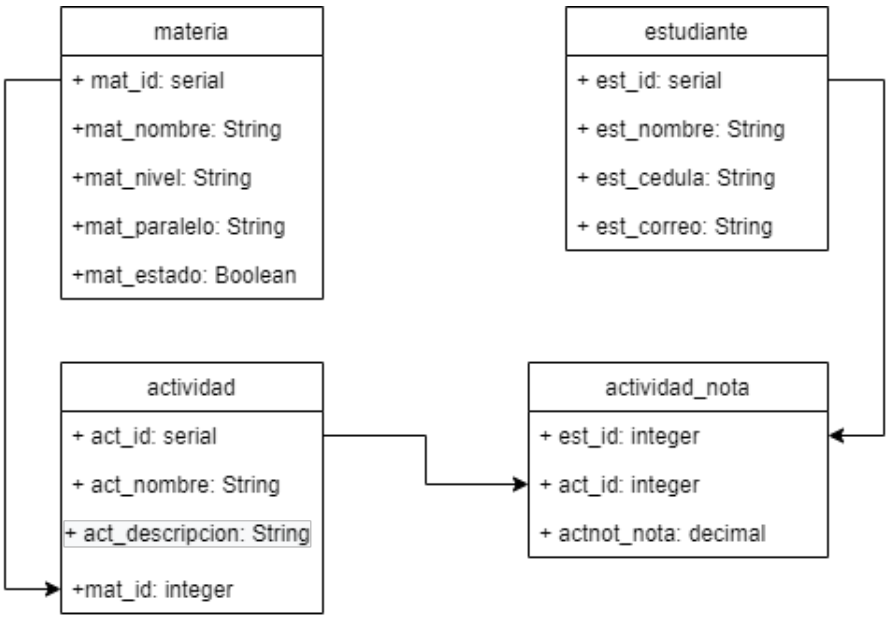


Figura 6.4: Diagrama relacional de la base de datos de notas

Los requisitos de operaciones de datos que debe proveer la API son:

- Insertar un registro de actividad

- Actualizar un registro de actividad
- Eliminar un registro de actividad
- Consultar los registros de las actividades
- Consultar el registro de una actividad, filtrado por el campo: *act_id*
- Asignar un estudiante a una actividad
- Eliminar un estudiante de una actividad
- Consulta compuesta: Consultar las materias y sus actividades
  - tablas: materia, actividad
  - campos: mat_id, mat_nombre, act_nombre, y act_descripcion
- Consulta compuesta: Consultar una actividad y la nota de los estudiantes
  - tablas: actividad, actividad_nota, estudiante
  - campos: act_nombre, est_cedula, est_nombre, y actnot_nota
  - parámetro: act_id
- Consulta compuesta: Consultar las materias, actividades y notas, de un estudiante
  - tablas: materia, actividad, actividad_nota, estudiante
  - campos: mat_nombre, act_nombre, act_descripcion, est_nombre, y actnot_nota
  - parámetro: est_id

6.2.8 Instrumentación

En esta sección se lista los artefactos que componen la instrumentación en cada fase del proceso experimental. La descripción completa de los artefactos se pueden ver en el paquete de laboratorio del experimento disponible en el Anexo §B.

Instrumentación de las tareas de formación

Encuesta demográfica/diagnóstica: Formulario de Microsoft Forms 365 Online (Anexo §B.1.1)

Ambiente de desarrollo, se conforma de varias aplicaciones y tecnologías que son similares para los paradigmas REST y GraphQL, las cuales se especifican a continuación:

- Ambiente de desarrollo integrado IDE: Visual Studio Code[3] v1.62

[3]Ver https://code.visualstudio.com

- Lenguaje de programación: JavaScript[4]
- Entorno de tiempo de ejecución de JavaScript: NodeJS[5] v14.17.5
- Librerías npm[6] para REST/GraphQL: pg-promise v10.10.2, express v4.17.1
- Librerías npm para GraphQL: graphql v15.5.0, graphql-server-express v1.4.1, graphql-import v1.0.2, y graphql-tools v7.0.5
- Motor de base de datos: PostgreSQL[7] v13, pgAdmin[8] 4 (herrramienta de administración)
- Aplicación cliente para el consumo del API REST: Postman[9] v8.12.2
- Aplicación cliente para el consumo del API GraphQL: GraphiQL[10]

Introducción teórica, se trata de artefactos de tipo presentaciones de Microsoft Power Point 365[11], que introducen la teoría de los siguientes temas:

- Ambiente de desarrollo y arquitectura de software (Anexo §B.1.2).
- Estilo arquitectónico REST (Anexo §B.1.3).
- Lenguaje de consultas GraphQL (Anexo §B.1.4).

Tareas de formación, contienen artefactos de requisitos y diseño de base de datos para el desarrollo de APIs, que fueron utilizados en las actividades de ejemplos, talleres prácticos y deberes (ver tabla §6.2). La Tabla §6.3 muestra el listado de artefactos usados como instrumentación de las tareas de formación.

Instrumentación de las tareas experimentales

Ambiente de desarrollo, se realizó en el mismo ambiente de desarrollo que las tareas de formación.

[4] Ver https://www.javascript.com
[5] Ver https://nodejs.org
[6] Ver https://www.npmjs.com
[7] Ver https://www.postgresql.org
[8] Ver https://www.pgadmin.org
[9] Ver https://www.postman.com
[10] Ver https://github.com/graphql/graphiql
[11] https://www.microsoft.com/en-ww/microsoft-365/powerpoint

Tabla 6.3: Instrumentación de las tareas de formación

Tarea	Artefactos	Anexo
Gestión de Pizzas	Respaldo de la base de datos (archivo .backup) Script estructura de la bdd y datos de prueba (archivo .sql) Diagrama relacional de la base de datos (archivo .png) Requisitos y registro del tiempo de solución (Microsoft Excel 365)	§B.1.5
Gestión de Libros	Respaldo de la base de datos (archivo .backup) Script estructura de la bdd y datos de prueba (archivo .sql) Diagrama relacional de la base de datos (archivo .png) Requisitos y registro del tiempo de solución (Microsoft Excel 365)	§B.1.6
Gestión de Blogs	Respaldo de la base de datos (archivo .backup) Script estructura de la bdd y datos de prueba (archivo .sql) Diagrama relacional de la base de datos (archivo .png) Requisitos y registro del tiempo de solución (Microsoft Excel 365)	§B.1.7

Tareas experimentales, al igual que las tareas de formación, contienen artefactos de requisitos y diseño de base de datos para el desarrollo de APIs, que deben implementarse en los paradigmas REST y GraphQL, de acuerdo al diseño del experimento establecido en la tabla §6.1. La instrumentación utilizada en las tareas experimentales se listan en la Tabla §6.4.

Tabla 6.4: Instrumentación de las tareas experimentales

Tarea	Artefactos	Anexo
Gestión de Eventos	Respaldo de la base de datos (archivo .backup) Script estructura de la bdd y datos de prueba (archivo .sql) Diagrama relacional de la base de datos (archivo .png) Requisitos y registro del tiempo de solución (Microsoft Excel 365)	§B.2.1
Gestión de Facturas	Respaldo de la base de datos (archivo .backup) Script estructura de la bdd y datos de prueba (archivo .sql) Diagrama relacional de la base de datos (archivo .png) Requisitos y registro del tiempo de solución (Microsoft Excel 365)	§B.2.2
Gestión de Notas	Respaldo de la base de datos (archivo .backup) Script estructura de la bdd y datos de prueba (archivo .sql) Diagrama relacional de la base de datos (archivo .png) Requisitos y registro del tiempo de solución (Microsoft Excel 365)	§B.2.3

Instrumentación de las tareas de análisis

Encuesta evaluativa: se utilizará la misma encuesta demográfica/diagnóstica, pero sin las preguntas demográficas. Se operativizará mediante un formulario de Microsoft

Forms 365 Online (Anexo §B.1.1).

Pruebas de aceptación, se utilizará una hoja de cálculo de Microsoft Excel 365 para registrar el resultado de las pruebas de aceptación que se realizarán a cada pregunta de las tareas experimentales de los participantes (Anexo §B.3.1).

Análisis estadístico, el análisis estadístico de los resultados del experimento se realizará en las herramientas IBM SPSS Statistics[12] v21.0, y Statgraphics[13] v16.1.03. Ver los scripts de cálculos en el Anexo §B.3.1.

6.2.9 Recolección de datos

En la Tabla §6.5 se muestra la agenda de planificación de actividades previstas para ejecutar el diseño del experimento presentado en la tabla §6.1.

Tabla 6.5: Agenda del experimento

Horario	Actividades	Tiempo (Min)
14H00-14H05	Indicaciones generales	5
14H05-14H10	Entrega del respaldo y script de la base de datos	5
14H10-14H25	Restauración de la base de datos	15
14H25-14H30	Entrega de requisitos - Sesión 1 (Excel y diagrama relacional)	5
14H30-16H30	SESIÓN 1	120
16H30-16H40	Recolección de tareas	10
16H40-16H45	Receso	5
16H45-16H50	Entrega del respaldo y script de la base de datos	5
16H45-16H50	Restauración de la base de datos	5
16H50-17H00	Entrega de requisitos - Sesión 2 (Excel y diagrama relacional)	5
17H00-19H00	SESIÓN 2	120
19H00-19H10	Recolección de tareas	10

Los experimentadores encargados de la ejecución del experimento, recolección del código fuente de las tareas de los participantes, y medición de la variable respuesta fueron (A. Quiña, C. Guevara, y F. Chulde).

Para la obtención de los elementos necesarios para las métricas de calidad interna, en uso, y la curva de aprendizaje, se lo hará con las siguientes actividades: i) información del documento Excel de requisitos, en donde los participantes registran el tiempo

[12] https://www.ibm.com/products/spss-statistics
[13] https://www.statgraphics.com

de solución de cada requisito, y evidencian con capturas de pantallas la ejecución de la solución. ii) Se cargará el código fuente de los participantes en un ambiente de desarrollo y luego se realizará las pruebas de aceptación de la ejecución de cada pregunta de las tareas experimentales, luego el resultado se lo registrará en una hoja de Excel de resultados.

6.2.10 Análisis

El análisis estadístico del experimento se realizará con las herramientas IBM SPSS Statistics, y Statgraphics. Para obtener el análisis de los datos se utilizará la media, desviación estándar, mínimo, máximo, rango, asimetría y curtosis estandarizada. Mientras que, para comprobar la hipótesis se utilizará el modelo lineal de efectos mixtos, el cual, es propuesto y validado en estudios similares [30, 83].

6.3 Resumen

En este capítulo, se ha presentado un diseño experimental que tiene como objetivo analizar el proceso de desarrollo del software, con el propósito de comparar el efecto de los paradigmas de programación GraphQL y REST con respecto a la calidad del software, desde un punto de vista de los investigadores en un contexto académico. En este sentido, se ha elaborado una serie de artefactos necesarios para operativizar las fases de formación, ejecución, y análisis del experimento; que están descritos con más detalle en el Anexo §B.

El experimento diseñado en el presente capítulo, se validará en el Capítulo §9, en donde se mostrará los resultados de su ejecución, el análisis, y conclusiones de dichos resultados.

Efectos de las arquitecturas híbridas: Diseño Experimental

El aprendizaje es hijo de la repetición

Robin Sharma (1965 –),
Profesor, escritor y orador nepalés

En este capítulo, se propone un diseño experimental para analizar el proceso de desarrollo del software, con el propósito de comparar los efectos del uso de una arquitectura híbrida REST/GraphQL frente a una arquitectura REST con respecto a la calidad del software, desde un punto de vista de los investigadores en un contexto controlado de un laboratorio computacional.

En la sección §7.1, se destaca una breve discusión de antecedentes y motivaciones que impulsan el desarrollo de la propuesta del presente capítulo. En la sección §7.2, se detalla el diseño del entorno experimental compuesto por los siguientes elementos: Objetivo del experimento; Factores y tratamientos; Variables de investigación; Hipótesis; Diseño; Casos de uso; Tareas experimentales; Instrumentación; Recolección de datos; y Análisis. Por último, la sección §7.3 resume el capítulo.

7.1 INTRODUCCIÓN

La motivación de este capítulo es contestar la pregunta de investigación **PI**$_3$: ¿En qué escenarios resultan adecuadas las arquitecturas híbridas REST/GraphQL?, la misma que se deriva y concreta en las siguientes sub preguntas:

- **PI**$_{3.1}$: ¿Cuál es el efecto del desarrollo de una arquitectura híbrida REST/GraphQL en la calidad del software?.
- **PI**$_{3.2}$: ¿En qué escenarios es más adecuado usar las arquitecturas híbridas REST/GraphQL?.

Para lo cual, se propone realizar un experimento que compare la eficiencia del rendimiento de una arquitectura híbrida REST/GraphQL frente a una arquitectura REST, con respecto a la calidad del software para analizar los efectos del uso de las arquitecturas híbridas REST/GraphQL en el proceso de desarrollo de software. En donde, se caracteriza los escenarios de uso mediante el diseño de un laboratorio computacional que contenga diversos casos del uso de las arquitecturas. Para medir los efectos del uso de las arquitecturas se utiliza varias métricas de la característica de calidad *Eficiencia del Rendimiento* del software, establecidas en las normas ISO/IEC 25010 [75] e ISO/IEC 25022 [76].

7.2 ENTORNO EXPERIMENTAL

7.2.1 Objetivo del experimento

Para definir el objetivo de este experimento, se utilizó el enfoque GQM [6], en donde, se consideró:

- **Analizar** el proceso de desarrollo del software
- **Con el propósito de** comparar la eficiencia del rendimiento de las arquitecturas híbridas de software
- **Con respecto** a la calidad del software
- **Desde el punto de vista** del investigador
- **En el contexto** controlado de laboratorio

7.2.2 Factores y tratamientos

El factor que se investiga es la *arquitectura de software*, concretamente en el desarrollo de APIs.

Los tratamientos que se aplican a este factor son:

- Arquitectura híbrida REST/GraphQL para el desarrollo de APIs.
- Arquitectura REST para el desarrollo de APIs (considerada como el nivel de control).

Para el desarrollo de la arquitectura híbrida REST/GraphQL, se propone implementar un API GraphQL que envuelva al API REST del gestor de referencias Mendeley[1]. Y como arquitectura REST se propone utilizar directamente el API REST de Mendeley.

7.2.3 Variables

Por un lado, se definió como *variable independiente* a la "Arquitectura de software para el desarrollo de APIs", aplicado con dos tratamientos: arquitectura híbrida REST/GraphQL y arquitectura REST. Esta variable se operativiza principalmente en el desarrollo de una API GraphQL que envuelva y utilice al API REST de Mendeley como origen de datos; la cual, luego se usará para comparar las arquitecturas en un laboratorio computacional.

Por otro lado, se definió como *variable dependiente* a la "Calidad del Software" [74], que se operativiza utilizando la característica "Eficiencia de Rendimiento" de la calidad externa del software, la que se describe en la norma ISO/IEC 25010 como la evaluación del rendimiento relativo a la cantidad de recursos usados bajo condiciones establecidas [75]. A continuación, se detallan las métricas escogidas para cuantificar la variable dependiente:

- *Tiempo medio de respuesta*, es el tiempo que se tarda en completar un trabajo o un proceso asíncrono [76]. La función de medición es:

[1] https://www.mendeley.com/

$$X = \sum_{I=1}^{n} (B_I = A_I)/n \tag{7.1}$$

Donde, A_I es el tiempo para iniciar el trabajo I; B_I es el tiempo para completar el trabajo I; y n = número de mediciones [76].

- *Tamaño de la respuesta*, es el tamaño del archivo de respuesta (formato JSON) de una consulta realizada a una API. La unidad de medida son los bytes.

7.2.4 Hipótesis

Basándose en la pregunta de investigación **PI**$_{3.1}$, se ha definido las siguientes *hipótesis* para el experimento:

- H_0 (*Hipótesis nula*): No hay diferencia en los efectos producidos por aplicar arquitecturas híbridas REST/GraphQL en la calidad del software.
- H_1 (*Hipótesis alternativa 1*): Existe diferencia en los efectos producidos por aplicar arquitecturas híbridas en la calidad del software. La calidad del software que produce las arquitecturas híbridas REST/GraphQL es superior a la calidad que produce la arquitectura REST.
- H_2 (*Hipótesis alternativa 2*): Existe diferencia en los efectos producidos por aplicar arquitecturas híbridas en la calidad del software. La calidad del software que produce la arquitectura REST es superior a la calidad que produce las arquitecturas híbridas REST/GraphQL.

7.2.5 Diseño

El diseño del experimento tiene como propósito establecer las condiciones necesarias para comparar la eficiencia de la arquitectura híbrida REST/GraphQL con la arquitectura REST mediante la construcción de un laboratorio computacional. Para lo cual, se estableció dos tareas experimentales que comparten escenarios comunes de consulta de datos. La primera tarea consumirá datos de la API REST, y la otra tarea consumirá datos de la API GraphQL (arquitectura híbrida REST/GraphQL).

Las tareas experimentales serán implementadas mediante aplicaciones que ejecuten varios casos de uso de consultas de datos comunes para las API REST y la API

GraphQL. Se definen seis casos de uso con tres diferentes niveles de complejidad, las que se ejecutarán tres veces con diferentes cantidades de datos (registros) para comprobar la eficiencia de las APIs en diferentes escenarios. Para medir la eficiencia se utilizará las métricas establecidas en la sección §7.2.3, que capturan el tiempo y peso de las respuestas de la ejecución de cada caso de uso. A continuación, en la Tabla §7.1 se muestra el diseño del experimento.

Tabla 7.1: Diseño del Experimento

Casos de uso	Repeticiones	REST	REST/GraphQL
Caso de uso 1	3	Registros: 1, 10, 100	Registros: 1, 10, 100
Caso de uso 2	3	Registros: 1, 10, 100	Registros: 1, 10, 100
Caso de uso 3	3	Registros: 1, 10, 100	Registros: 1, 10, 100
Caso de uso 4	3	Registros: 1, 10, 100	Registros: 1, 10, 100
Caso de uso 5	3	Registros: 1, 10, 25	Registros: 1, 10, 25
Caso de uso 6	3	Registros: 1, 10, 25	Registros: 1, 10, 25

7.2.6 Casos de uso

En esta sección, se describen los casos de uso que contienen las consultas de datos que se realizarán a la API REST de Mendeley. Se estableció seis casos de uso con tres diferentes niveles de complejidad, en donde, la complejidad (nivel de consulta) se mide por el número de endpoints que se necesita acceder para obtener el requerimiento de la consulta de datos:

Caso de uso 1: Consulta de documentos:

- Nivel de consulta: 1
- Endpoint: https://api.mendeley.com/documents
- Campos: id, title, type, profile_id, group_id, created, last_modified, abstract, source, year, authors(first_name, last_name), identifiers(doi)

Caso de uso 2: Consulta de carpetas:

- Nivel de consulta: 1
- Endpoint: https://api.mendeley.com/folders
- Campos: id, name, created, modified, parent_id

Caso de uso 3: Consulta de documentos por carpetas:

- Nivel de consulta: 2
- Endpoints:
  - https://api.mendeley.com/folders
  - https://api.mendeley.com/documents
- Campos: Folder: id, name, created, parent_id; Document: id, title, type, abstract, year, authors(first_name, last_name)

Caso de uso 4: Consulta de archivos por documentos:

- Nivel de consulta: 2
- Endpoints:
  - https://api.mendeley.com/documents
  - https://api.mendeley.com/files
- Campos: Document: id, title, type, abstract, year, authors(first_name, last_name); File: id, file_name, size, document_id

Caso de uso 5: Consulta de documentos por carpeta y por grupo:

- Nivel de consulta: 3
- Endpoints:
  - https://api.mendeley.com/groups/v2
  - https://api.mendeley.com/folders
  - https://api.mendeley.com/documents
- Campos: Group: id, name, access_level; Folder: id, name, created, parent_id; Document: id, title, type, abstract, authors(first_name); File: id, file_name, size

Caso de uso 6: Consulta de archivos por documento y por carpeta:

- Nivel de consulta: 3
- Endpoints:
  - https://api.mendeley.com/folders
  - https://api.mendeley.com/documents
  - https://api.mendeley.com/files
- Campos: Folder: id, name, created, parent_id; Document: id, title, type, abstract, authors(first_name, last_name); File: id, file_name, size

7.2.7 Tareas experimentales

En esta sección, se diseñó un laboratorio computacional con dos tareas experimentales de acuerdo con el diseño, tratamientos y casos de usos definidos para el experimento.

Tarea experimental 1: Consulta a la API REST de Mendeley

El objetivo de esta tarea es ejecutar los casos de uso de consultas de datos (ver sección §7.2.6) en la API REST de Mendeley. Para lo cual, se debe desarrollar una aplicación cliente (Node JS) que implemente los casos de uso de consulta de datos a la API REST de Mendeley[2] disponible en la nube, de acuerdo al diseño del experimento establecido en la tabla §7.1. Luego, la aplicación deberá almacenar las respuestas de las consultas en archivos con extensión JSON. La aplicación se ejecutará desde una PC-local, tal como se muestra en la arquitectura del laboratorio experimental (Figura §7.1).

Tarea experimental 2: Consulta a la API GraphQL (envoltorio)

El objetivo de esta tarea es ejecutar los mismos casos de uso de la tarea experimental 1, pero utilizando el lenguaje de consulta de GraphQL. Para lo cual, se debe desarrollar dos aplicaciones: la primera aplicación debe ser una API GraphQL que proporcione el servicio de datos de Mendeley; y la otra debe ser una aplicación que consuma mediante lenguaje GraphQL los datos expuestos por la primera API. En este sentido, se propone desarrollar la primera aplicación con una arquitectura híbrida REST/GraphQL, en donde, se implemente una API GraphQL que envuelva al API REST de Mendeley mediante la caracterización de su esquema de datos en GraphQL, y se utilice como origen de datos al API REST de Mendeley. Para la segunda aplicación, se propone utilizar el mismo cliente de la tarea experimental 1, pero que ejecute los casos de uso de consultas a la API GraphQL y que de igual manera almacene las respuestas de las consultas en archivos con extensión JSON, tal como se muestra en la arquitectura del laboratorio experimental propuesto en la Figura §7.1.

7.2.8 Instrumentación

En esta sección se lista la instrumentación como infraestructura, tecnología, y librerías que componen el laboratorio computacional:

[2] https://dev.mendeley.com/

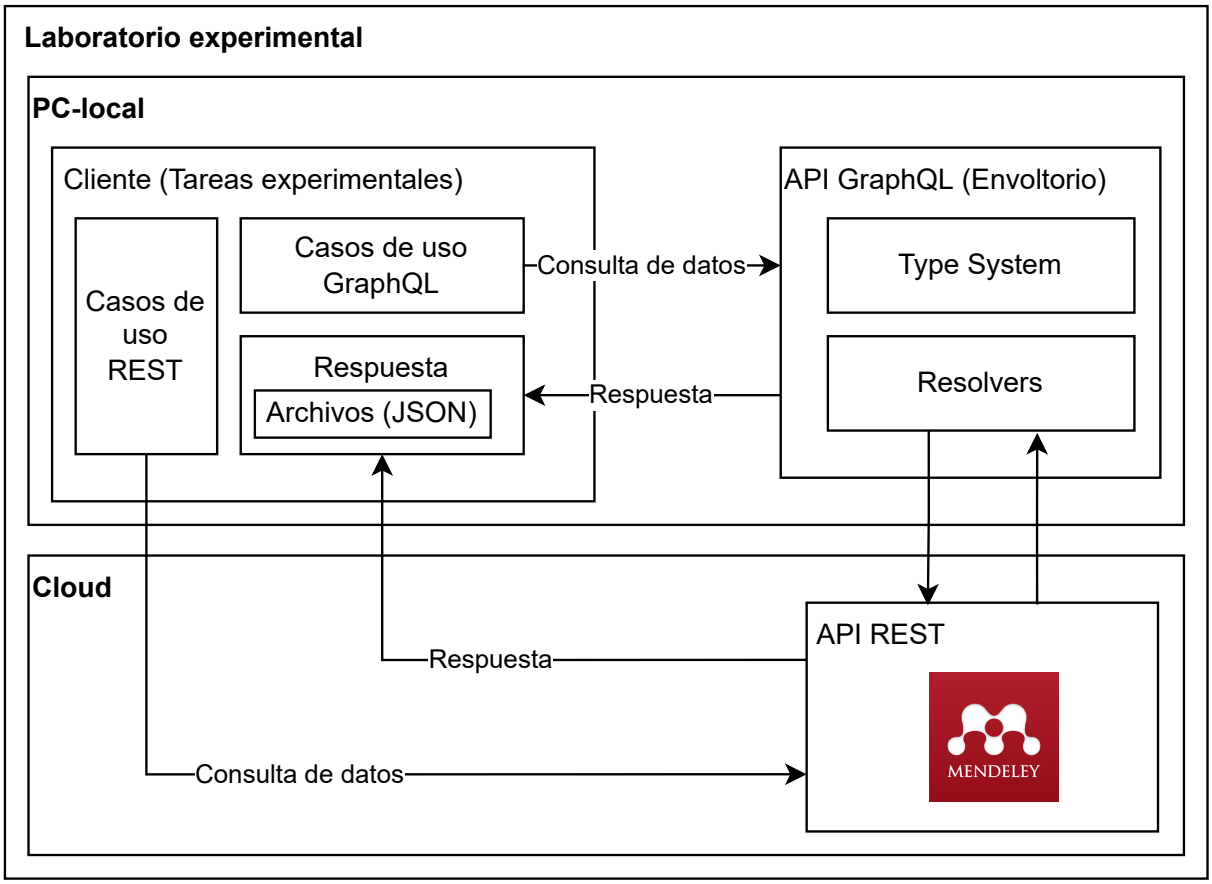


Figura 7.1: Arquitectura del laboratorio experimental

Especificaciones de la PC-local, es una computadora que tiene las siguientes características:

- Sistema Operativo: Windows 10 Pro 64-bit
- Procesador: Intel (R) Core (TM) i7-3110M CPU CPU @ 2.40GHz 2.40 GHz
- Memoria: RAM 8 GB
- Conexión de internet de 75 MB de ancho de banda.

Ambiente de desarrollo, se conforma de las siguientes aplicaciones y tecnologías:

- Ambiente de desarrollo integrado IDE: Visual Studio Code v1.62
- Lenguaje de programación: JavaScript
- Entorno de tiempo de ejecución de JavaScript: NodeJS v14.17.5
- Librerías npm de la aplicación cliente: react 16.8.6, react-apollo 2.5.5

- Librerías npm para el API GraphQL: graphql v15.5.0, graphql-server-express v1.4.1, graphql-import v1.0.2, y graphql-tools v7.0.5, cors 2.8.5
- Aplicación cliente para el consumo del API REST: Postman v8.12.2
- Aplicación cliente para el consumo del API GraphQL: GraphiQL

Recolección y análisis de datos, se conforma de las siguientes aplicaciones:

- Microsoft Excel 365
- IBM SPSS v21.0
- Statgraphics v16.1.03

7.2.9 Recolección de datos

En la Tabla §7.2 se muestra la estructura de los datos que tendrá el archivo Microsoft Excel, donde se registrará los resultados de la ejecución del experimento diseñado en tabla §7.1.

Tabla 7.2: Estructura de la recolección de datos

Campo	Descripción
Nro.	Número de la muestra (auto-numérico)
Caso de uso	Caso de uso que se ejecuta
Nivel	Nivel de complejidad de la consulta del caso de uso
Nro. Registros	Cantidad de registros consultados en el caso de uso
Repetición	Número de repetición de ejecución (valores entre 1 y 3)
Paradigma	REST, o GraphQL (representa la arquitectura híbrida REST/GraphQL)
Tiempo	Tiempo de respuesta de la ejecución del caso de uso medido en milisegundos
Peso	Tamaño del archivo de respuesta medido en bytes

7.2.10 Análisis

El análisis estadístico del experimento se realizará con las herramientas IBM SPSS Statistics, y Statgraphics. Para realizar las pruebas de normalidad de los datos, en donde, se utilizará la desviación estándar, coeficiente de variación, mínimo, máximo, rango, sesgo estandarizado, y curtosis estandarizada. Mientras que, para comprobar la hipótesis se utilizará el modelo lineal mixto.

7.3 RESUMEN

En este capítulo, se presentó el diseño experimental que tiene como objetivo analizar el proceso de desarrollo del software, con el propósito de comparar la eficiencia de las arquitecturas híbridas REST/GraphQL con respecto a la calidad del software, desde un punto de vista de los investigadores en un contexto controlado de un laboratorio computacional. En este sentido, el experimento se diseñó y documentó basado en la guía de Wohlin *et al.* [186] con la intención de ser lo más metodológicos posibles para aportar una mayor validez a los resultados obtenidos.

El experimento diseñado en este capítulo se validará en el capítulo §10, en donde se ejecutará y mostrará el análisis de los resultados, conclusiones y trabajos futuros.

Acuerdos de nivel de servicio para APIs GraphQL

La innovación distingue a los líderes de los seguidores.

Steve Jobs (1955 – 2011),
Empresario y magnate estadounidense

En este capítulo, se presenta una propuesta para modelar y operativizar acuerdos de nivel de servicio (SLA, por sus siglas en inglés de Service Level Agreements) para APIs GraphQL. Además, como parte de la propuesta se propone una herramienta para validar y registrar SLA Planes de proveedores de APIs GraphQL.

En la sección §8.1, se destaca una breve discusión de los antecedentes y motivación que impulsan el desarrollo de la propuesta.En la sección §8.2, se establece la especificación de SLA para APIs GraphQL que contiene: i) La especificación y esquema del documento de acuerdos de nivel de servicios (SLA); ii) Ciclo de vida del desarrollo de APIs GraphQL basado en SLA; iii) Procesos generales para modelar y operativizar los documentos SLA. La sección §8.3 se propone la herramienta SLA GraphQL Manager para registrar SLA Planes de proveedores de APIs GraphQL, y registrar clientes a los SLA Planes ofertados por los proveedores. Por último, la sección §8.4 resume el capítulo.

8.1 Introducción

La motivación de este capítulo es contestar la pregunta de investigación **PI**$_4$: ¿En qué medida es posible modelar los acuerdos de APIs GraphQL?.

Para lo cual, se propone un estándar de código abierto llamado ***SLA4GraphQL***, que sirve para describir SLA para APIs GraphQL, la que a su vez se considera como la principal contribución de la presente memoria. La especificación del estándar describe cómo modelar y operativizar acuerdos de nivel de servicio (SLA) para APIs GraphQL, que se podrá utilizar como instrumentación en la gobernanza de arquitecturas híbridas orientadas a microservicios. El objetivo de *SLA4GraphQL*, es presentar un estándar que establezca un contrato entre el proveedor de una API GraphQL y el cliente de consumo de la API, en donde, se establece explícitamente los términos, limitaciones, y planes del servicio de la API.

En primer lugar, se toma como base la propuesta *SLA4OAI* que promueve la especificación abierta para acuerdos de nivel de servicios (SLA) sobre las API REST definidas a través de la especificación OpenAPI [46, 71] revisada en el capítulo §4; a la cual, se propone complementar las especificaciones de modelado necesarias que cubran las particularidades del uso y consumo de acuerdos de nivel de servicios (SLA) para APIs GraphQL. Luego se establece un ciclo de vida para el desarrollo de APIs GraphQL basadas en SLA, y la descripción de los principales procesos para operativizar los SLA. Además, se complementa la propuesta con una herramienta que valida el modelado de acuerdos y operativiza el registro de los planes SLA de proveedores, y el registro e instanciación de los planes para clientes de la API GraphQL.

La propuesta *SLA4GraphQL*, se validará en el capítulo §11.

8.2 Especificación de SLA para APIs GraphQL

En esta sección, se propone un estándar de código abierto llamado **SLA4GraphQL** que se basa en el estándar SLA4OAI [46, 71] que especifica como describir acuerdos de nivel de servicios SLA para APIs GraphQL, el cual, se conforma por: la especificación del esquema del documento SLA (SLA Plan), un ciclo de vida para desarrollo de APIs, procesos de modelado y operativización de SLA, y una herramienta para registrar y validar los SLA modelados. La descripción de SLA es neutral de la tecnología del proveedor de la API GraphQL, con la intención de fomentar la innovación en la do-

cumentación y operativización de APIs. Los proveedores que exponen APIs GraphQL podrán importar por medio de sus herramientas tecnológicas los componentes y métricas claves de los SLA, para implementar las limitaciones establecidas en ellos de una manera estandarizada.

8.2.1 Especificación del documento SLA Plan

El documento SLA o también mencionado en la memoria como SLA, o SLA Plan, contiene los acuerdos de nivel de servicio para una API GraphQL descrito en un archivo de extensión (.yaml o .yml) con formato de serialización de datos YALM[1] . El esquema del documento SLA, está basado en la estructura propuesta en *SLA4OAI* [71].

La Tabla §8.1 muestra la descripción del objeto SLA, que contiene la estructura de los principales componentes del documento SLA para la propuesta ***SLA4GraphQL***, y al mismo tiempo en la columna "SLA4OAI" de la tabla se muestra la comparación de los componentes que tienen el mismo comportamiento y descripción que la especificación *SLA4OAI*; en donde, se establece con el valor de "Si" a los componentes que tienen la misma especificación y comportamiento entre las propuestas SLA4GraphQL y SLA4OAI.

Tabla 8.1: Objeto SLA

Comienzo de tabla			
Componente	Tipo	Descripción	SLA4OAI
Contexto	Objecto Contexto	*Obligatorio* Contiene la información principal del contexto SLA.	Si
Infraestructura	Objecto Infrastructura	*Obligatorio* Proporciona información sobre las herramientas utilizadas para la gestión del SLA como almacenamiento, cálculo, o gobernanza.	Si
Fijación de precios	Objecto Precios	*Opcional*. Describe la información general de los precios de la API.	Si
Métricas	Objecto Métricas	*Obligatorio* Una lista de métricas para usar en el contexto del SLA.	No
Planes	Objecto Planes	*Opcional* Un conjunto de planes para definir diferentes niveles de servicio por plan.	No

[1] https://yaml.org/

Continuación de la Tabla §8.1

Componente	Tipo	Descripción	SLA4OAI
Cuotas	Objecto Cuotas	*Opcional* Cuotas globales, estas son las cuotas predeterminadas, pero cada plan podría anularlas más adelante.	No
Tarifas	Objecto Tarifas	*Opcional* Tarifas globales, estas son las tarifas predeterminadas, pero podrían ser anuladas por cada plan más adelante.	No
Garantías	Objecto Garantías	*Opcional* Garantías globales, estas son las garantías predeterminadas, pero podrían ser anuladas por cada plan más adelante.	Si
Configuraciones	Objecto Configuraciones	*Opcional* Defina las configuraciones predeterminadas, luego cada plan puede ser anulado.	Si

Fin de tabla

En este sentido, en esta sección para los componentes que tienen el mismo comportamiento y descripción que el estándar SLA4OAI, no se describirán sus especificaciones y se asumirán las especificaciones desplegadas por SLA4OAI [71] en `https://sla4oai.specs.governify.io/Specification.html` y en el capítulo §4; los componentes que *no se describirán* son: contexto, infraestructura, fijación de precios, métricas, garantías, y configuración.

Por el contrario, en esta sección se profundizará las especificaciones de los componentes: planes, cuotas, y tarifas; que justamente especifican las particularidades del manejo para las APIs GraphQL.

Objeto de cuotas (quotasObject)

Este componente se estructura por un conjunto de objetos operación (operationObject) que contiene las operaciones disponibles del servicio de la API GraphQL proveedor. La finalidad de este objeto es describir las cuotas de las operaciones específicas que proporcionará el servicio de la API. La Tabla §8.2 muestra la estructura del objeto cuotas.

Tabla 8.2: Objeto Cuotas (quotasObject)

Patrón de campo	Tipo	Descripción
{nombreOperación}	Objecto Operación	*Opcional* Describe las operaciones del API GraphQL.

Objeto Operación (operationObject): contiene dos de las tres operaciones dispo-

nibles del servicio de la API GraphQL proveedor: consultas (queries) y mutaciones (mutations), en este punto, es importante aclarar que la operación suscripciones (subscriptions) se implementará en una siguiente versión de SLA4GraphQL. La Tabla §8.3 muestra la estructura del objeto operación.

Tabla 8.3: Objeto Operación (operationObject)

Patrón de campo	Tipo	Descripción
{nombreRuta}	Objecto Ruta	*Opcional* Son las rutas de los recursos consulta o mutación del API GraphQL.

Objeto Ruta (pathObject): muestra la ruta de las consultas o mutaciones de la API GraphQL utilizando expresiones de rutas del lenguaje *XPath*[2]. La Tabla §8.4 muestra la estructura del Objeto Ruta.

Tabla 8.4: Cuota: Objeto Ruta (pathObject)

Patrón de campo	Tipo	Descripción
{nombreMétrica}	Objecto Límite	*Opcional* Son los límites de la métrica en la operación query/mutation de la API GraphQL.

En este aspecto, es necesario aclarar que GraphQL expone sus recursos (operaciones) a través de una única ruta (endpoint), por tal motivo, para obtener y validar las rutas específicas de las operaciones se utilizó las consultas de introspección al esquema de la API. Las respuestas de las consultas de introspección contienen en datos el esquema de la API, la que puede ser leída de una manera recursiva y/o referencial de objetos, por lo cual, es necesario realizar un tratamiento en la respuesta de la consulta para obtener el esquema de la API y seleccionar sus recursos (operaciones) de una manera más simple. Por la naturaleza de los datos, en el tratamiento se estructuró el esquema del API en un documento de mapa de nodos jerárquico en formato XML para poder navegar y consultar sus objetos principales como operaciones, parámetros de operaciones y campos de respuesta de las operaciones. En este sentido, se analizó varias opciones para cubrir estos requisitos; y una de las opciones más viables fue utilizar el lenguaje XPath para buscar y seleccionar recursos en una estructura jerárquica de documentos XML. Por consiguiente, se estructuró el esquema del API GraphQL en un documento XML, para ser consultado mediante expresiones de rutas con el lenguaje XPath. Con el ánimo de ilustrar esta sección, en la Figura §8.1 se muestra a) un ejemplo de una consulta de introspección del esquema de API GraphQL; b) un ejemplo

[2] https://www.w3schools.com/xml/xpath_intro.asp

de la respuesta de la consulta de introspección; c) un ejemplo de transformación de la respuesta a formato XML; y d) un ejemplo de una expresión regular para escoger el elemento "product" de la consulta "InStore" del esquema de una API GraphQL.

a)

```
query query_schema{
  __schema {
  queryType {
    name
    fields {
      name
      args{name}
      type {
        fields {
          name
        }
. . . .
```

b)

```
{ "data": {
  "__schema": {
    "queryType": {
      "name": "Query",
      "fields": [
        { "name": "inStore",
          "args": [{ "name": "key"    }],
          "type": {
            "fields": [{ "name": "me"       },
                       { "name": "product"  },
                       { "name": "customer" },
. . . .                { "name": "cart"     }]
```

c)

```
<query>
  <inStore key="">
    <me>me</me>
    <product>product</product>
    <customer>customer</customer>
    <cart>cart</cart>
  </inStore>
</query>
```

d)

```
//inStore/product
```

Figura 8.1: Ejemplo del proceso de Objeto Ruta

Objeto Límite (limitObject): determina los límites permitidos de las métricas aplicadas a la ruta de consulta o mutación de la API. Los nombres de las métricas se pueden definir de dos maneras: por un lado, se puede utilizar las métricas definidas en el Objeto Métricas, o por otro lado se puede hacer referencia a las funciones XPath[3]; en este caso, se deberá poner el nombre de la función XPath antecedido por el símbolo "_", por ejemplo: "_count". Las funciones XPath son: boolean, ceiling, choose, concat, contains, count, current, document, element-available, false, floor, format-number, function-available, generate-id, id, key, lang, last, local-name, name, namespace-uri, normalize-space, not, number, position, round, starts-with, string, string-length, substring, substring-afteri, substring-before, sum, system-property, translate, true, unparsed-entity-url.

[3] https://developer.mozilla.org/en-US/docs/Web/XPath/Functions

La Tabla §8.5 muestra la estructura del objeto límite:

Tabla 8.5: Cuota: Objeto Límite (limitObject)

Patrón de campo	Tipo	Descripción
max	número	*Opcional* Valor máximo, que se puede aceptar.
min	número	*Opcional* Valor mínimo, que se puede aceptar.
over	texto	*Requerido* Define cuando se validará la limitación establecida en la métrica. Los posibles valores son: request (validará la métrica en la solicitud), response (validará la métrica en la respuesta), model (validará la métrica en el modelo de datos).
description	texto	*Requerido* Describe el objeto métrica.

La Figura §8.2 muestra un ejemplo de la estructura del objeto tarifas, en donde, se asume la definición de la métrica ”nodes”. El ejemplo intenta definir que el límite máximo de nodos que se puede consultar a ”pets” (query) es de 500 nodos.

```
quotas:
  queries:
    //pets:
      nodes:
        max: 500
        over: request
        description: "Cantidad de nodos solicitados"
```

Figura 8.2: Ejemplo del objeto cuotas del SLA

Objeto de tarifa (Rates)

La tarifa contiene un conjunto de operationObject que describen los límites y frecuencias de los recursos (consultas y mutaciones) del servicio de la API en el plan actual. La Tabla §8.6 muestra la estructura del objeto tarifa.

Tabla 8.6: Objeto Tarifa (rateObject)

Patrón de campo	Tipo	Descripción
{nombreOperación}	Objecto Operación	*Opcional* Son las operaciones del API GraphQL.

Objeto Operación (operationObject): Son las operaciones de la API GraphQL, los posibles valores son: consultas (queries) o mutaciones (mutations). Tiene la misma estructura del operationObject de cuotas (ver tabla §8.3).

Objeto Ruta (pathObject): utiliza expresiones de rutas de XPath para especificar una consulta o mutación a partir del esquema de la API GraphQL. Este objeto tiene

el mismo comportamiento y estructura especificado en el pathObject de Cuotas (ver sección §8.2.1 y tabla §8.4).

Objeto Límite (limitObject): Especifica los límites de la métrica establecida para la consulta o mutación. En este sentido, al igual que la métrica de Cuotas, se puede especificar de dos maneras: métricas definidas en metricObject, o la referencia a las funciones de XPath.

La tabla §8.7 muestra la estructura del objeto límite:

Tabla 8.7: Tarifa: Objeto Límite (limitObject)

Patrón de objeto	Tipo	Descripción
max	Número	*Opcional* Valor máximo, que se puede aceptar.
over	texto	*Requerido* Define cuando se validará la limitación establecida en la métrica. Los posibles valores son: request (validará la métrica en la solicitud), response (validará la métrica en la respuesta), model (validará la métrica en el modelo de datos).
period	Texto	*Opcional* Es el período del objetivo de la métrica, puede tener los valores de: por segundo (secondly), por minuto (minutely), por hora (hourly), por día (daily), por mes (monthly) o por año (yearly).
description	Texto	*Requerido* Describe el objetivo de la métrica.

La Figura §8.3 muestra un ejemplo de la estructura del objeto tarifas, en donde, se asume la definición de la métrica ”nodes”. El ejemplo intenta definir que el límite máximo de nodos que se puede consultar a ”pets” (query) es de 10000 nodos por mes.

```yaml
rates:
  queries:
    //pets:
      nodes:
        max: 10000
        over: request
        period: monthly
        description: "Cantidad de nodos solicitados"
```

Figura 8.3: Ejemplo del objeto tarifas del SLA

8.2.2 Ciclo de vida del desarrollo de APIs GraphQL basado en SLA

Tomando como base la propuesta de *SLA4OAI* [46], se identificó un conjunto mínimo de actores, roles, y actividades generales inmersas en el ciclo de vida de desarrollo

de APIs GraphQL basada en acuerdos de nivel de servicio, tal como se presenta en la Figura §8.4.

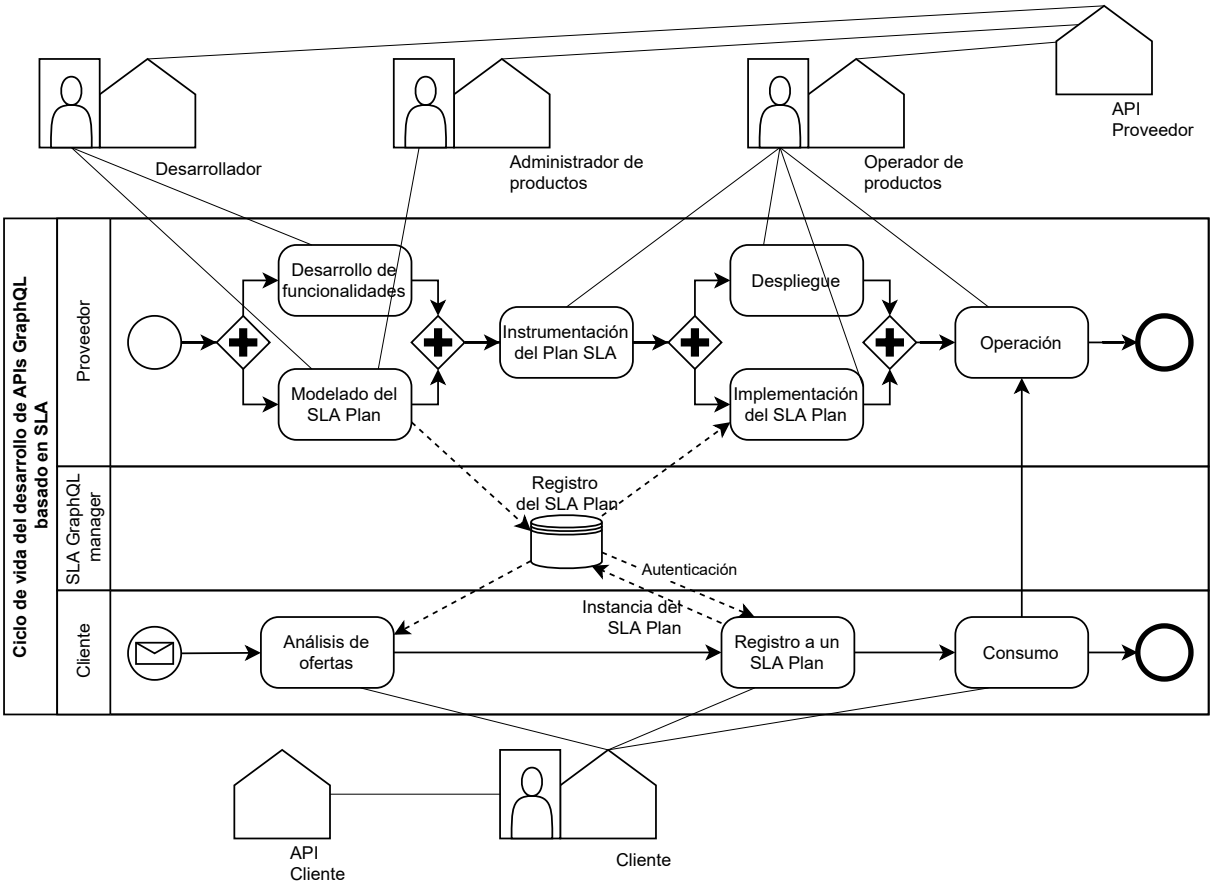


Figura 8.4: Ciclo de vida de desarrollo de APIs GraphQL basado en SLA

Se comienza a describir el ciclo de vida mediante la identificación de los principales autores que interactúan en el desarrollo de APIs basadas en SLA: proveedor y cliente de la API, y un administrador del SLA (Herramienta: *SLA GraphQL Manager*, la cual se describe a fondo en la sección §8.3). Luego, se analiza la figura §8.4 vista desde una perspectiva de proveedor, en donde, se identifica como primera actividad al *desarrollo de funcionalidades* de la API proveedor que implementa los requisitos funcionales que brindará la API. De manera paralela a esta actividad, el proveedor debe realizar el *modelado del SLA plan* o los SLA Planes que contendrán la configuración y limitaciones de la oferta de operación de la API, la cual se debe registrar mediante la herramienta *SLA GraphQL Manager* que valida y almacena el archivo del SLA Plan en un repositorio. Una vez concluidas estas actividades, se debe realizar la *instrumentación del SLA Plan*, en donde se parametrizan las herramientas y/o artefactos desarrollados para que se

comporten de acuerdo a las limitaciones descritas en el SLA plan, y que a su vez debe proporcionar las métricas necesarias para analizar el estado del SLA Plan. Luego, se realiza el despliegue de la API Proveedor, y al mismo tiempo se debe configurar la infraestructura de la implementación del SLA Plan en la API proveedor para que se pueda cumplir las limitaciones establecidas, antes que la API realice la actividad de *Operación* de las solicitudes realizadas por los clientes.

Luego que el proveedor despliegue el servicio de su API y haya registrado su(s) SLA Plan(es) como ofertas de operación; se complementa la visión del ciclo de vida desde la perspectiva del cliente del API proveedor. En este sentido, las actividades del cliente empieza por *analizar las ofertas* registradas por los proveedores, en donde debe escoger una oferta y debe realizar el *registro a un SLA Plan* mediante la herramienta *SLA GraphQL Manager*, la cual, registrará al cliente y el SLA escogido en un repositorio en la nube, el que a su vez solicita al proveedor la autenticación del cliente a su API; como consecuencia de esta actividad, la herramienta proporciona al cliente un token que contiene la configuración y autenticación de acceso a la API proveedor; para que finalmente el cliente pueda realizar el *Consumo* del API proveedor de acuerdo a las limitaciones establecidas en las cuotas y tarifas del SLA que se registró.

Para la implementación de este ciclo de vida se debe tener en cuenta los siguientes aspectos: i) La instrumentación del SLA, implantación del SLA, y la operación de la API, deben cumplir las limitaciones descritas en las secciones de cuotas y tarifas descritas en los SLA Planes ofertados por el proveedor. ii) El API proveedor debe proporcionar las funciones necesarias de registro y autenticación de clientes de la API. iii) El API proveedor debe proporcionar las métricas necesarias para recoger la información del consumo de los clientes del API proveedor. En este sentido, para la implementación también se debe tomar en cuenta al ciclo de vida vista desde una perspectiva industrial, en donde, se identifica para el actor proveedor los siguientes roles: Desarrollador, Administrador de productos, Operador de productos, y el API proveedor. Y por parte del actor cliente se identifican los roles: Cliente, y API Cliente. Los roles antes mencionados están descritos en la sección §4.3.1

Cabe mencionar, que el ciclo de vida propuesto se podría abordar con ligeras diferencias dependiendo del contexto de su aplicación, o debido a que este proceso puede evolucionar para añadir pasos intermedios (como pruebas) o para incluir ciclos evolutivos para el despliegue progresivo de versiones.

8.2.3 Procesos para modelar y operativizar el documento SLA Plan

En esta sección, se describen los procesos generales para el modelado y operativización del documento *SLA Plan*. Tomando como base el ciclo vida de desarrollo de APIs GraphQL basado en SLA (ver figura §8.4), se considera que las actividades de *Modelado del SLA Plan*, *Registro del SLA Plan* y *Operación* son hasta cierto punto generales para su implementación, las mismas se describen a continuación.

Procesos para el Modelado y Registro del SLA Plan

Esta actividad tiene como objetivo registrar un SLA Plan creado por un proveedor. Primeramente, se realiza la actividad de *Modelado del SLA Plan*, la cual se describe con detalle en la sección §8.2.1 para crear el documento SLA Plan. Luego que el proveedor ha creado el SLA, lo debe registrar mediante la herramienta *SLA GraphQL Manager*, esta herramienta valida el SLA antes del registro. Para la validación, se toma como entrada por un lado el documento SLA, y por otro lado un documento pre-establecido que contiene el esquema SLA con el cual se validará el esquema del documento SLA. Luego de manera simultánea se valida los componentes de Cuotas/Tarifas del SLA Plan, y la validación del esquema del documento SLA Plan, con el documento SLA esquema. Si el resultado de la validación es igual a ”pasa”, se procede a registrar el SLA Plan, en una colección de SLA Planes de la herramienta SLA GraphQL Manager. Y si el resultado de la validación es igual a ”error”, se envía el mensaje de error como resultado, tal como se muestra en la Figura §8.5.

La Figura §8.6 muestra el sub-proceso de validación de cuotas/tarifas del SLA Plan, que comienza por un lado realizando consultas de introspección del esquema del API proveedor, y convirtiendo ese resultado en un archivo jerárquico de nodos de tipo XML. Por otro lado, se toma el documento SLA Plan para realizar dos validaciones simultáneas, una validación de las expresiones de las rutas de las operaciones consultas/mutaciones con lenguaje XPath en el esquema del API, y luego la validación de las métricas de las cuotas/tarifas con las métricas definidas en el SLA o con las funciones XPath tal como se explica en la sección §8.2.1.

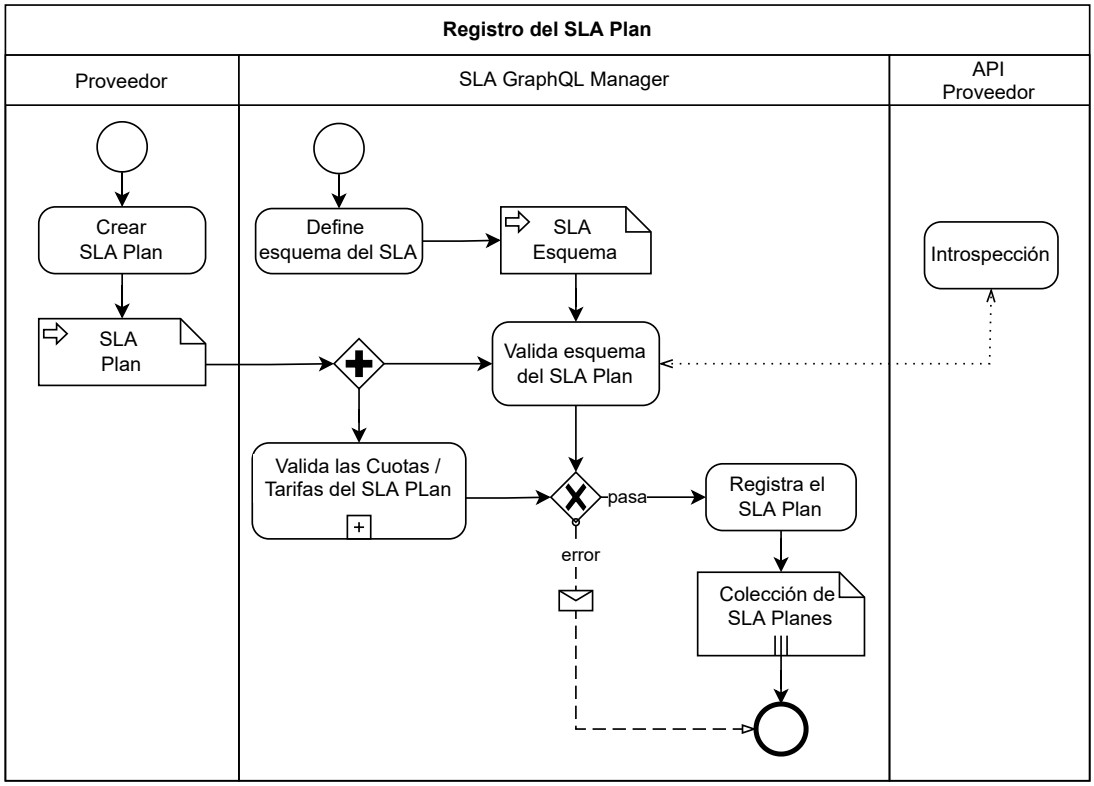


Figura 8.5: Proceso del registro del SLA Plan

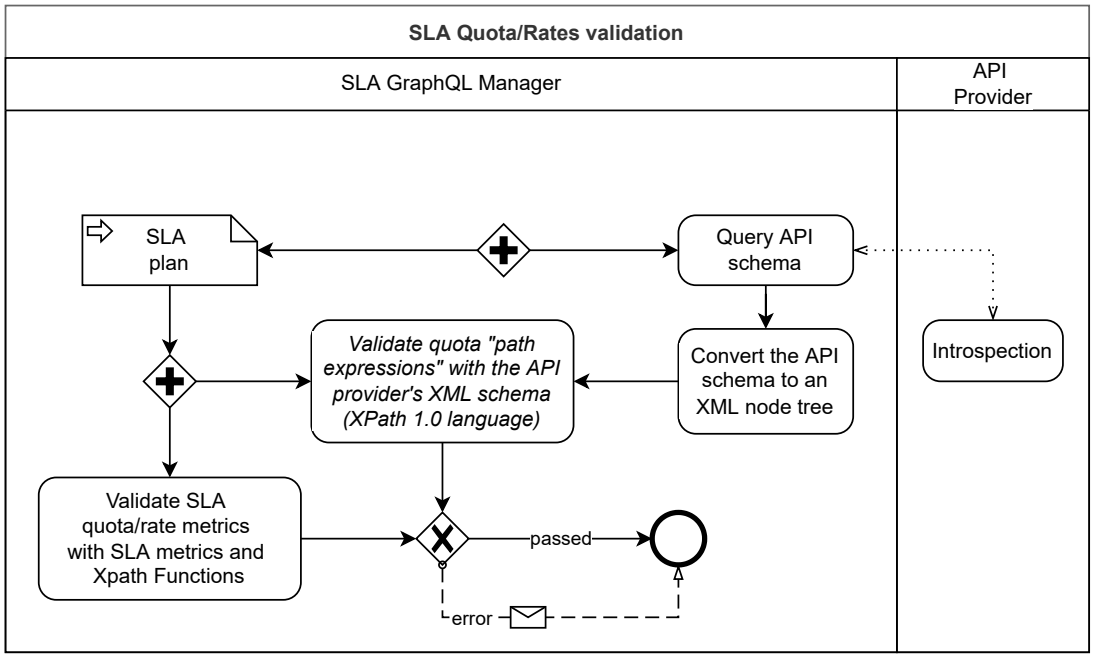


Figura 8.6: Validación de cuotas y tarifas del SLA Plan

Procesos de implementación y operación del SLA Plan

La actividad de *implementación del SLA Plan* se lo realiza en el API proveedor de tal manera que la *Operación* de la API responda a las solicitudes de consumo del Cliente de acuerdo a las limitaciones que especifique el SLA Plan que se haya registrado el Cliente (ver figura §8.4).

La Figura §8.7 muestra el proceso de referencia de la implementación y Operación del SLA Plan, en donde, se centra en validar las métricas de Cuotas/Tarifas sobre la solicitud, la respuesta, y el modelo de datos, para que al final se entregue la respuesta de la operación del API GraphQL al Cliente.

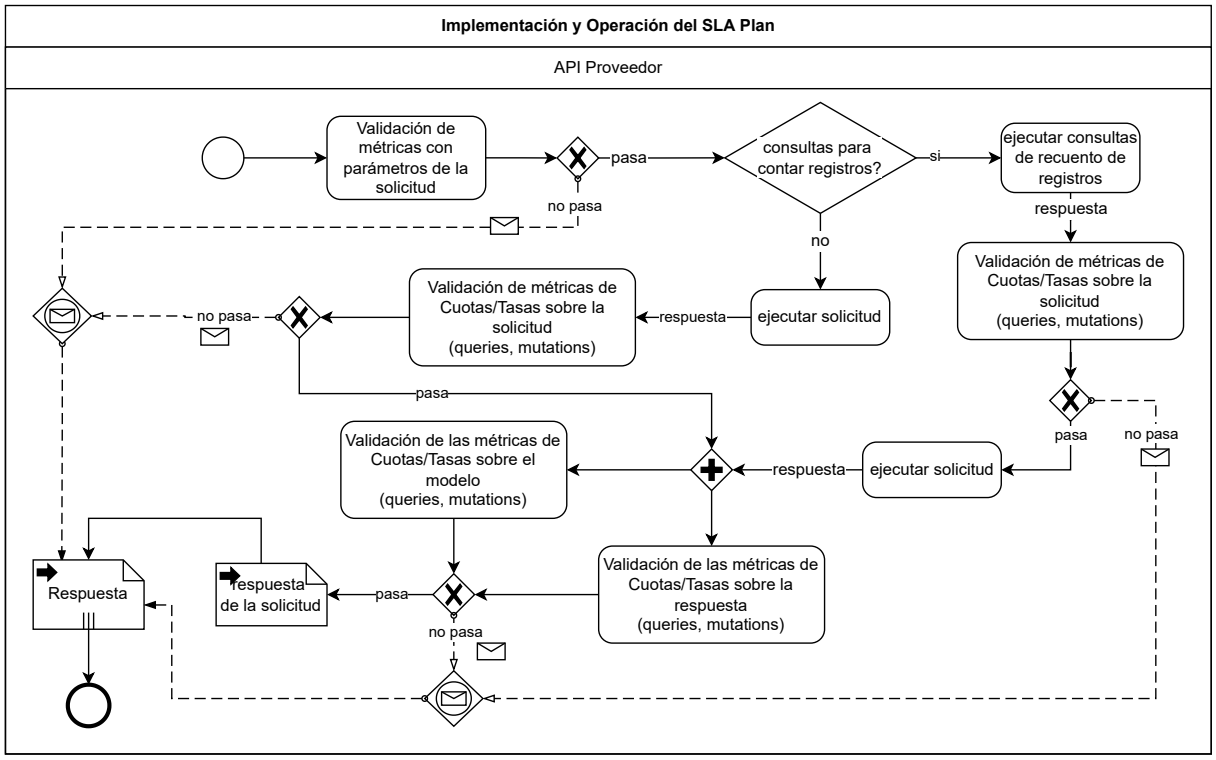


Figura 8.7: Implementación y Operación del SLA Plan

Este proceso comienza validando los parámetros de la solicitud con las métricas que especifican limitaciones con parámetros, si pasa la validación continua el proceso, sino no pasa la validación se emite un mensaje de error como respuesta y se termina el proceso. En el caso que no existan métricas con parámetros, pasa la validación y continúa el proceso. Luego para cada ruta de cuotas/tarifas se verifica si tienen consultas de conteo de registros: Por un lado, si existe las consultas de conteo, se las ejecutará, y con su respuesta se validará las métricas; en el caso que no pase la validación se emite

un mensaje de error como respuesta y se termina el proceso; en el caso que pase la validación se ejecuta la solicitud en el API GraphQL del proveedor. Por otro lado, si no existen las consultas de conteo, se ejecuta la solicitud del cliente y con la respuesta se validan las métricas. Luego, de haber validado las métricas sobre la solicitud, se debe validar simultáneamente la solicitud de consulta o mutación con las métricas de cuotas/tarifas sobre la respuesta de la consulta; y sobre el modelo de datos. Si la validación de cualquiera de las métricas falla, se debe emitir un mensaje de error como respuesta y se termina el proceso, en el caso que pase la validación, se toma la respuesta de la solicitud, y se retorna como resultado del proceso.

8.3 SLA GraphQL Manager (Herramienta)

De acuerdo a lo establecido en la sección §8.2.2, la herramienta *SLA GraphQL Manager* es un actor más en el ciclo de vida del desarrollo de APIs basadas en SLA; el cual, tiene como objetivo registrar los SLA Planes de los proveedores, y registrar la instanciación de los clientes en los SLA Planes de proveedores. La interfaz de la herramienta inicialmente, está diseñada para que se utilice como una librería que los proveedores, y clientes deben integrar a sus programas para gestionar los SLA de las API GraphQL.

8.3.1 Arquitectura del SLA GraphQL Manager

La herramienta *SLA GraphQL Manager*, se compone por una librería que es la interfaz para que los Proveedores pueda registrar sus SLA Planes, y los Clientes puedan registrar la instanciación a los SLA Planes de proveedores. La librería a su vez usa una API desplegada en la nube para gestionar un repositorio de registros de proveedores, clientes, SLA Planes, e instanciación de los SLA Planes. La Figura §8.8 muestra la arquitectura de la herramienta SLA GraphQL Manager.

SLA GraphQL Manager (Librería npm)

Esta librería de código abierto tiene como objetivo gestionar el registro de proveedores, clientes, SLA Planes, e instanciación de los SLA Planes. Esta librería es de tipo (NPM, por sus siglas en inglés de Node Package Manager), está disponible en la nube en el repositorio *NPMJS* con el nombre de *sla-graphql-manager*, la cual, se debe descargar en tiempo de diseño en los programas de proveedor o cliente (ver figura §8.8) para realizar las actividades de gestión de los SLA antes mencionadas.

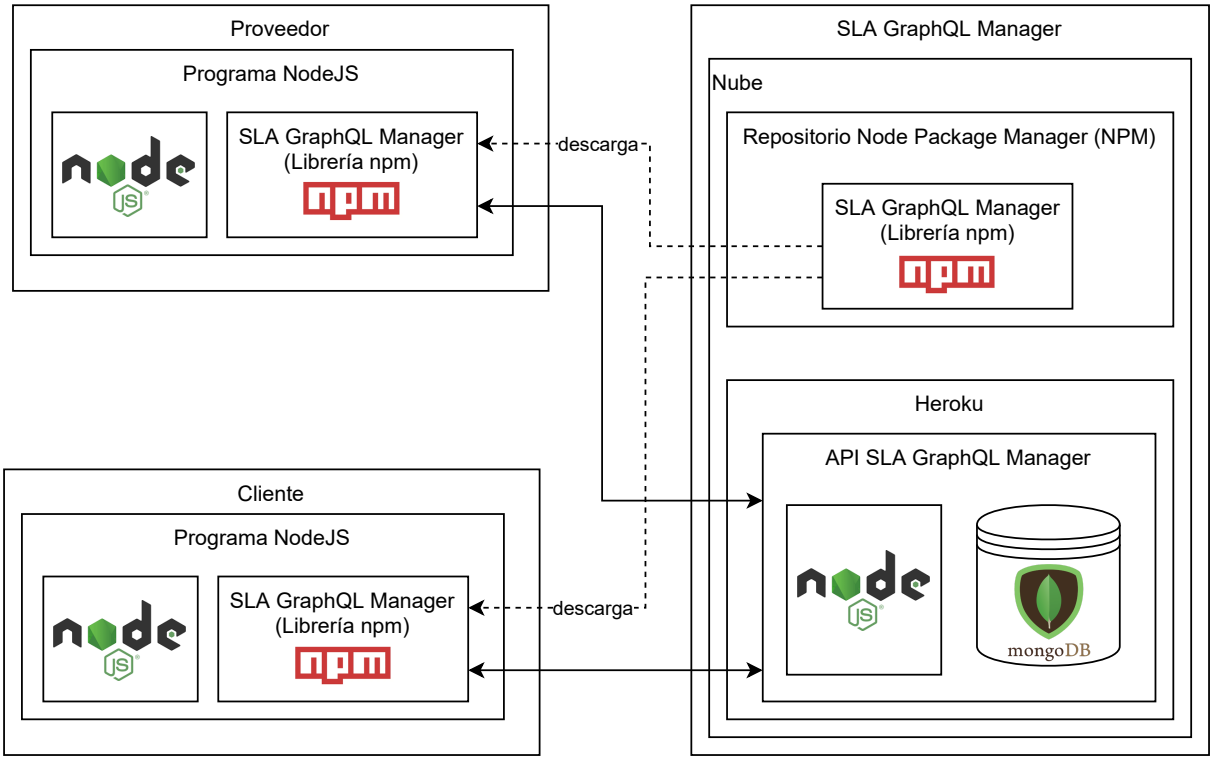


Figura 8.8: Arquitectura de la herramienta SLA GraphQL Manager

A continuación se muestra las **especificaciones técnicas** de la librería npm:

- Nombre técnico de la librería: sla-graphql-manager
- Repositorio de la librería: `https://www.npmjs.com/`
- Página de la librería: `https://www.npmjs.com/package/sla-graphql-manager`
- Repositorio GitHub: `https://github.com/antonio-quina/sla-graphQL`
- Lenguaje de programación: JavaScript
- Entorno de tiempo de ejecución de JavaScript: NodeJS v14.17.5

SLA GraphQL Manager (API)

La API SLA GraphQL Manager tiene como objetivo persistir la información de la gestión realizada por la librería npm. Esta API, se encuentra disponible en la nube, para que pueda ser accedida por la librería tal como se muestra en la figura §8.8.

A continuación se muestra las **especificaciones técnicas** de la API:

- Nombre técnico de la API: sla-graphql-server
- Repositorio de la API: `https://id.heroku.com/`
- URL del API: `https://sla-graphql-server.herokuapp.com/`
- Lenguaje de programación: JavaScript
- Entorno de tiempo de ejecución de JavaScript: NodeJS v14.17.5
- Base de datos: MongoDB

A continuación, en las siguientes secciones se describe el uso de las principales funcionalidades de la librería npm.

8.3.2 Registro del SLA Plan de proveedores

El registro del SLA Plan lo debe realizar el proveedor del API utilizando la librería sla-graphql-manager. Para lo cual, primero se debe describir el SLA Plan conforme a la especificación descrita en la sección §8.2.1. Luego, mediante un programa basado en NodeJS debe descargar e instalar la librería *sla-graphql-manager* de tipo npm del repositorio de la nube *NPMJS*. Y al final debe usar la librería para registrar el SLA Plan, en este paso, es importante aclarar que antes de registrar el SLA Plan, la librería realiza dos acciones previas: i) validará el SLA Plan de acuerdo a lo descrito en la figura §8.5; y ii) registrará o actualizará la información del proveedor descrito en el componente "context" del SLA Plan.

La *descarga e instalación de la librería*, se puede realizar con uno de estos comandos (depende del sistema operativo):

- npm i –save sla-graphql-manager
- yarn add sla-graphql-manager

La Figura §8.9 describe el código fuente en lenguaje JavaScript para registrar un SLA Plan del proveedor mediante la librería sla-graphql-manager.

```
import { instanceSLAGraphqlManager } from "sla-graphql-manager"
import fs from "fs"

async function app() {
    // Leer el contenido del acuerdo SLA
    const slaAgreement = fs.readFileSync("sla-plan.yaml", "utf-8")
    // Configuración de la API Proveedor
    const ApiProvider = instanceSLAGraphqlManager({
        introspectionConf: {
            // Cabecera de autenticación al API Proveedor
            headers: {
                authorization: `Bearer TOKEN`,
            },
            url: "https://direccion.url.del.api.provider",
            levels: 4 // Nivel de profundidad de la instrospección
        }
    })
    // Función que registra el SLA Plan
    const result = await ApiProvider.registerSLA(slaAgreement)
    // Resultado del registro del SLA Plan
    if (result.status === "SUCCESS")
        console.log("Registro exitoso del SLA Plan");
    else
        process.exit(1)
};
app();
```

Figura 8.9: Registro del SLA Plan

8.3.3 Registro del cliente a un SLA Plan de proveedor

El cliente para registrarse a un SLA Plan de proveedor debe utilizar un programa NodeJS y la librería sla-graphql-manager. Para lo cual, como requisito previo el proveedor debe haber registrado el SLA Plan (ver figura §8.5). Luego el cliente mediante la librería debe instanciar su registro, indicando el proveedor y SLA Plan al que desea registrarse. El resultado de esta acción, si es correcta, la librería devuelve al cliente un TOKEN de acceso al API proveedor (tokenAccess), el cual, le servirá para autenticar sus conexiones al API proveedor; caso contrario devuelve un mensaje de error.

Al igual que la sección anterior, la *descarga e instalación de la librería*, se puede realizar con uno de estos comandos (depende del sistema operativo):

- npm i –save sla-graphql-manager
- yarn add sla-graphql-manager

La Figura §8.10 describe el código fuente del uso de la librería sla-graphql-manager en lenguaje JavaScript para que el cliente se registre a un SLA Plan de proveedor.

```
import {slaGraphqlClient} from "sla-graphql-manager"

async function app (){
    // Función de registro al SLA Plan (sla-plan) del Proveedor (nombre.provider)
    const tokenAccess = await slaGraphqlClient.registerToSlaPlan({
        email: 'correo@del.cliente',
        agreement: 'sla-plan',
        provider: 'nombre-proveedor'
    })
    // Conservar el TOKEN de acceso al API provider
    console.log(tokenAccess);
  };
app();
```

Figura 8.10: Registro del cliente a un SLA Plan de proveedor

8.3.4 Consumo del API Proveedor

El cliente para consumir el servicio del API proveedor basado en el SLA, debe utilizar un programa NodeJS y la librería sla-graphql-manager. Para lo cual, como requisito previo debe tener el TOKEN de acceso al API proveedor que se obtiene al momento de registrarse al SLA Plan (ver figura §8.10). Luego el cliente mediante la librería debe configurar con el TOKEN de acceso una instancia de conexión al API GraphQL del proveedor. Finalmente, el cliente puede realizar operaciones de consumo al API proveedor mediante al instanciación configurada.

La *descarga e instalación de la librería*, se debe realizar igual que lo descrito en las secciones anteriores. Cabe indicar que el código fuente del registro y consumo del SLA Plan, son referenciales, los cuales podrán implementarse con ligeros cambios, debido al uso de políticas específicas del contexto de su aplicación o por la implementación de diversas buenas prácticas en el desarrollo de software.

La Figura §8.11 describe la forma de usar la librería para que el cliente consuma el servicio del API proveedor basado en el SLA Plan que se registró.

```
import { slaGraphqlClient } from "sla-graphql-manager"
import { gql } from "graphql-request"

async function app() {
    // Token de acceso recuperado del registro al SLA Plan del proveedor
    const tokenAccess = "TOKEN-DE-ACCESO"
    // Configuración de conección al API Proveedor
    const clientSession = await slaGraphqlClient.newSession({
        accessToken: tokenAccess
    })
    // Consulta en lenguaje GraphQL
    const query = gql`
        Execute Query
        query nombre_consulta{
            consulta
            {
                campo1
            }
        }`
// Resultado de la consulta
const result = await clientSession.request(query)
}
app();
```

Figura 8.11: Consumo del API Proveedor

8.4 RESUMEN

En este capítulo se detalla la propuesta principal de esta memoria, donde se propone un estándar de código abierto llamado ***SLA4GraphQL***, que especifica como modelar y operativizar acuerdos de nivel de servicio SLA para APIs GraphQL.

La propuesta *SLA4GraphQL*, se estructuró por una especificación para modelar documentos SLA; un ciclo de vida de desarrollo de APIs GraphQL basadas en SLA; y la especificación de la herramienta *SLA GraphQL Manager* que aporta la operativización de los SLA. La principal motivación de esta propuesta es dar el primer paso para contestar la pregunta de investigación $\mathbf{PI}_4$: ¿En qué medida es posible modelar los acuerdos de APIs GraphQL?. El segundo paso y complemento para contestar la pregunta $\mathbf{PI}_4$, es la validación de la propuesta *SLA4GraphQL*; esto se lo realizará en el capítulo §11.

Parte IV

Validación

Efectos de las arquitecturas REST y GraphQL: Validación

Cuéntamelo y me olvidaré, enséñamelo y lo recordaré, involúcrame y lo aprenderé.

Benjamin Franklin (1706 – 1790),
Erudito, inventor, filósofo, y político estadounidense

En este capítulo, se valida y contesta las preguntas de investigación propuestas en el diseño experimental del capítulo §6 que analiza el proceso de desarrollo del software, con el propósito de comparar la facilidad de aprendizaje del paradigma de programación GraphQL y REST con respecto a la calidad del software, desde un punto de vista de los investigadores en un contexto académico.

En la sección §9.1, se presenta la introducción y antecedentes del capítulo. En la sección §9.2, se detalla la ejecución del experimento propuesto en el capítulo §6. La sección §9.3 muestra los resultados de la ejecución del experimento. En la sección §9.4 se analiza las amenazas a la validez de los resultados y el proceso experimental. La sección §9.5 muestra las conclusiones del estudio. Por último, la sección §9.6 resume el capítulo.

9.1 Introducción

La motivación de este capítulo es ejecutar y validar el diseño experimental propuesto en el capítulo §6. Con los resultados obtenidos de la ejecución, se realizará un análisis utilizando la estadística descriptiva y prueba de hipótesis para responder las siguientes preguntas de investigación establecidas en el mencionado diseño:

- **PI**$_{2.1}$: ¿Cuál es el efecto del paradigma de desarrollo en la calidad del software?.
- **PI**$_{2.2}$: ¿Cuál es la curva de aprendizaje de los paradigmas REST y GraphQL en el proceso de desarrollo de APIs?.

Estas preguntas de investigación, a su vez responderán la pregunta de investigación **PI**$_2$: ¿Qué condiciones y circunstancias hacen más recomendable seguir el paradigma GraphQL frente al paradigma REST?.

9.2 Ejecución del experimento

El experimento se realizó en la tarde del 5 de junio del 2021 de 14H0 a 19H10, en un ambiente virtual utilizando la herramienta de colaboración Microsoft Teams, el grupo de trabajo en la herramienta fue privado con acceso controlado solo para los experimentadores y sujetos experimentales.

9.2.1 Muestra

Los 23 participantes fueron estudiantes de quinto nivel de la Carrera de Ingeniería en Software. Las clases síncronas de entrenamiento tuvieron lugar los días jueves (3 horas) y viernes (2 horas) durante 3 semanas conforme a la planificación establecida en la Tabla §6.2, en el mismo ambiente virtual que se ejecutó el experimento. Los participantes tenían la misma experiencia como desarrolladores que se basaba en los niveles alcanzados en su formación profesional.

9.2.2 Preparación

Los sujetos ingresaron puntuales al entorno virtual del experimento, de acuerdo, a la planificación establecida en la Tabla §6.5. Se comenzó con las indicaciones generales,

en donde, el experimentador A. Quiña, expuso las tareas, los tiempos e instrumentos que debían ejecutar en las dos sesiones de trabajo. Luego los experimentadores C. Guevara y F. Chulde entregaron a cada participante el respaldo y script de base de datos de acuerdo al proyecto que debían desarrollar, previsto en el diseño del experimento (ver Tabla §6.1). Los participantes, cargaron el respaldo o ejecutaron el script de creación de base de datos en sus ambientes de desarrollo, que era el mismo ambiente en que hicieron las tareas de entrenamiento; en este punto cabe mencionar, que tenían el soporte técnico de los experimentadores, sin embargo, ningún participante necesito el soporte ofrecido. El siguiente paso se realizó de manera similar a la que se entregó los instrumentos de base de datos, en donde, se distribuyó un archivo Excel que contenía los requisitos de desarrollo del proyecto y una sección para registrar evidencias de solución de cada requisito.

En la ejecución de las dos sesiones del experimento, los participantes registraron las evidencias de solución de la siguiente manera: i) para comenzar a desarrollar un requisito, tenían que registrar la hora de inicio; ii) cuando terminaban de desarrollar el requisito, debían registrar la hora de finalización, y un estimado del porcentaje de solución; y iii) registrar una captura de pantalla de la ejecución del requisito. En este proceso, los experimentadores atendieron consultas de los sujetos experimentales respecto a aclaraciones de la actividad, absteniéndose de dar respuesta a consultas relativas a la resolución de la tarea experimental para no interferir en el experimento. Debido a que el experimento se realizó en un ambiente virtual, se tomó varias medidas y prácticas para paliar el efecto de copia entre participantes, las medidas fueron las siguientes: i) Los participantes, tuvieron que encender la cámara, y micrófono durante las sesiones de ejecución del experimento; ii) durante cada sesión, los participantes tuvieron que grabar su pantalla para evidenciar el proceso de desarrollo del proyecto asignado; iii) los experimentadores, controlaban gestos atípicos de los participantes, como movimientos de la cabeza (por ejemplo cuando revisa dos monitores, o un monitor y un celular).

9.2.3 Recolección de datos

Al final de cada sesión, los participantes subieron el código fuente de los proyectos desarrollados, y el video de la captura de la pantalla a un repositorio digital del grupo de Teams previsto para el experimento. En este proceso, luego de cada sesión, los experimentadores verificaron que todos los participantes carguen los proyectos y videos en el repositorio, de acuerdo con lo establecido en las indicaciones.

La encuesta demográfica/autoevaluación se realizó al siguiente día de la ejecución del experimento.

9.3 Resultados

En esta sección, se presenta el análisis estadístico de los resultados de la ejecución del experimento. En donde, por un lado, se realiza la estadística descriptiva de los efectos de los paradigmas REST y GraphQL en la calidad de software mediante la operativización de las métricas de completitud y eficacia; además, se complementa este estudio con el análisis de la curva de aprendizaje de los paradigmas. Por otro lado, se realizan las pruebas de hipótesis del experimento.

9.3.1 Estadística descriptiva

Calidad del Software

Por un lado, para la calidad de software se analizó el efecto de la *Completitud* de la guía del usuario, que se refleja cuando el estudiante completa actividades o funciones que se le explicaron para su desarrollo, en este sentido, la Tabla §9.1 muestra los resultados estadísticos, obtenidos de la ejecución del experimento para la métrica *completitud* en función al paradigma REST y GraphQL.

Tabla 9.1: Estadística descriptiva de la completitud

Estadísticos descriptivos	REST	GraphQL
Media ± Desviación estándar	46.09 ± 34.87	52.61 ± 33.20
Mediana	55	60
Media CI 95%	[31.01; 61.17]	[38.25; 66.96]
Mín. - Máx.	0 - 100	0 - 100
Rango	100	100
Rango intercuartílico	65	50
Asimetría	0.02	-0.11
Curtosis	-1.20	-1.21
Error típico	7.27	6.92

El valor medio de *Completitud* para el paradigma GraphQL fue de 52.61 ± 33.20 (Media ± desviación estándar) con un intervalo de confianza del 95% de [38.25 - 66.96]. En el paradigma REST, tanto el valor medio (46.09 ± 34.87) como los límites del intervalo

de confianza del 95% de [31.01 - 61.17] fueron ligeramente menores en comparación con el paradigma GraphQL; es decir, los resultados indican que el porcentaje de las respuestas completadas por los estudiantes en el paradigma REST oscila entre 11.22 a 80.96, y en GraphQL oscilan entre 23.41 - 85.81. En donde, se aprecia que en GraphQL se obtiene un nivel adecuado de completitud de tareas asignadas. Sin embargo, de acuerdo la ISO/IEC 14598-1 [72] el rango inferior (rango: inaceptable) y superior (rango objetivo) de los dos paradigmas están en los mismos rangos de significancia de resultados (ver Figura §9.1).

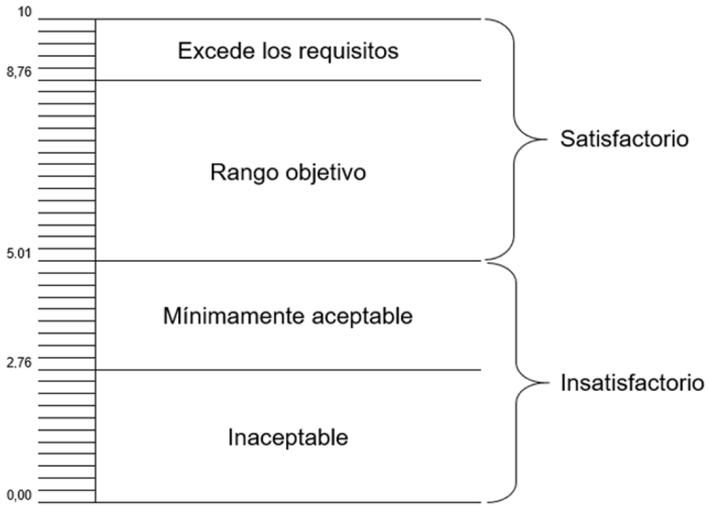


Figura 9.1: Escala de medición de resultados

Además, se observó que el rango y el rango intercuartílico fue ligeramente mayor para el paradigma REST. La dispersión en ambos niveles fue considerablemente grande en comparación a la media. Además, el amplio rango 100 para ambos paradigmas, indicó una gran variabilidad entre los participantes. Las medidas de asimetría y exceso de curtosis indicaron distribuciones asimétricas con una concentración plana (platicúrtica) para ambos paradigmas y una distribución moderadamente sesgada hacia la derecha para el paradigma REST y a la izquierda para GraphQL. Esto también es evidente en los bigotes del BloxPlot que se presenta en la Figura §9.2, donde la tendencia indica una relación entre la tarea, caso, paradigma y *Completitud*.

Por otro lado, para la calidad de software se analizó también el efecto de la *Eficacia* en el ejercicio del experimento que se operativiza mediante la métrica de *tareas completadas* correctamente sin ayuda. En este sentido, la Tabla §9.2 muestra los estadísticos descriptivos de los resultados de las métricas. En donde, se pudo observar que los estadísticos de eficacia tuvieron un comportamiento similar a los estadísticos de com-

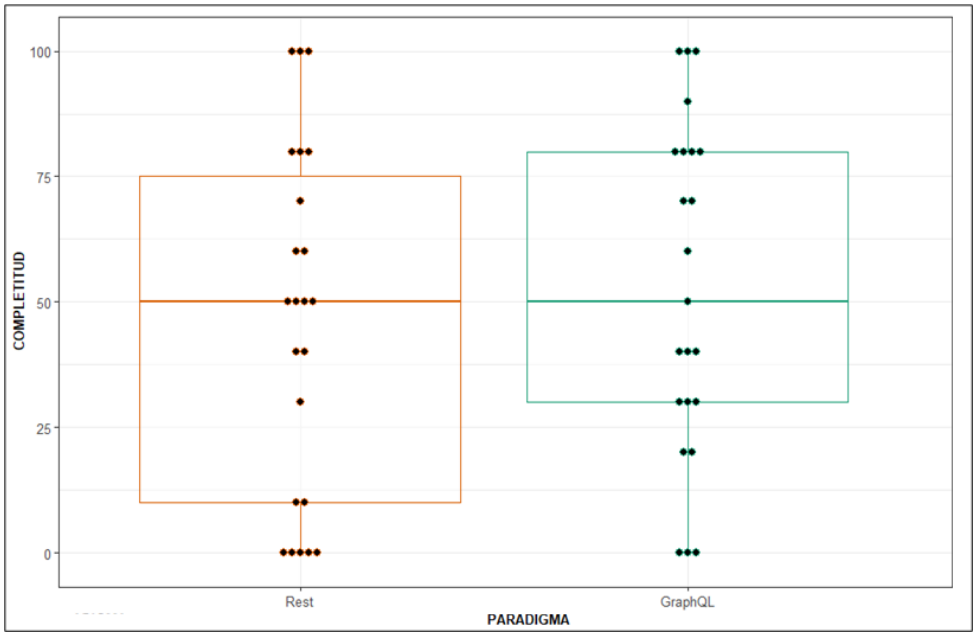


Figura 9.2: Diagrama de caja y bigotes de la completitud

pletitud, es decir, la media del valor de la eficacia para REST osciló en [0.61 ± 0.40] y para GraphQL en [0.66 ± 0.36]. Es así, que debido al porcentaje de eficacia de las tareas completadas en el paradigma GraphQL fueron ligeramente superior a las tareas completadas en el paradigma REST, sin embargo, de acuerdo la ISO/IEC 14598-1 [72] (ver Figura §9.1) los resultados tienen el mismo rango de significancia tanto para el rango inferior (rango: inaceptable) como superior (rango objetivo).

Tabla 9.2: Estadística descriptiva de la eficacia

Estadísticos descriptivos	REST	GraphQL
Media ± Desviación estándar	0.61 ± 0.40	0.66 ± 0.36
Mediana	0.71	0.75
Media CI 95%	[0.43; 0.78]	[0.50; 0.81]
Mín. - Máx.	0 – 1	0 – 1
Rango	1	1
Rango intercuartílico	0.77	0.64
Asimetría	-0.54	-0.68
Curtosis	-1.35	-0.89
Error típico	0.08	0.08

Además, en la eficacia y de manera similar a la completitud, se puede evidenciar en el Plot-violín de la Figura §9.3, el aprendizaje tiende a tener una relación con la tarea, el caso, paradigma y *eficacia*, por lo tanto, se puede decir que, las tareas en el paradigma

GraphQL es ligeramente más fácil de aprender a comparación del paradigma REST.

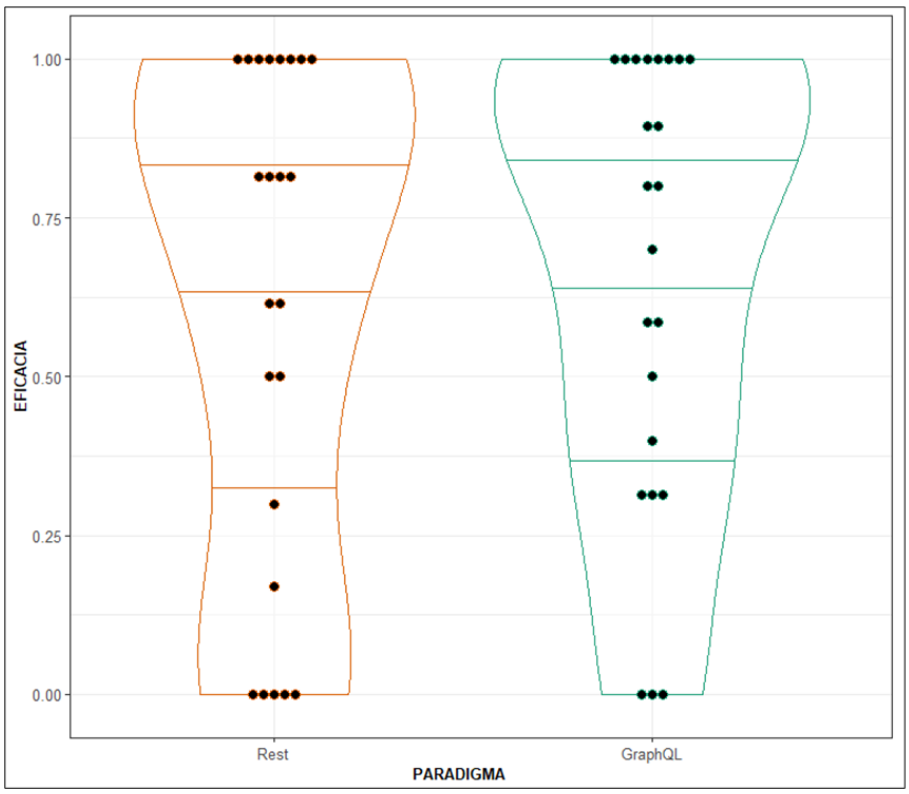


Figura 9.3: Diagrama de violín de la eficacia

Curva de aprendizaje

La curva de aprendizaje representa el aprendizaje individual adquirido por los participantes en los paradigmas de programación REST y GraphQL. Los cálculos se realizaron de acuerdo a lo establecido en la sección §6.2.3, en donde, el tiempo medio de la solución de los requisitos de las tareas (taller práctico, deber, tarea experimental) está medido en milisegundos; y que luego se complementa con la proyección para diez iteraciones, tal como se muestra en la Figura §9.4.

La Tabla §9.3 muestra los estadísticos descriptivos del cálculo de la curva de aprendizaje. En donde, se puede notar que el valor medio de *Curva de aprendizaje* para el paradigma GraphQL fue de 3518.23 ± 2361.25 con un intervalo de confianza del 95% de [1829.09 - 5207.37]; mientras que para el paradigma REST el valor medio es 4269.18 ± 846.48 con límites del intervalo de confianza del 95% de [3663.64 - 4874.71].

Como se puede apreciar en la figura §9.4 y tabla §9.3, la diferencia entre los paradigmas es representativa en el tiempo que los participantes aprenden cada uno de los paradigmas, en este sentido, se denota que *GraphQL tiene una curva de aprendizaje más*

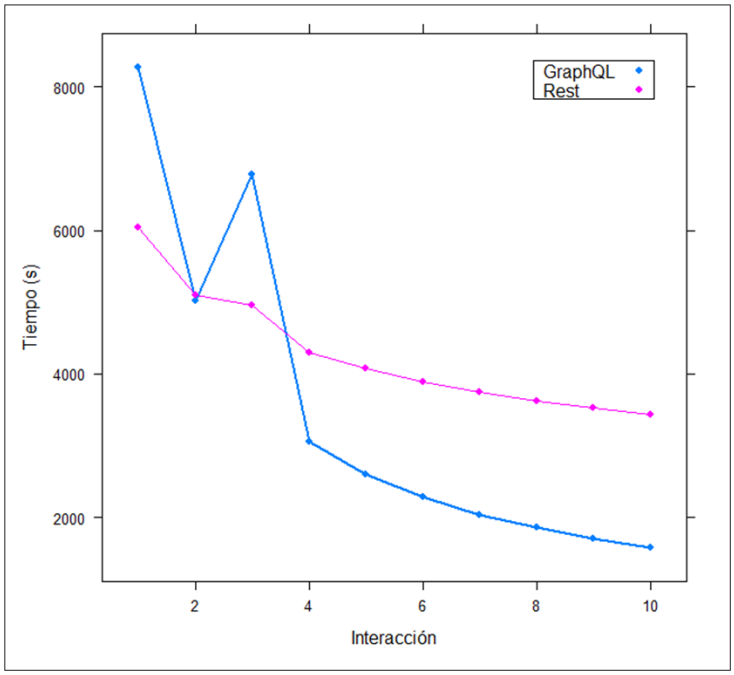


Figura 9.4: Tendencia de la curva de aprendizaje

eficiente que la curva de aprendizaje del paradigma REST.

9.3.2 Prueba de hipótesis

En este apartado, se realiza el análisis estadístico para verificar las hipótesis planteadas en la sección §6.2.4 acerca del efecto que tienen los paradigmas de programación en la calidad de software. Por un lado, se comenzó analizando la *completitud*, en donde, se utilizó el modelo lineal de efectos mixtos tal como se detalló en la sección §6.2.10, con el cual, se describe la ecuación mixta $Completitud \sim Paradigma + (1|Participante)$ para

Tabla 9.3: Estadística descriptiva de la curva de aprendizaje

Estadísticos descriptivos	REST	GraphQL
Media ± Desviación estándar	4269.18 ± 846.48	3518.23 ± 2361.25
Mediana	3981.35	2438.67
Media CI 95 %	[3663.64;4874.71]	[1829.09;5207.37]
Mín. - Máx.	3432.37 – 6048.44	1577.67 – 8277.27
Rango intercuartílico	1132.72	2632.68
Asimetría	1.16	1.29
Curtosis	0.67	0.38
Error típico	267.68	746.69

calcular los efectos del aprendizaje utilizando las métricas propuestas en la ISO/IEC 25022 [76] del modelo de calidad de software. En primer lugar, se presenta el modelo, los resultados de las pruebas estadísticas de los efectos, y el diagnóstico del modelo. Para determinar que los supuestos del modelo están de acuerdo con los parámetros paramétricos, se aplicó una transformación inversa de la acumulada de la distribución normal estándar (probit) sobre la completitud; y se incluyó al paradigma como factores fijos y a los participantes como factores aleatorios.

La Tabla §9.4 muestra las estimaciones de los parámetros para los factores fijos y los intervalos de confianza, en donde, el parámetro del paradigma:REST denota que el promedio de las tareas completadas en REST tiene una disminución del 0.34 de la completitud a comparación de las tareas completadas con GraphQL.

Tabla 9.4: Resultados del modelo lineal de efectos mixtos de la completitud

Parámetro	Estimación	Error	Intervalo de confianza (95%)
Intercept	5.22	0.20	(4.84, 5.60)
Paradigma:REST	-0.34	0.24	(-0.81, 0.14)

La importancia de los efectos del paradigma que describen la complejidad de las tareas realizadas en este modelo, se investigaron utilizando la *Prueba de Wald* para efectos fijos, los resultados se muestran en la Tabla §9.5. En donde, se observa que el nivel de significancia *valor-P* es mayor que 0.05, por lo tanto, se acepta la *Hipótesis Nula* (H_0) establecida para el experimento.

Tabla 9.5: Prueba de Wald para efectos fijos de la completitud

Fuentes de variación	Grados de libertad numerador	Grados de libertad denominador	F	Valor-P
Paradigma	1	22	-1.46	0.078

Por otro lado, el análisis estadístico de la *Eficacia* se realizó de manera similar que el análisis de la completitud, en donde, en ambos casos se utilizó el modelo lineal de efectos mixtos, sin embargo, para normalizar los datos en este caso se utilizó la transformación de la raíz cuadrada sobre la eficacia, en donde, la ecuación mixta basada en la métrica de *tareas completadas* de la característica *Eficacia* de calidad en uso del software descrita en la ISO/IEC 25022 [76] quedó de la siguiente manera ($sqrt(Eficacia) \sim Paradigma + (1|Participante)$).

La Tabla §9.6 muestra las estimaciones de los parámetros para los factores fijos y los intervalos de confianza, en donde, el parámetro del paradigma:REST denota que el promedio de las tareas completadas en REST tiene una disminución del 0.08 de la eficacia a comparación de las tareas completadas de GraphQL.

Tabla 9.6: Resultados del modelo lineal de efectos mixtos de la eficacia

Parámetro	Estimación	Error	Intervalo de confianza (95%)
Intercept	2.28	0.04	(2.19, 2.36)
Paradigma:REST	-0.08	0.05	(-0.18, 0.03)

Los efectos del paradigma en la eficacia de las tareas completadas, de acuerdo al test de *Prueba de Wald* que se muestra en la Tabla §9.7, denota que el nivel de significancia *valor-P* es mayor que 0.05, por lo tanto, en este caso, al igual que la completitud, se acepta la *Hipótesis Nula* (H_0) establecida en el experimento, es decir, *no hay diferencia en los efectos de los paradigmas de programación en la calidad del software*.

Tabla 9.7: Prueba de Wald para efectos fijos de la eficacia

Fuentes de variación	Grados de libertad numerador	Grados de libertad denominador	F	Valor-P
Paradigma	1	22	-1.46	0.072

9.4 AMENAZAS A LA VALIDEZ

Las amenazas a la validez son un conjunto de situaciones, factores, debilidades y limitaciones que podrían interferir con la validez de los resultados del presente estudio empírico, por lo cual se analizan las posibles amenazas relevantes basándose en la clasificación propuesta por Wohlin *et al.* [186].

9.4.1 Validez interna

El proceso de desarrollo de APIs se manipuló mediante la descripción de tareas experimentales para implementar tres casos en los paradigmas de programación REST y GraphQL. Para confirmar que esto no causó algún factor de confusión a los grupos del tratamiento, como la falta de comprensión de la tarea o la mala adecuación de las

especificaciones. En el documento de registro de la ejecución de la tarea se estableció un apartado para que el sujeto experimental pueda registrar inconvenientes de adecuación y comprensión de indicaciones en caso de que los tuviera.

Dado que el ámbito de este estudio era el paradigma de programación, la capacidad de los participantes para implementar en los paradigmas también fue relevante para la validez de los resultados. Como precaución, se proporcionó una explicación teoría, ejemplos, talleres, y deberes como tareas de formación sobre los paradigmas antes de realizar el experimento. Además, se evaluó el grado de conformidad de los sujetos experimentales acerca de los paradigmas de programación mediante una autoevaluación (encuesta) luego de realizar las tareas experimentales.

9.4.2 Validez externa

El experimento se llevó a cabo en un entorno académico en el contexto de un curso de formación de pregrado, lo que limita la capacidad de generalizar los resultados a la práctica industrial. Esto fue en parte por diseño, ya que parte de la propuesta es identificar la facilidad de aprendizaje en desarrolladores novatos. Las tareas utilizadas en el experimento fueron sencillas con una duración de dos horas de ejecución. Por lo tanto, la generalización a tareas más complejas de la vida real es limitada, sin embargo, por tratarse de tareas básicas de manipulación de datos CRUD, es razonable esperar que también tenga el mismo efecto en tareas más complejas. Las tareas elegidas para el experimento fueron equivalentes en complejidad, por lo que no se esperaba que afecte a los resultados de experimento, por el contrario, se esperaba reducir el efecto de copia entre sujetos experimentales; sin embargo, las tareas en la fase de formación si tuvieron complejidades diferentes, y el efecto fue el mismo. El hecho de que el efecto del tratamiento fuera el mismo para las tareas de formación y experimentales se sugiere que los resultados pueden generalizarse a tareas con complejidades similares.

9.4.3 Validez de constructo

Los constructos utilizados en la fase de ejecución del experimento se diseñaron para medir las métricas establecidas para la variable dependiente del estudio. Estos constructos se definieron de manera consensuada por tres expertos en ingeniería de software, y luego validados por los sujetos experimentales con constructos equivalentes en las tareas de formación. La validación se realizó luego de la ejecución de los constructos, utilizando una sección de observaciones y recomendaciones y también realizando

una reunión de retroalimentación entre los sujetos y experimentadores. El estudio no sufrió de un sesgo mono-operativo ya que se utilizaron tres tareas separadas: eventos, facturas, y notas.

9.4.4 Validez de la conclusión

Para mitigar las amenazas en las conclusiones, se realizó una validación de pruebas estadísticas no paramétricas a los resultados para poder elegir y aceptar una de las hipótesis planteadas en el experimento y así poder fundamentar las conclusiones de este estudio empírico. Se realizaron pruebas no paramétricas porque en el análisis de estadísticos descriptivos se comprobó que los resultados obtenidos presentaban datos que no tenían normalidad. De esta manera se pudo aceptar la hipótesis nula establecida para el experimento.

9.5 Conclusiones y trabajo futuro

9.5.1 Conclusiones

En este capítulo se ejecutó y validó el diseño experimental propuesto en el capítulo §6 para comparar los efectos del paradigma REST frente al paradigma GraphQL en términos de la calidad de software, con la finalidad de contestar la pregunta de investigación $\mathbf{PI}_2$: ¿Qué condiciones y circunstancias hacen más recomendable seguir el paradigma GraphQL frente al paradigma REST? Para lo cual, se caracterizó las condiciones y circunstancias con las subcaracterísticas de calidad del producto de software: *Facilidad de Aprendizaje* que fue operativizada por la métrica de "completitud" de la guía de usuario y la *Eficacia* que se operativiza con la métrica "tareas completadas", basadas en la normas ISO/IEC 25010, 25022, 25023 [75, 76, 77]. Además, se complementó la caracterización de las condiciones con el estudio de la *Curva de Aprendizaje* de los paradigmas REST y GraphQL.

En este sentido, con los resultados obtenidos de la ejecución del experimento mencionado, se contesta las dos preguntas derivadas de la $\mathbf{PI}_2$:

$\mathbf{PI}_{2.1}$: ¿Cuál es el efecto del paradigma de desarrollo en la calidad del software?. De acuerdo con el análisis estadístico descriptivo se pudo observar que GraphQL presenta una ligera mejora en la *Eficacia* y *Facilidad de Aprendizaje* con respecto al paradigma

REST. Sin embargo, de acuerdo a la ISO/IEC 14598 y las pruebas estadísticas de *Wald*, estas diferencias no son significativas, por lo cual, se acepta la *Hipótesis Nula* (H_0) del estudio, es decir, *no hay diferencias significativas en los efectos de los paradigmas de programación en la calidad del software*.

PI$_{2.2}$: ¿Cuál es la curva de aprendizaje de los paradigmas REST y GraphQL en el proceso de desarrollo de APIs?. De acuerdo al análisis estadístico realizado en la sección §9.3, se puede notar que el tiempo medio de la solución de los requisitos realizados con GraphQL es menor que los requisitos realizados con REST. Además, se pudo observar que el cálculo de la curva de aprendizaje denota que los participantes aprenden más rápido el paradigma GraphQL frente al paradigma REST.

9.5.2 Trabajo futuro

Como trabajo futuro, se incentiva realizar la replicación de este experimento en otras poblaciones de participantes de pregrado y posgrado en el contexto académico, como también replicar el experimento en el contexto industrial. También, se promueve la experimentación para comparar otras características de la calidad del producto de software entre los paradigmas REST y GraphQL. Finalmente, se promueve el estudio comparativo de la facilidad de aprendizaje y eficiencia en los dominios del proveedor y cliente del servicio de APIs, como también la eficiencia a nivel de acceso de datos con los paradigmas REST y GraphQL.

9.6 RESUMEN

En este capítulo, se ejecutó el diseño experimental propuesto en el capítulo §6, en donde se analizó los efectos de los paradigmas REST y GraphQL, con respecto a la calidad del software, en el proceso de desarrollo del software, en un contexto académico. Con el análisis de los resultados obtenidos de la ejecución del experimento se logró responder la pregunta **PI**$_2$: ¿Qué condiciones y circunstancias hacen más recomendable seguir el paradigma REST frente al paradigma GraphQL?, mediante la contestación de sus preguntas de investigación derivadas: **PI**$_{2.1}$: ¿Cuál es el efecto del paradigma de desarrollo en la calidad del software?; y **PI**$_{2.2}$: ¿Cuál es la curva de aprendizaje de los paradigmas REST y GraphQL en el proceso de desarrollo de APIs?.

Efectos de las arquitecturas híbridas: Validación

El éxito es la suma de pequeños esfuerzos, que se repiten día tras día.

Robert Collier (1855 – 1950),
Escritor estadounidense

En este capítulo, se valida y contesta las preguntas de investigación propuestas en el diseño experimental del capítulo §7 que analiza el proceso de desarrollo del software, con el propósito de comparar la eficiencia del rendimiento de una arquitectura híbrida REST/GraphQL contra una arquitectura REST con respecto a la calidad del software, desde un punto de vista de los investigadores en un contexto controlado de un laboratorio computacional.

En la sección §10.1, se presenta una introducción y antecedentes del capítulo. En la sección §10.2, se detalla la ejecución del experimento propuesto en el capítulo §7. La sección §10.3 muestra los resultados de la ejecución del experimento. En la sección §10.4 se analiza las amenazas a la validez de los resultados y el proceso experimental. La sección §10.5 muestra las conclusiones del estudio. Por último, la sección §10.6 resume el capítulo.

10.1 Introducción

Este capítulo tiene como motivación ejecutar y validar el diseño experimental propuesto en el capítulo §7. Con los resultados obtenidos de la ejecución, se realizará un análisis de estadística descriptiva y de prueba de hipótesis para responder las siguientes preguntas de investigación establecidas en el mencionado diseño:

- **PI**$_{3.1}$: ¿Cuál es el efecto del desarrollo de una arquitectura híbrida REST/GraphQL en la calidad del software?.
- **PI**$_{3.2}$: ¿En qué escenarios es más adecuado usar las arquitecturas híbridas REST/-GraphQL?.

Estas preguntas de investigación, a su vez responderán la pregunta de investigación **PI**$_3$: ¿En qué escenarios resultan adecuadas las arquitecturas híbridas REST/-GraphQL?.

10.2 Ejecución del experimento

El experimento se ejecuto en diciembre 2020, utilizando el entorno experimental propuesto en la sección §7.2.

10.2.1 Muestra

Como muestra se tomó el repositorio de Mendeley de uno de los experimentadores, el cual, contiene la cantidad adecuada de información para que se pueda ejecutar el experimento de acuerdo al diseño propuesto en la sección §7.2.5.

10.2.2 Preparación

Se inicio verificando que el canal de conexión a internet sea de uso exclusivo para la ejecución del experimento. Se verificó que exista una correcta conexión y autenticación al repositorio Mendeley, también se verificó el funcionamiento que las tareas experimentales descritas en la sección §7.2.7 mediante el despliegue de la API GraphQL que envuelve la API REST de Mendeley, y la ejecución de varios casos de uso desde la

aplicación cliente. Luego de verificar el funcionamiento del entorno experimental, se realizó la ejecución del experimento de acuerdo al diseño establecido.

La ejecución del experimento se realizó de manera interactiva por caso de uso, en donde, se corrió el programa cliente una vez para cada caso de uso. En cada corrida del programa se configuró y ejecutó un caso de uso por tres ocasiones para tres diferentes cantidades de datos en cada paradigma conforme a los establecido en el diseño experimental (ver tabla §7.1), es decir, se ejecutó 9 veces el caso de uso para cada paradigma, en total se obtuvo una muestra de 108 registros por todas las corridas. La Figura §10.1 muestra el proceso de ejecución del experimento.

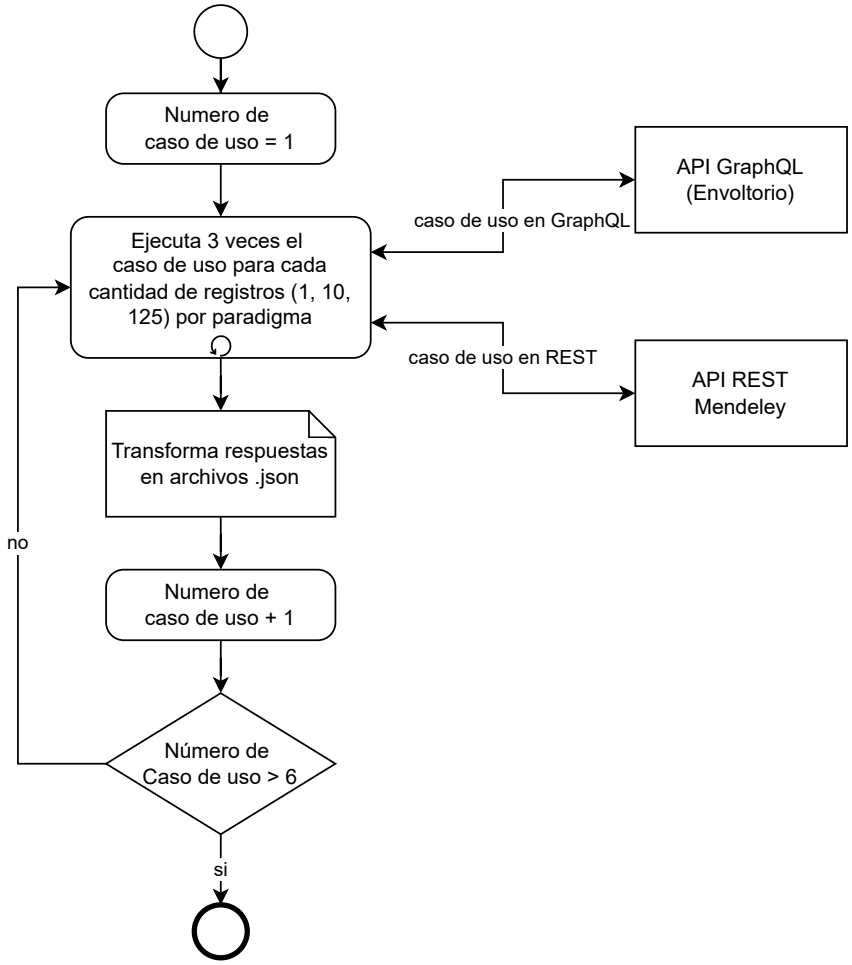


Figura 10.1: Proceso de ejecución del experimento

10.2.3 Recolección de datos

De acuerdo con el proceso de ejecución del experimento (ver figura §10.1), la recolección de datos se efectuó luego de cada corrida del programa cliente, en donde, el programa imprimió los resultados de los tiempos de respuesta en la consola de ejecución del IDE Visual Studio, y creó los archivos de respuesta en formato JSON en una carpeta local del programa. Los resultados de tiempo de respuesta se copiaron de la consola y tabularon en un libro de Microsoft Excel 365, y el tamaño de archivo de respuesta se extrajo de las propiedades del archivo y se registró en el libro Excel. La Figura §10.2 muestra un ejemplo de los resultados en consola, y la Figura §10.3 muestra un ejemplo de la recolección de resultados en el libro de Excel.

```
PS D:\Development\Tesis\Wendelev wrapper\Wendelev graphql wrapper client> node .\src\index.js
G-010-2-(UC_04) 1043.0815000534058
G-010-0-(UC_04) 1053.6917999982834
R-010-0-(UC_04) 1047.7414999008179
R-001-2-(UC_04) 1041.4327000379562
```

Figura 10.2: Ejemplo de resultados de la corrida del programa *Cliente*

A	B	C	D	E	F	G	H
Nro.	Caso de Uso	Nivel	Nro. Registros	Repetición	Paradigma	Tiempo (mseg)	Peso (bytes)
1	UC_04	2	10	2	GraphQL	1043,08	2672
2	UC_04	2	10	0	GraphQL	1053,69	2672
3	UC_04	2	10	0	REST	1047,74	2953
4	UC_04	2	1	2	REST	1041,43	276

Figura 10.3: Ejemplo de la recolección de resultados

10.3 RESULTADOS

En esta sección, se presenta el análisis del efecto del desarrollo de las arquitecturas híbridas REST/GraphQL en la calidad del software. Para lo cual, partiendo de los resultados obtenidos en la ejecución del experimento descrito en la sección §10.2; por un lado, se realizó un análisis estadístico de la eficiencia del rendimiento de las arquitecturas, y por otro lado, un análisis del tamaño de respuestas de las consultas de las arquitecturas REST/GraphQL y REST.

10.3.1 Análisis estadístico de la eficiencia

La característica de calidad *eficiencia del rendimiento*, se lo operativiza mediante la métrica del tiempo medio de respuesta basados en la ISO/IEC 25022 [76], en este sentido, en la Tabla §10.1 se muestra los resultados estadísticos, obtenidos de la ejecución del experimento. En donde, se puede observar que el valor medio de *Eficiencia* para la arquitectura híbrida REST/GraphQL fue de 2587 ± 2313 (Media ± desviación estándar) con un intervalo de confianza del 95% de [1956 - 3219]. En la arquitectura REST, tanto el valor medio (22695 ± 37349) como los límites del intervalo de confianza del 95% de [12501 - 32890] fueron significativamente mayores en comparación con la arquitectura REST/GraphQL. Es decir, que el tiempo medio de respuesta de la arquitectura híbrida REST/GraphQL es hasta ocho veces más eficiente que la arquitectura REST.

Tabla 10.1: Estadística descriptiva de la eficiencia

Estadísticos descriptivos	REST	GraphQL
Media ± Desviación estándar	22695.73 ± 37349.55	2587.55 ± 2313.48
Mediana	2312.12	2058.99
Media CI 95%	[12501.26; 32890.19]	[1956.09; 3219.01]
Mín. - Máx.	843.75 - 152792.24	738.91 - 13339.84
Rango	151948.5	12600.93
Rango intercuartílico	34175.31	1745.15
Asimetría	1.83	2.78
Curtosis	2.61	9.43
Error típico	5082.63	314.83

Además, la Figura §10.4 mediante un diagrama BloxPlot de medias del tiempo de respuesta, muestra que mientras más complejas son las consultas de datos, más eficiente se vuelve la arquitectura híbrida REST/GraphQL a comparación con la arquitectura

REST.

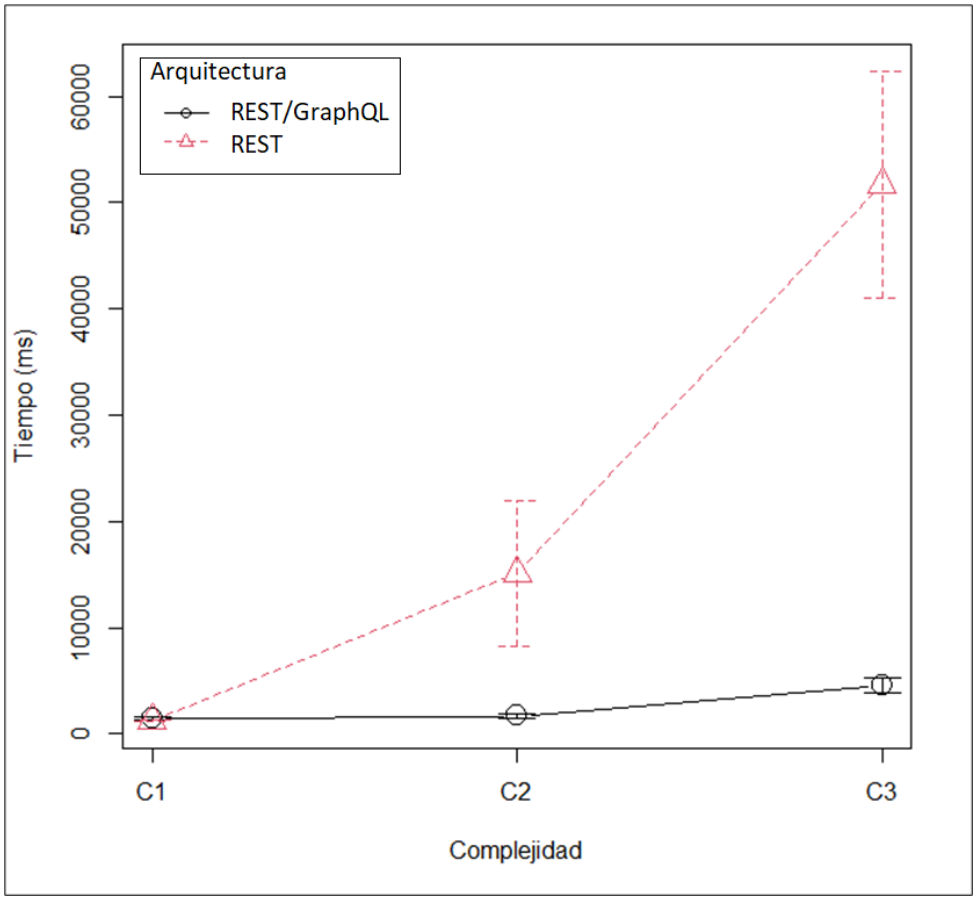


Figura 10.4: Diagrama BoxPlot de medias de la eficiencia

Análisis del tamaño de respuesta

En esta sección, se realiza el análisis del tamaño de respuesta de las consultas obtenidas de la ejecución del experimento. La Tabla §10.2 y la Figura §10.5 muestran la comparación de los tamaños medios de las respuestas medidos en Kilo bytes (Kb), en función a la complejidad de los casos de uso de las tareas experimentales. En donde, se observa que el tamaño medio de las respuestas de la arquitectura híbrida REST/-GraphQL es más pequeña que el tamaño medio de las respuestas que la arquitectura REST, dando como resultado un efecto de disminución del 18% en el promedio de respuestas.

Tabla 10.2: Comparativa del tamaño de respuesta entre REST/GraphQL y REST

Complejidad	REST	REST/GraphQL	Efecto
C1	26	27	4%
C2	145	184	21%
C3	176	250	30%
		Promedio	18%

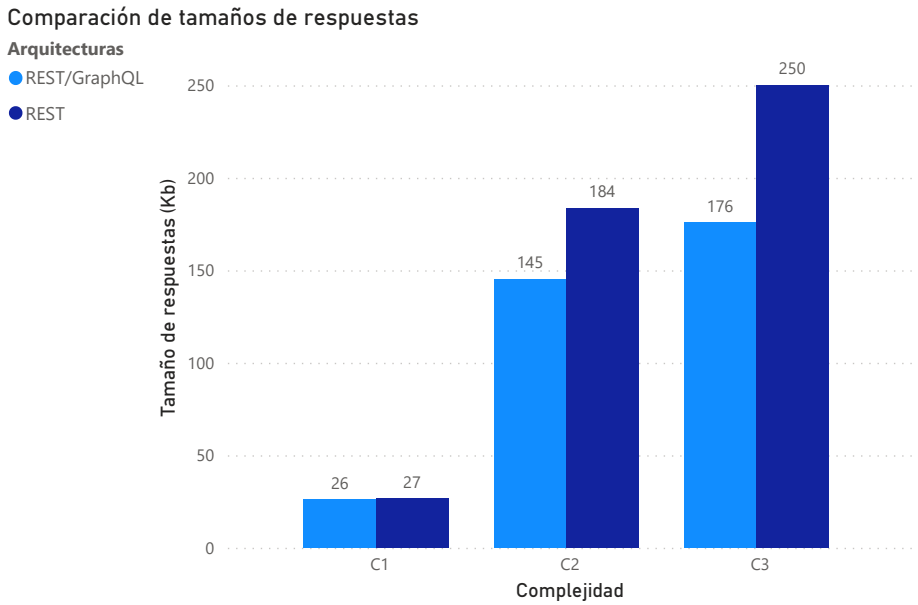


Figura 10.5: Comparación del tamaño de respuesta entre REST/GraphQL y REST

10.4 AMENAZAS A LA VALIDEZ

Las amenazas a la validez son un conjunto de situaciones, factores, debilidades y limitaciones que podrían interferir con la validez de los resultados del presente estudio empírico, por lo cual se analizan las posibles amenazas relevantes basándose en la clasificación propuesta por Wohlin *et al.* [186].

10.4.1 Validez interna

En el proceso del desarrollo de software, se construyó la API GraphQL (envoltorio) y la aplicación cliente de las tareas experimentales. En este sentido, el desarrollo de los programas se realizó utilizando la metodología de desarrollo ágil SCRUM, en donde se hizo un seguimiento interactivo entre los miembros del equipo de trabajo (investi-

gadores), y el apoyo técnico de un estudiante de pregrado de la Carrera de Ingeniería en Sistemas Computacionales de la Universidad Técnica del Norte. Además, se realizó pruebas de aceptación de cada caso de uso implementado en el entorno del despliegue del servicio y consumo de la API GraphQL que envolvió la API REST de Mendeley.

10.4.2 Validez externa

El experimento se llevó a cabo en el contexto de un laboratorio computacional, con un repositorio de Mendeley, lo que limita la capacidad de generalizar el comportamiento para las APIs de la práctica industrial. Esto fue en parte por diseño, ya que el alcance tácito de la propuesta primeramente fue identificar la factibilidad de realizar una arquitectura híbrida que envuelva un servicio disponible y luego comparar la eficiencia de esta arquitectura propuesta. Los casos de uso como tareas experimentales fueron planteados con una complejidad sencilla-media con el propósito de abarcar la práctica común del desarrollo de APIs. Por lo tanto, la generalización a tareas más complejas de la vida real es limitada, sin embargo, por tratarse de tareas básicas de manipulación de datos CRUD, es razonable esperar que también tenga el mismo efecto en tareas más complejas. Las tareas experimentales fueron equivalentes en complejidad, debido a que utilizaron los mismos casos de uso para su ejecución en los paradigmas REST y GraphQL.

10.4.3 Validez de constructo

Los constructos utilizados en la fase de ejecución del experimento, se diseñaron para medir las métricas establecidas para la variable dependiente del estudio. Estos constructos se definieron de manera consensuada por tres expertos en ingeniería de software, y luego validadas con pruebas de aceptación. El estudio no sufrió de un sesgo mono-operativo ya que se utilizó seis casos de uso diferentes, dos por cada nivel de complejidad de consulta de datos.

10.4.4 Validez de la conclusión

Para mitigar las amenazas en las conclusiones, se realizó una validación de pruebas estadísticas a los resultados para poder elegir y aceptar una de las hipótesis planteadas en el experimento y así fundamentar las conclusiones de este estudio empírico.

10.5 Conclusiones y trabajo futuro

10.5.1 Conclusiones

En este capítulo se validó el diseño experimental propuesto en el capitulo §7 parar comparar la eficiencia del rendimiento de una arquitectura híbrida REST/GraphQL contra una arquitectura REST en términos de la calidad de software. En este sentido, se contesta la pregunta de investigación $\mathbf{PI}_3$: ¿En qué escenarios resultan adecuadas las arquitecturas híbridas REST/GraphQL? En donde, los escenarios del uso de las arquitecturas híbridas REST/GraphQL se caracterizó con seis casos de uso de consumos de datos, y el efecto se midió con: la métrica *"tiempo medio de respuesta"* de la característica de calidad *Eficiencia de Rendimiento* del producto de software basada en la norma ISO/IEC 25022 [76] y el *tamaño de la respuesta*. Mediante esta caracterización se logró contestar las dos preguntas derivadas de la pregunta $\mathbf{PI}_3$:

$\mathbf{PI}_{3.1}$: ¿Cuál es el efecto del desarrollo de una arquitectura híbrida REST/GraphQL en la calidad del software?. El efecto del desarrollo de arquitecturas híbridas, por un lado, se pudo observar en la eficiencia de la ejecución de varias consultas de diferentes complejidades. El tiempo medio de respuesta de las consultas al API GraphQL que envuelve el API REST de Mendeley, es menor al tiempo medio de respuesta de las consultas que se hizo directamente al API REST de Mendeley. Por otro lado, el tamaño medio de las respuestas de las consultas realizadas al API GraphQL que envuelve el API REST de Mendeley, es menor que el tamaño medio de respuesta de las consultas que se hizo directamente al API REST de Mendeley.

Por tal razón, se puede concluir que las arquitecturas híbridas REST/GraphQL son más eficientes que las arquitecturas REST.

$\mathbf{PI}_{3.2}$: ¿En qué escenarios es más adecuado usar las arquitecturas híbridas REST/-GraphQL?. De acuerdo, con los resultados obtenidos en la sección §10.3 se puede observar que los escenarios más convenientes para usar las arquitecturas híbridas REST/GraphQL son cuando se realizan consultas más complejas, mientras más complejidad tengan las consultas, más eficientes se vuelven las arquitecturas híbridas. De igual forma, se nota el mismo comportamiento cuando se realizan consultas con más cantidad de datos, mientras más datos se consulte las arquitecturas híbridas muestran más eficiencia a comparación de las arquitecturas REST.

En este sentido, se puede concluir que las arquitecturas híbridas REST/GraphQL

son convenientes usar cuando se presentan escenarios de consultas complejas o con gran cantidad de datos.

10.5.2 Trabajo futuro

Como trabajo futuro, se plantea complementar este experimento con nuevos casos de uso que manipulen operaciones como mutaciones al API Mendeley. Además, se incentiva a la comunidad científica a realizar estudios similar con arquitecturas híbridas con otras APIs públicas de preferencia que estén aplicadas en la práctica industrial de la ingeniería de software. También, se promueve la experimentación para comparar arquitecturas híbridas REST y GraphQL con otras características de la calidad del producto de software. Finalmente, se promueve estudios de arquitecturas híbridas de GraphQL con otras tecnologías como SOAP, SPARQL o Falcor[1].

10.6 Resumen

En este capítulo, se ejecutó el diseño experimental propuesto en el capítulo §7, en donde se analizó el proceso de desarrollo del software, y comparó la eficiencia de una arquitectura híbrida REST/GraphQL contra una arquitectura REST con respecto a la calidad del software, desde un punto de vista del investigador en un contexto controlado de un laboratorio computacional. Con la ejecución del experimento y análisis de sus resultados se logró responder la pregunta $\mathbf{PI}_3$: ¿En qué escenarios resultan adecuadas las arquitecturas híbridas REST/GraphQL?, mediante la contestación de sus preguntas derivadas: $\mathbf{PI}_{3.1}$: ¿Cuál es el efecto del desarrollo de una arquitectura híbrida REST/GraphQL en la calidad del software?. Los resultados mostraron que el software desarrollado con arquitecturas híbridas REST/GraphQL presenta mejor calidad en eficiencia de consultas de datos, que el software desarrollado con la arquitectura REST. $\mathbf{PI}_{3.2}$: ¿En qué escenarios es más adecuado usar las arquitecturas híbridas REST/GraphQL?. Los resultados muestran que los escenarios más adecuados para usar arquitecturas híbridas REST/GraphQL es cuando se realiza consultas complejas con gran cantidad de datos.

[1] https://netflix.github.io/falcor

Acuerdos de nivel de servicio para APIs GraphQL: Validación

El único modo de hacer un gran trabajo es amar lo que haces.

Steve Jobs (1955 – 2011),
Empresario y magnate estadounidense

En este capítulo, se valida el estándar SLA4GraphQL *que describe como modelar y operativizar acuerdos de nivel de servicios (SLA) para APIs GraphQL propuesto en el capítulo §8; con el propósito de contestar la pregunta de investigación* $\boldsymbol{PI_4}$*: ¿En qué medida es posible modelar los acuerdos de APIs GraphQL?.*

En la sección §11.1, se presenta la introducción y antecedentes del capítulo. En la sección §11.2, se diseña la metodología de validación de la propuesta compuesta por: diseño; planificación, recopilación, y análisis de datos recogidos; e informe conjunto. En la sección §11.3 se realiza la evaluación de viabilidad de la propuesta. En la sección §11.3 se realiza la evaluación de flexibilidad de la propuesta. En la sección §11.5 se realiza la evaluación de idoneidad funcional de la propuesta. En la sección §11.6 se presenta las conclusiones del estudio de validación. Por último, la sección §6.3 resume el capítulo.

11.1 Introducción

En el capítulo §8 se propuso el estándar *SLA4GraphQL* que describe cómo modelar y operativizar acuerdos de nivel de servicios (SLA) para APIs GraphQL, el cual consta de: una especificación para modelar documentos SLA para APIs GraphQL; un ciclo de vida de desarrollo de APIs GraphQL basadas en SLA; y la especificación de la herramienta *SLA GraphQL Manager* para validar los documentos SLA como parte de su operativización. Por lo cual, el objetivo de este capítulo es validar la propuesta *SLA4GraphQL* y así contestar la pregunta de investigación **PI**$_4$: ¿En qué medida es posible modelar los acuerdos de APIs GraphQL?, en este sentido, para relacionar esta pregunta al estándar *SLA4GraphQL*, se deriva las siguientes preguntas de investigación:

- **PI**$_{4.1}$: Viabilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar los acuerdos de nivel de servicio (SLA) de un caso en concreto?.
- **PI**$_{4.2}$: Flexibilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) en distintos casos concretos?.
- **PI**$_{4.3}$: Idoneidad Funcional. ¿En qué medida el estándar *SLA4GraphQL* proporciona funciones que cumplan con necesidades establecidas en condiciones específicas?.

Para contestar **PI**$_{4.1}$, se realiza la evaluación de viabilidad, mediante la instanciación de la propuesta para modelar el documento SLA de una API GraphQL Pública. Luego, para contestar **PI**$_{4.2}$, se realiza la evaluación de flexibilidad, mediante la instanciación de la propuesta para modelar el documento SLA en varias APIs GraphQL Públicas. Para finalizar con la validación de la propuesta, se contesta **PI**$_{4.3}$ con la evaluación de la idoneidad funcional en las instanciaciones realizadas en las evaluaciones de viabilidad y flexibilidad.

11.2 Diseño de la validación

El diseño de la validación de la propuesta *SLA4GraphQL*, está basado en el enfoque DSR [119] y los casos de estudios [139], en donde, se identifican tres fases principales: i) Diseño del estudio de caso; ii) Planificación, recopilación y análisis; y iii) Análisis e informe conjunto; tal como se muestran en la Figura §11.1.

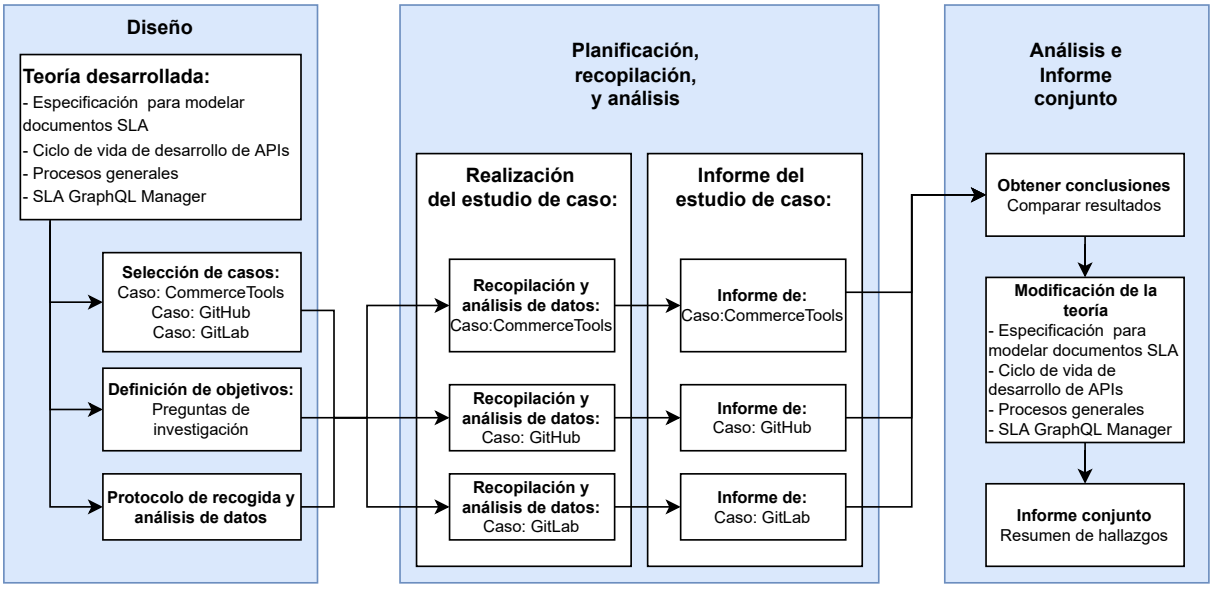


Figura 11.1: Diseño de la validación de *SLA4GraphQL*

11.2.1 Diseño

En esta primera fase se describe el contexto de la investigación, en donde, basados en Runeson *et al.* [139] se determina las siguientes actividades: Teoría desarrollada; Selección de casos; Definición de objetivos; y Protocolo de recogida y análisis de datos. Las cuales se describen a continuación:

1. *Teoría desarrollada*. esta sección se hace referencia al capítulo §8 que presenta la especificación de *SLA4GraphQL* que propone un estándar para modelar y operativizar acuerdos de nivel de servicio (SLA) para APIs GraphQL, el cual se compone por: i) Especificación para modelar documentos SLA; ii) Ciclo de vida de desarrollo de APIs; iv) Procesos generales de operativización del estándar; y iv) Descripción de la herramienta SLA GraphQL Manager.

2. *Selección de casos*. Para evaluar la teoría desarrollada, se seleccionaron tres APIs GraphQL públicas, en las cuales se aplicará el modelado de acuerdos de nivel de servicios basado en *SLA4GraphQL*. Para la selección, se revisó la documentación de las limitaciones de 35 APIs públicas recuperadas de la lista colectiva de "APIs GraphQL oficiales"[1] de la plataforma APIs.guru[2]. En donde, se escogió a

[1] https://github.com/APIs-guru/graphql-apis
[2] https://apis.guru/

las siguientes APIs, las cuales, presentaron la documentación de limitaciones más detalladas y adecuadas para aplicar el estándar:

- *CommerceTools*: es una plataforma de comercio electrónico construida sobre principios tecnológicos modernos (microservicios , API-first, Cloud-native, y Headless) [25]. La documentación de las limitaciones de la API está disponible en: `https://docs.commercetools.com/api/limits`.
- *GitHub*: es una plataforma de alojamiento de código para el control de versiones y la colaboración. El control de versiones es una forma de guardar los cambios a lo largo del tiempo sin sobrescribir las versiones anteriores. La colaboración se presenta cuando varios desarrolladores pueden trabajar en un repositorio de Git, los cuales, tiene una copia de ese repositorio completo [52]. La documentación de las limitaciones de la API está disponible en [51]: `https://docs.github.com/en/graphql/overview/resource-limitations`.
- *GitLab*: es una plataforma DevOps de código abierto basado en Git que proporciona control de versiones de repositorios abiertos y privados, y crea un flujo de trabajo de software DevOps. Además, ofrece alojamiento de wikis y un sistema de seguimiento de errores [53]. La documentación de las limitaciones de la API está disponible en: `https://docs.gitlab.com/ee/api/graphql/#limits`.

3. *Definición de objetivos*. El objetivo principal de este capítulo es validar el estándar *SLA4GraphQL* propuesto en el capítulo §8, y así contestar la pregunta de investigación $\mathbf{PI}_4$: ¿En qué medida es posible modelar los acuerdos de APIs GraphQL?. Por lo cual, se evaluará y analizará los siguientes aspectos: viabilidad, flexibilidad e idoneidad funcional de la propuesta. En este sentido, se concreta la relación de $\mathbf{PI}_4$ con el estándar *SLA4GraphQL* y los aspectos de evaluación en las siguientes preguntas de investigación (las cuales se mencionaron también en la sección de introducción §11.1):

   - $\mathbf{PI}_{4.1}$: Viabilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar los acuerdos de nivel de servicio (SLA) de un caso en concreto?.
   - $\mathbf{PI}_{4.2}$: Flexibilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) en distintos casos concretos?.
   - $\mathbf{PI}_{4.3}$: Idoneidad Funcional. ¿En qué medida el estándar *SLA4GraphQL* proporciona funciones que cumplan con necesidades establecidas en condiciones específicas?.

4. *Protocolo de recogida y análisis de datos*. Se trata de identificar las fuentes para recopilar los datos y definir los procedimientos para llevar a cabo el análisis de los resultados. Para este estudio, se estableció el siguiente protocolo para este propósito: i) revisar las limitaciones de las APIs públicas escogidas; ii) modelar en documentos SLA las limitaciones de las APIs públicas de acuerdo a la especificación del estándar *SLA4GraphQL*; iii) validar el documento SLA de las APIs públicas mediante la herramienta *SLA GraphQL Manager*; y iv) Evaluar la idoneidad funcional de los acuerdos modelados con las limitaciones expuestas en la documentación de las APIs.

11.2.2 Planificación, recopilación y análisis

En esta fase se aplica el *protocolo de recogida y análisis de datos* de los *casos seleccionados* en la fase anterior, y se redacta el informe. Para lo cual, el estudio para cada caso se realiza con las siguientes actividades: i) recopilación de datos; ii) análisis de los datos recogidos; y iii) elaboración del informe del estudio del caso. A continuación, se detalla los pasos de las actividades:

1. *Proceso de recopilación de datos*. para esta actividad se definen los siguientes pasos:
   - Se recopila la información de las limitaciones desde la documentación de la API.
   - Se realiza la configuración de acceso a la API pública y se verifica las limitaciones con las rutas de consultas y mutaciones basándose en el esquema de la API GraphQL.
   - Se configura aspectos generales del modelado en el documento SLA, con la información de la infraestructura de la API GraphQL pública.
   - Se modela acuerdos específicos de las limitaciones de la API, en donde, en la configuración de cuotas o tarifas documento SLA se modela una entrada por cada limitación especificada en la documentación de la API GraphQL pública.
2. *Análisis de los datos recogidos*. Luego de haber recopilado los datos de las limitaciones en los documentos SLA de las APIs GraphQL públicas, se analizan estos datos de forma cualitativa y cuantitativa. Para aquello se realiza dos comprobaciones:

- Se valida que el esquema del documento SLA, y los acuerdos específicos en cuotas y tarifas estén de acuerdo a la especificación del estándar *SLA4GraphQL*. Esta validación se la realiza utilizando la herramienta *SLA GraphQL Manager*.
- Se realiza la evaluación de idoneidad funcional del estándar *SLA4GraphQL*, utilizando las métricas de la ISO/IEC 25023 [77]. Básicamente, se evalúa qué limitaciones descritas en la documentación de las APIs públicas se pudo o no pudo modelar con la especificación del estándar.

3. *Informe del estudio del caso*. Es la redacción del informe de cada caso.

11.2.3 Análisis e informe conjunto

Luego de haber analizado los tres casos en su entorno, el siguiente paso es integrar los resultados obtenidos de manera que se puedan analizarlos de manera global y así lograr contestar las preguntas de investigación establecidas en este capítulo de validación, y por ende obtener las conclusiones del estudio. A continuación, los pasos a seguir son:

1. *Obtener conclusiones*. Se comparan los resultados obtenidos de la aplicación de la propuesta en los tres casos; y se extraen conclusiones cruzadas. Las conclusiones se presentan en la sección §11.6.
2. *Modificación de la teoría*. Al momento de instanciar la propuesta *SLA4GraphQL* en los casos, se descubrió varios aspectos de mejora, los cuales, se incorporaron en la teoría expuesta en el capítulo §8 posibilitando su evolución desde una versión inicial a la versión final que está reportada en dicho capítulo.
3. Informe conjunto. Para finalizar, se escribe el informe a lo largo de este capítulo.

11.3 Evaluación de Viabilidad

En esta sección se realiza la evaluación de viabilidad de la propuesta, en donde, se comprueba la factibilidad técnica y administrativa mediante la instanciación del estándar en un caso en concreto. La instanciación consiste en realizar el modelado del documento de acuerdos de nivel de servicio (SLA) para una API GraphQL pública, basado en la propuesta *SLA4GraphQL*.

11.3.1 Definición de los aspectos de viabilidad

El caso escogido para la instanciación del estándar *SLA4GraphQL* es la API GraphQL de CommerceTools; debido que tiene la documentación de limitaciones más extensa y detallada, prevista en el apartado de selección de casos de la sección §11.2.1. Para describir el documento SLA se utiliza la estructura propuesta en la sección §8.2.1. Los resultados se analizan para evaluar el aspecto de viabilidad de la propuesta y contestar a la pregunta $\mathbf{PI}_{4.1}$. Los atributos de viabilidad que se utilizarán en la evaluación han sido adaptados del estudio de Solari [152], los cuales se listan a continuación:

- Obtención de la instancia. Indica si se puede obtener o no el documento SLA, a partir de la información recogida en la documentación de limitaciones de la API GraphQL de CommerceTools.
- Idoneidad funcional. Evalúa el grado de funcionalidad del estándar para cumplir con las necesidades especificadas.

11.3.2 Caso: API GraphQL de CommerceTools

Sitio oficial de la plataforma de CommerceTools: `https://commercetools.com/`

Recopilación de datos

Como primer paso se recopila las 33 limitaciones expuestas en la documentación de la API CommerceTools disponibles en: `https://docs.commercetools.com/api/limits`.

Luego, se accedió a la API pública para verificar las rutas de consultas y mutaciones en el esquema de la API GraphQL CommerceTools descritas en la documentación de las limitaciones. Para lo cual, se realizó los siguientes pasos:

- Crear un usuario de prueba para acceder a la plataforma (prueba gratuita por 60 días), en el siguiente enlace: `https://commercetools.com/free-trial`.
- Acceder a la plataforma de CommerceTools, en el siguiente enlace: `https://mc.us-central1.gcp.commercetools.com/login`.
- Crear un nuevo proyecto en la plataforma.

- Crear un nuevo cliente del API. Acceder al menú: Settings → Developer settings → Create new API client. En este paso, se obtiene las configuraciones de infraestructura y credenciales de acceso como cliente del API de CommerceTools, además, se obtiene el código para generar un TOKEN de autenticación para el acceso al API (ver ejemplo de código cURL en la Figura §11.2).

```
C:\>curl https://auth.us-central1.gcp.commercetools.com/oauth/token --basic --user "[redacted]" -X POST -d "grant_type=client_credentials&scope=manage_project:sales-test"
{"access_token":"JrFWGBp1tmeB_ShrppIyqfAjqU5kaLPa","token_type":"Bearer","expires_in":172800,"scope":"manage_project:sales-test"}
```

Figura 11.2: Solicitud y respuesta de generación de TOKEN de acceso.

- Acceder a la interface cliente *GraphiQL Explorer* de la API GraphQL de CommerceTools, mediante al siguiente enlace: `https://impex.us-central1.gcp.commercetools.com/`, y luego escoger la opción "GraphQL IDE".
- Acceder a la información del esquema de la API GraphQL en la sección "Documentation Explorer", tal como se muestra en la Figura §11.3.

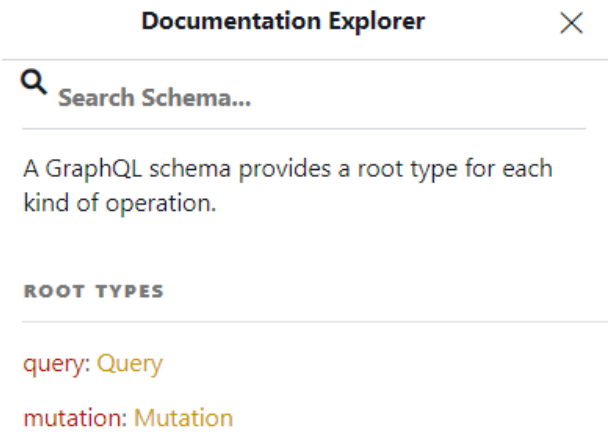


Figura 11.3: Interface cliente GraphiQL Explorer de la API CommerceTools.

Como siguiente paso, se creó la instanciación del documento SLA (.yaml) y se configuró aspectos generales como contexto, infraestructura y métricas, ver Figura §11.4.

Finalmente, se modeló en los componentes de *Cuotas y Tarifas* del documento SLA los acuerdos de las 33 limitaciones de la API CommerceTools, en donde, se muestra cada limitación de la API, con su acuerdo equivalente modelado con el estándar

```yaml
context:
  id: commercetools-sample
  type: plans
  version: v1.0
  api: https://api.us-central1.gcp.commercetools.com/jquina/graphql
  provider: commercetools
infrastructure:
  supervisor: http://sla-graphql-manager/supervisor/
  monitor: http://sla-graphql-manager/monitor/
metrics:
  json-size:
    type: "integer"
    description: "JSON payload request in bytes"
  elements:
    type: "integer"
    description: "Elements returned by a given request"
  update_actions:
    type: "integer"
    description: "the maximum number of update actions within a single request"
  facets:
    type: "integer"
    description: "The number of terms per facet"
  search_characters:
    type: "integer"
    description: "Search results are based on the first characters of the full-text search query parameter only"
  refresh_tokens:
    type: "integer"
    description: "Limit on the number of refresh tokens"
  audit_log_records:
    type: "integer"
    description: "Audit Log Records"
  complexity:
    type: "integer"
    description: "Complexity of query"
```

Figura 11.4: CommerceTools: Documento SLA - configuraciones generales

SLA4GraphQL. Cabe mencionar que por ser dicácticos, en cada limitación se repetirá los dos primeros niveles del acuerdo, es decir, primer nivel(quotas o rates), y segundo nivel (queries o mutations):

- Tamaño del documento JSON (*JSON document size*). Cada documento JSON persistente a través del extremo de la API no debe exceder los 16 mega bytes.

  La Figura §11.5 muestra el acuerdo para la limitación *JSON document size*.

```yaml
quotas:
  mutations:
  //*:
    json-size:
      max: 16384
      over: request
      description: "https://docs.commercetools.com/api/limits#json-document-size"
```

Figura 11.5: CommerceTools: Limitación ”JSON document size”

- Tamaño del contenido del campo (*Field content size*). Por motivo de rendimiento, el tamaño máximo de un campo de búsqueda dentro de una AttributeDefinition no puede superar los 10.922 caracteres.

  La Figura §11.6 muestra el acuerdo para la limitación *Field content size*.

```yaml
quotas:
  queries:
    //AttributeDefinition[@*]:
      _string-length:
        max: 10922
        over: request
        description: "https://docs.commercetools.com/api/limits#field-content-size"
```

Figura 11.6: CommerceTools: Limitación "Field content size"

- Slugs. Un Slug debe coincidir con el patrón $[a-zA-Z0-9\_\-]\{2,256\}$, por lo tanto, están limitados a 256 caracteres.

  La Figura §11.7 muestra el acuerdo para la limitación *Slugs*.

```yaml
quotas:
  mutations:
    //*/slug:
      _string-length:
        max: 256
        over: request
        description: "https://docs.commercetools.com/api/limits#slugs"
```

Figura 11.7: CommerceTools: Limitación "Slugs"

- Actualizar acciones por solicitud (*Update actions per request*). Por motivo de rendimiento, la cantidad máxima de acciones de actualización dentro de una sola solicitud está limitada a 500.

  La Figura §11.8 muestra el acuerdo para la limitación *Update actions per request*.

```yaml
quotas:
  mutations:
    //*:
      update_actions:
        max: 500
        over: request
        description: "https://docs.commercetools.com/api/limits#update-actions-per-request"
```

Figura 11.8: CommerceTools: Limitación "Update actions per request"

- Consultas (*Queries*). Dentro de una consulta, se pueden recuperar hasta 500 elementos. Cuando se usa la paginación, el desplazamiento máximo es 10.000. El campo *total* de PagedQueryResult se limita al máximo *offset* cuando los resultados se filtran con un predicado de consulta.

  La Figura §11.9 muestra los acuerdos para la limitación *Queries*.

```yaml
quotas:
  queries:
    //*:
      elements:
        max: 500
        over: response
        description: "https://docs.commercetools.com/api/limits#queries"
    //@limit:
      _number:
        max: 500
        over: request
        description: "https://docs.commercetools.com/api/limits#search-and-facets"
    //@offset:
      _number:
        max: 10000
        over: request
        description: "https://docs.commercetools.com/api/limits#search-and-facets"
```

Figura 11.9: CommerceTools: Limitación ”Queries”

- Búsqueda y facetas (*Search and facets*). Por motivos de rendimiento, el tamaño máximo de un campo de búsqueda dentro de una AttributeDefinition no puede superar los 10.922 caracteres. (Nota: Esta limitación ya se abarca en el modelado del acuerdo de la figura §11.6)

  Dentro de una solicitud de búsqueda, se pueden recuperar hasta 500 elementos. En paginación, el desplazamiento máximo es 10.000. (Nota: Esta limitación ya se abarca en el modelado de los acuerdos de la figura §11.9)

  El número de términos por faceta está limitado a 200.

  Los resultados de la búsqueda se basan únicamente en los primeros 256 caracteres del parámetro de consulta de búsqueda de texto completo.

  La Figura §11.10 muestra los acuerdos para la limitación *Search and facets*.

```yaml
quotas:
  queries:
    //*:
      search_characters:
        max: 256
        over: request
        description: "https://docs.commercetools.com/api/limits#search-and-facets"
      facets:
        max: 200
        over: request
        description: "https://docs.commercetools.com/api/limits#search-and-facets"
```

Figura 11.10: CommerceTools: Limitación "Search and facets"

- Categorías (*Categories*). Se puede crear un número máximo de 10.000 categorías.

  La Figura §11.11 muestra el acuerdo para la limitación *Categories*.

```yaml
quotas:
  mutations:
    /createCategory/id:
      _count:
        max: 10000
        over: model
        description: "https://docs.commercetools.com/api/limits#categories"
```

Figura 11.11: CommerceTools: Limitación "Categories"

- Variantes del producto (*Product Variants*). Se puede especificar un número máximo de 100 variantes en un Producto.

  La Figura §11.12 muestra los acuerdos para la limitación *Product Variants*.

```yaml
quotas:
  mutations:
    /createProduct/masterData/current/variants:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#product-variants"
    /updateProduct/masterData/current/variants:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#product-variants"
```

Figura 11.12: CommerceTools: Limitación "Product Variants"

- Precios (Prices). Se puede especificar un número máximo de 100 precios en un ProductVariant.

La Figura §11.13 muestra los acuerdos para la limitación *Prices*.

```yaml
quotas:
  mutations:
    /createProduct/masterData/current/variants/prices:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#prices"
    /updateProduct/masterData/current/variants/prices:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#prices"
```

Figura 11.13: CommerceTools: Limitación ”Prices”

- Descuentos en productos (*Product Discounts*). El número máximo de ProductDiscounts que pueden estar activos al mismo tiempo es 500.

  La Figura §11.14 muestra los acuerdos para la limitación *Product Discounts*.

```yaml
quotas:
  mutations:
    /createProductDiscount/isActive:
      _true:
        max: 500
        over: model
        description: "https://docs.commercetools.com/api/limits#product-discounts"
    /updateProductDiscount/isActive:
      _true:
        max: 500
        over: model
        description: "https://docs.commercetools.com/api/limits#product-discounts"
```

Figura 11.14: CommerceTools: Limitación ”Product Discounts”

Posible limitación en el modelado: para modelar de manera más explícita los acuerdos de la figura §11.14 se debería considerar dos funciones XPath; la función *_true* para referir a los valores verdaderos del campo isActive; y la función *_count* para contar los registros de ProductDiscounts. En este caso, en la implementación del SLA se podría asumir la operativización de estos acuerdos, mediante la ejecución implícita de la función *_count* cuando el acuerdo referencie a la función *_true*.

- Descuentos en carrito (*Cart Discounts*). El número de CartDiscounts activos que no requieren un código de descuento (isActive=true y requiresDiscountCode=false) está limitado a 100.

Limitación en el modelado: para modelar este acuerdo, se debería referir como métricas a los valores de los campos isActive=true y requiresDiscountCode=false. Por lo cual, se identifica como limitaciones del modelado: i) No se puede referir a los valores de los elementos en las rutas o métricas. ii) No se puede definir a métricas compuestas de dos o más condicionales.

- Carritos (*Carts*). El número máximo de carritos que se pueden agregar a un proyecto es 10'000.000.

La Figura §11.15 muestra el acuerdo para la limitación *Carts*.

```
quotas:
  mutations:
    /createCart/id:
      _count:
        max: 10000000
        over: model
        description: "https://docs.commercetools.com/api/limits#carts"
```

Figura 11.15: CommerceTools: Limitación "Carts"

- Listas de compras (*Shopping Lists*). Una ShoppingList puede contener hasta 100 elementos de línea (lineItems) y hasta 100 elementos de línea de texto (textLineItems). El número máximo de Listas de Compra que se pueden agregar a un proyecto es 10'000.000.

La Figura §11.16 muestra los acuerdos para la limitación *Shopping Lists*.

```yaml
quotas:
  mutations:
    /createShoppingList/lineItems:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#shopping-lists"
    /createShoppingList/textLineItems:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#shopping-lists"
    /updateShoppingList/lineItems:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#shopping-lists"
    /updateShoppingList/textLineItems:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#shopping-lists"
    /createShoppingList/id:
      _count:
        max: 10000000
        over: model
        description: "https://docs.commercetools.com/api/limits#shopping-lists"
```

Figura 11.16: CommerceTools: Limitación ”Shopping Lists”

- Zonas (*Zones*). El número máximo de zonas que se pueden agregar a un proyecto es 100.

  La Figura §11.17 muestra el acuerdo para la limitación *Zones*.

```yaml
quotas:
  mutations:
    /CreateZone/id:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#zones"
```

Figura 11.17: CommerceTools: Limitación ”Zones”

- Categorías de impuestos (*Tax Categories*). El número máximo de categorías de impuestos que se pueden agregar a un proyecto es 100.

  La Figura §11.18 muestra el acuerdo para la limitación *Tax Categories*.

- Métodos de envío (*Shipping Methods*). El número máximo de métodos de envío que se pueden agregar a un proyecto es 100.

  La Figura §11.19 muestra el acuerdo para la limitación *Shipping Methods*.

```yaml
quotas:
  mutations:
    /createTaxCategory/id:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#tax-categories"
```

Figura 11.18: CommerceTools: Limitación "Tax Categories"

```yaml
quotas:
  mutations:
    /createShippingMethod/id:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#shipping-methods"
```

Figura 11.19: CommerceTools: Limitación "Shipping Methods"

- Grupos de clientes (*Customer Groups*). El número máximo de grupos de clientes que se pueden agregar a un proyecto es 1.000.

  La Figura §11.20 muestra el acuerdo para la limitación *Customer Groups*.

```yaml
quotas:
  mutations:
    /createCustomerGroup/id:
      _count:
        max: 1000
        over: model
        description: "https://docs.commercetools.com/api/limits#customer-groups"
```

Figura 11.20: CommerceTools: Limitación "Customer Groups"

- Clientes (*Customers*). El número máximo de Clientes que se pueden agregar a un proyecto es 10'000.000.

  La Figura §11.21 muestra el acuerdo para la limitación *Customers*.

```yaml
quotas:
  mutations:
    /customerSignUp/customer:
      _count:
        max: 10000000
        over: model
        description: "https://docs.commercetools.com/api/limits#customers"
```

Figura 11.21: CommerceTools: Limitación "Customers"

- Edición de pedidos (*Order Edits*). El número máximo de ediciones de pedidos que pueden existir por proyecto es 100.000.

  La Figura §11.22 muestra el acuerdo para la limitación *Order Edits*.

```yaml
quotas:
  mutations:
    /createOrderEdit/id:
      _count:
        max: 100000
        over: model
        description: "https://docs.commercetools.com/api/limits#order-edits"
```

Figura 11.22: CommerceTools: Limitación "Order Edits"

- Búsqueda de pedidos *BETA* (*Order Search*). Durante el período beta, el índice de búsqueda de pedidos tiene una capacidad de 10'000.000 pedidos en el proyecto.

  Nota: No se modeló este acuerdo debido a que la limitación está en fase *BETA*, y la documentación no está clara y aparentemente incompleta.

- Carritos - códigos de descuento (*Carts - discount codes*). El número de códigos de descuento en un carrito está limitado a 10.

  La Figura §11.23 muestra los acuerdos para la limitación *Carts - discount codes*.

```yaml
quotas:
  mutations:
    /createCart/discountCodes:
      _count:
        max: 10
        over: model
        description: "https://docs.commercetools.com/api/limits#carts-1"
    /updateCart/discountCodes:
      _count:
        max: 10
        over: model
        description: "https://docs.commercetools.com/api/limits#carts-1"
```

Figura 11.23: CommerceTools: Limitación "Carts - discount codes"

- Códigos de descuento (*Discount Codes*). El número de descuentos de carrito en un código de descuento está limitado a 10.

  La Figura §11.24 muestra los acuerdos para la limitación *Discount Codes*.

- Tiendas (*Stores*). El número de Tiendas está limitado a 50.000.

  La Figura §11.25 muestra el acuerdo para la limitación *Stores*.

```yaml
quotas:
  mutations:
    /createDiscountCode/cartDiscounts:
      _count:
        max: 10
        over: model
        description: "https://docs.commercetools.com/api/limits#discount-codes"
    /updateDiscountCode/cartDiscounts:
      _count:
        max: 10
        over: model
        description: "https://docs.commercetools.com/api/limits#discount-codes"
```

Figura 11.24: CommerceTools: Limitación ”Discount Codes”

```yaml
quotas:
  mutations:
    /createStore/id:
      _count:
        max: 50000
        over: model
        description: "https://docs.commercetools.com/api/limits#stores"
```

Figura 11.25: CommerceTools: Limitación ”Stores”

- Canales de suministro de inventario de la tienda (*Store's Inventory Supply Channels*). El número de canales de suministro de inventario configurados para una tienda está limitado a 100.

  La Figura §11.26 muestra los acuerdos para la limitación *Store's Inventory Supply Channels*.

```yaml
quotas:
  mutations:
    /createStore/supplyChannels:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#stores-inventory-supply-channels"
    /updateStore/supplyChannels:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#stores-inventory-supply-channels"
```

Figura 11.26: CommerceTools: Limitación ”Store's Inventory Supply Channels”

- Canales de distribución de productos de la tienda (*Store's Product Distribution Channels*). El número de canales de distribución de productos configurados para una tienda está limitado a 100.

  La Figura §11.27 muestra los acuerdos para la limitación *Store's Product Distribu-*

tion Channels.

```
quotas:
  mutations:
    /createStore/distributionChannels:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#stores-product-distribution-channels"
    /updateStore/distributionChannels:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#stores-product-distribution-channels"
```

Figura 11.27: CommerceTools: Limitación ”Store’s Product Distribution Channels”

- Selección de productos de la tienda *BETA*(*Store’s Product Selections*). Durante el período de prueba pública, la cantidad de selecciones de productos configuradas para una tienda está limitada a 100.

  La Figura §11.28 muestra los acuerdos para la limitación *Store’s Product Selections*.

```
quotas:
  mutations:
    /createStore/productSelections:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#stores-product-selections"
    /updateStore/productSelections:
      _count:
        max: 100
        over: model
        description: "https://docs.commercetools.com/api/limits#stores-product-selections"
```

Figura 11.28: CommerceTools: Limitación ”Store’s Product Selections”

- Actualización de Tokens (*Refresh tokens*). El número de tokens de actualización está limitado a 10 millones.

  La Figura §11.29 muestra el acuerdo para la limitación *Refresh tokens*.

  Posible limitación en el modelado: debido a que esta limitación aparentemente no referencia a una ruta específica del esquema del API. En este sentido, para este acuerdo se asumió la expresión XPath que selecciona todas las rutas ”//*”. Sin embargo, se debería discutir la viabilidad de modelar acuerdos que no hagan referencia a una ruta específica.

```yaml
quotas:
  mutations:
    //*:
      refresh_tokens:
        max: 10000000
        over: request
        description: "https://docs.commercetools.com/api/limits#refresh-tokens"
```

Figura 11.29: CommerceTools: Limitación "Refresh tokens"

- Objetos personalizados (*Custom Objects*). Se puede crear un máximo de 20'000.000 de objetos personalizados.

  La Figura §11.30 muestra el acuerdo para la limitación *Custom Objects*.

```yaml
quotas:
  mutations:
    /createOrUpdateCustomObject/id:
      _count:
        max: 20000000
        over: model
        description: "https://docs.commercetools.com/api/limits#custom-objects"
```

Figura 11.30: CommerceTools: Limitación "Custom Objects"

- Tipos de productos (*Product Types*). Se puede crear un máximo de 1.000 tipos de productos.

  La Figura §11.31 muestra el acuerdo para la limitación *Product Types*.

```yaml
quotas:
  mutations:
    /createProductType/id:
      _count:
        max: 1000
        over: model
        description: "https://docs.commercetools.com/api/limits#product-types"
```

Figura 11.31: CommerceTools: Limitación "Product Types"

- Extensiones (*Extensions*). Se pueden crear un máximo de 25 Extensiones por proyecto.

  La Figura §11.32 muestra el acuerdo para la limitación *Extensions*.

- Suscripciones (*Subscriptions*). Se pueden crear un máximo de 50 suscripciones por proyecto.

  La Figura §11.33 muestra el acuerdo para la limitación *Subscriptions*.

```yaml
quotas:
  mutations:
    /createExtension/id:
      _count:
        max: 25
        over: model
        description: "https://docs.commercetools.com/api/limits#extensions"
```

Figura 11.32: CommerceTools: Limitación "Extensions"

```yaml
quotas:
  mutations:
    /createSubscription/id:
      _count:
        max: 50
        over: model
        description: "https://docs.commercetools.com/api/limits#subscriptions"
```

Figura 11.33: CommerceTools: Limitación "Subscriptions"

- Registros de auditoría (*Audit Log Records*). De forma predeterminada, se almacena un máximo de 100.000 registros por año.

  La Figura §11.34 muestra el acuerdo para la limitación *Audit Log Records*.

```yaml
rates:
  mutations:
    //*:
      audit_log_records:
        max: 100000
        over: model
        period: yearly
        description: "https://docs.commercetools.com/api/limits#audit-log-records"
```

Figura 11.34: CommerceTools: Limitación "Audit Log Records"

Posible limitación en el modelado: al igual que *Refresh tokens*, aparentemente no se refiere a una ruta específica del esquema para operativizar esta limitación. En este sentido, se asumió la expresión XPath que selecciona todas las rutas "//*" para este acuerdo. Sin embargo, se debería discutir la viabilidad de definir acuerdos que no hagan referencia a rutas específicas.

- GraphQL - complejidad de consulta (*GraphQL - query complexity*). Si una consulta tiene una puntuación de complejidad superior o igual a 20.000, no se ejecutará y se devolverá un código de error QueryComplexityLimitExceeded.

  La Figura §11.35 muestra el acuerdo para la limitación *GraphQL - query complexity*.

```yaml
quotas:
  queries:
    //*:
      complexity:
        max: 20000
        over: request
        description: "https://docs.commercetools.com/api/limits#graphql"
```

Figura 11.35: CommerceTools: Limitación "GraphQL - query complexity"

Análisis de los datos recogidos

Luego de recopilar los datos de las limitaciones de la API CommerceTools en la instanciación del documento SLA modelado en la sección anterior, se procede a analizarlo. Por un lado, se realiza un análisis cualitativo, en donde, se valida el esquema del documento SLA, y los acuerdos modelados en Cuotas y Tarifas del documento SLA, de acuerdo a la especificación descrita en la sección §8.2.1; la validación se la realiza mediante el uso de la herramienta *SLA GraphQL Manager*. Por otro lado, se analiza cuantitativamente el documento SLA, mediante una evaluación de la idoneidad funcional, basada en las métricas de la ISO/IEC 25023 [77]. A continuación, se detallan los análisis realizados:

1. *Análisis cualitativo*. Este análisis se lo realiza mediante la herramienta *SLA GraphQL Manager*, la cual, automatiza los procesos de validación y registro del documento SLA descritos en las figuras §8.5 y §8.6. A continuación, se detalla los pasos de uso de la herramienta:

   - Creación de un proyecto NodeJS. En este caso, se utilizó el IDE Visual Studio Code, en donde, primero se crea una carpeta, y se la abre en el IDE, luego mediante una terminal del IDE se crea el proyecto NodeJS, ver Figura §11.36.

   - Instalación de la librería sla-graphql-manager (herramienta *SLA GraphQL Manager*) mediante la terminal del IDE, ver Figura §11.37.

```
PS D:\Development\SLA-concepts\SLA-CommerceTools> npm i sla-graphql-manager

added 37 packages, and audited 38 packages in 4s

6 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Figura 11.37: CommerceTools: Instalación de la librería sla-graphql-manager

```
PS D:\Development\SLA-concepts\SLA-CommerceTools> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (sla-commercetools)
version: (1.0.0)
description: Validación y registro del Documento SLA del API CommerceTools
entry point: (index.js)
test command:
git repository:
keywords:
author: Antonio Quiña
license: (ISC)
```

Figura 11.36: CommerceTools: Creación del proyecto NodeJS

- Configuración y uso de la librería. Primero, se debe crear el archivo index.js, en el cual, se instancia el uso de la librería, de acuerdo a las especificaciones indicadas en la sección §8.3.2, ver Figura §11.38.

```
1  import {instanceSLAGraphqlManager} from "sla-graphql-manager"
2  import fs from "fs"
3
4  async function app (){
5      const slaAgreement = fs.readFileSync("sla-commerce.yaml", "utf-8")
6      const ApiProvider = instanceSLAGraphqlManager({
7          introspectionConf:{
8              headers:{
9                  authorization: `Bearer EKHgQYCJMx_Gvoi_Wd5fftfSeCpxFtEE`,
10             },
11             url: "https://api.us-central1.gcp.commercetools.com/sales-test/graphql",
12             levels: 3
13         }
14     })
15     const result = await ApiProvider.registerSLA(slaAgreement)
16     console.log(result);
17     if (result.status === "ERROR"){
18         process.exit(1)
19     }
20 }
21 app();
```

Figura 11.38: CommerceTools: Configuración y uso de la librería sla-graphql-manager

- Resultado de la ejecución del programa (Validación del documento SLA). Se ejecuta el programa NodeJS mediante la terminal del IDE, en donde, la librería realiza la validación del documento SLA, y devuelve como resultado un objeto que tiene tres valores: i) *STATUS*: si pasa la validación devuelve el valor "SUCCESS", y si existe errores devuelve el valor "ERROR". ii) *Errors*:

retorna el detalle de los errores (si los tuviere) del esquema del documento o de los acuerdos modelados en el documento SLA. iii) *Message*: es un mensaje de descripción del resultado, ver Figura §11.39.

```
PS D:\Development\SLA-concepts\SLA-provider> node .\index.js
{
  status: 'SUCCESS',
  errors: [],
  message: 'SLA successfully registered'
}
```

Figura 11.39: CommerceTools: Resultado de ejecución del programa NodeJS

En este sentido, se observa que **la validación** del esquema, como de los acuerdos modelados en las cuotas y tarifas del documento SLA para la API CommerceTools **es exitosa** (SUCCESS).

- *NOTA*: Por temas didácticos en el uso de la herramienta, se inserta errores en el esquema, rutas de la API, y nombres de las métricas del documento SLA, para verificar el control de errores y validación que realiza la herramienta, la Figura §11.40 muestra el resultado de la ejecución.

```
{
  status: 'ERROR',
  errors: [
    'error validating the schema - data.context should NOT have additional properties',
    " data.context should have required property 'type'",
    '"_count-1" metric (xpath function) not found inside quotas > mutations > /createProductType/id-1',
    '"/createProductType/id-1" node not found inside quotas > mutations',
    '"search_characters-1" metric not found inside quotas > queries > //*',
    '"//@limit-1" node not found inside quotas > queries'
  ]
}
```

Figura 11.40: CommerceTools: Resultado de ejecución del programa NodeJS, con errores de validación

2. *Análisis cuantitativo*. Para este análisis, se realiza la evaluación de idoneidad funcional del documento SLA definido en la sección §11.3.2, utilizando la métrica "*Cobertura funcional*" de la ISO/IEC 25023, la cual, plantea la siguiente pregunta: ¿Qué proporción de funciones especificadas han sido implementadas?. Para contestar esta pregunta, la ISO describe la siguiente función de medición [77]:

$$x = 1 - \frac{missingFunctions}{specifiedFunctions} \tag{11.1}$$

En donde, *specifiedFunctions* representa el número de limitaciones especificadas en la documentación del API CommerceTools; y *missingFunctions* son las funciones ausentes, es decir, el número de limitaciones que no pudieron ser modeladas con el estándar *SLA4GraphQL*. La tabla §11.1 muestra la comprobación del modelado de los acuerdos en el documento SLA comparado con la documentación de las limitaciones de la API CommerceTools.

Tabla 11.1: Comprobación del modelado de las limitaciones de la API CommerceTools

Nro.	Limitación	Modelado	Observaciones en el modelado
1	Tamaño del documento JSON	si	
2	Tamaño del contenido del campo	si	
3	Slugs	si	
4	Actualizar acciones por solicitud	si	
5	Consultas	si	
6	Búsqueda y facetas	si	
7	Categorías	si	
8	Variantes del producto	si	
9	Precios	si	
10	Descuentos en productos	si (50 %)	Métrica con condicionales compuestas
11	Descuentos en carrito	no	Métricas con valores de campos; y condicionales compuestas
12	Carritos	si	
13	Listas de compras	si	
14	Zonas	si	
15	Categorías de impuestos	si	
16	Categorías de impuestos	si	
17	Métodos de envío	si	
18	Grupos de clientes	si	
19	Clientes	si	
20	Edición de pedidos	si	
21	Búsqueda de pedidos BETA	no	Limitación BETA; y Documentación aparentemente incompleta
22	Carritos - códigos de descuento	si	
23	Códigos de descuento	si	
24	Tiendas	si	
25	Canales de distribución de productos	si	
26	Selección de productos de la tienda	si	
27	Actualización de Tokens	si (50 %)	Acuerdo sin ruta
28	Objetos personalizados	si	
29	Tipos de productos	si	
30	Extensiones	si	
31	Suscripciones	si	

Continuación de la Tabla §11.1			
Nro.	Limitación	Modelado	Limitaciones en el modelado
32	Registros de registro de auditoría	si (50%)	Acuerdo sin ruta
33	GraphQL - complejidad de consulta	si	

El resultado de la tabla §11.1 evidencia por un lado, que existieron las siguientes dificultades en el modelado de acuerdos: i) Métricas con condicionales compuestas, se refiere a la posibilidad de modelar acuerdos que contengan métricas compuestas por dos o más rutas, o por dos o más métricas. ii) Métricas con valores de campos, se refiere a la posibilidad de modelar acuerdos que se estructuren por una ruta o métrica con valores específicos de los datos de los campos de la consulta. iii) Acuerdo sin ruta, se refiere a que haya la posibilidad de crear acuerdos que no hagan referencia a una ruta específica, y modelen comportamientos generales de la API. En este sentido, por lo antes mencionado los acuerdos 10, 27, y 32 se lograron modelar con cierta interpretación, por lo que se asignó al campo "Modelado" de la tabla §11.1 el valor de "sí (50%)". Por otro lado, de acuerdo a los valores de la fórmula *10.1* se obtuvo que: las limitaciones especificadas son *specifiedFunctions* = 33; y las funciones ausentes son *missingFunctions* = 3.5, es decir:

$$Cobertura\ funcional = 1 - (3.5 / 33) = 89\%$$

Por lo tanto, de acuerdo a la escala de medición de la ISO/IEC 14598-1 [72] (ver figura §9.1), el resultado de la evaluación de la ***Idoneidad Funcional*** **es** ***Satisfactorio***, en la escala que excede los requisitos.

Luego de realizar los análisis cualitativo y cuantitativo del documento SLA de la API CommerceTools, se puede decir que **la especificación del estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) de APIs GraphQL es viable**.

11.4 Evaluación de Flexibilidad

En esta sección, se evalúa el aspecto de flexibilidad del estándar **SLA4GraphQL** para modelar acuerdos de nivel de servicios (SLA) para APIs GraphQL; con el objetivo

de comprobar la capacidad de adaptación de la propuesta al aplicarse en diferentes contextos. La evaluación básicamente consiste en instanciar la propuesta en varias APIs GraphQL; el procedimiento para cada caso de instanciación es similar al realizado en la evaluación de viabilidad (ver sección §11.3).

11.4.1 Definición de los aspectos de flexibilidad

Tomando en cuenta el análisis de selección de casos (ver sección §11.2.1) y la evaluación de viabilidad; el estándar *SLA4GraphQL* se instanciará en las APIs GraphQL de los siguientes casos: GitHub y GitLab. Durante la instanciación, se verifica y reporta si se encuentra limitaciones al momento de modelar los acuerdos; Además, se realiza la evaluación de Idoneidad Funcional de cada caso. Los resultados de la evaluación de flexibilidad se utilizarán para contestar a la pregunta $\mathbf{PI}_{4.2}$ definida en la sección §11.1.

11.4.2 Caso: API GraphQL de GitHub

Sitio oficial de la plataforma de GitHub: `https://github.com/`

Recopilación de datos

Esta sección se reporta de manera más condensada, a comparación de la sección §11.3.2 de recopilación de datos de la evaluación de la viabilidad, la cual se realizó de manera más detallada por temas didácticos. La estructura para instanciar el documento SLA se conforma por tres pasos: i) Recopilación de la documentación de limitaciones de la API GraphQL de GitHub. ii) Configuración de acceso a la API GraphQL de GitHub. iii) Creación del documento SLA para la API GraphQL de GitHub; tal como se describe a continuación:

1. La documentación de la API GraphQL de GitHub reporta básicamente cuatro limitaciones; la misma que está disponible en: `https://docs.github.com/en/graphql/overview/resource-limitations`. A continuación, se resume las limitaciones:

    - Los clientes deben proporcionar un argumento *first* o *last* en cualquier consulta.
    - Los valores de *first* y *last* deben tener valores entre 1 y 100.
    - Las llamadas individuales no pueden solicitar más de 500.000 nodos en total. La Figura §11.41 muestra la manera de calcular los nodos en una llamada.

```
query {
  viewer {
    repositories(first: 50) {
      edges {
        repository:node {
          name
          issues(first: 10) {
            totalCount
            edges {
              node {
                title
                bodyHTML
              }
            }
          }
        }
      }
    }
  }
}
```

Calculation:

```
50          = 50 repositories
+
50 x 10     = 500 repository issues

            = 550 total nodes
```

Figura 11.41: Ejemplo cálculo de nodos en una llamada del API GitHub.

- Límite de tarifa de la API GraphQL es de 5000 puntos por hora. A continuación, en la Figura §11.42 se muestra un ejemplo de consulta y la descripción del cálculo de puntuación [51].

```
query {
  viewer {
    login
    repositories(first: 100) {
      edges {
        node {
          id

          issues(first: 50) {
            edges {
              node {
                id

                labels(first: 60) {
                  edges {
                    node {
                      id
                      name
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Figura 11.42: Ejemplo de una consulta para el cálculo de puntos del API GitHub.

Para el cálculo de puntos de la consulta de la figura §11.42, se requiere 5.101 solicitudes para cumplir, en donde:

- Aunque se devuelve 100 ”repositories”, la API debe conectarse a la cuenta del proveedor una vez para obtener la lista de ”repositories”. Entonces, solicitudes de ”repositories” = 1.

- Aunque se devuelve 50 ”issues”, la API debe conectarse a cada uno de los 100 ”repositories” para obtener la lista de ”issues”. Entonces, solicitudes de ”issues” = 100.
- Aunque se devuelve 60 ”labels”, la API debe conectarse a cada uno de los 5000 posibles ”issues” totales para obtener la lista de ”labels”. Entonces, solicitudes de ”labels” = 5.000.
- Total = 5.101.

Como parte del cálculo el total de solicitudes lo divide para 100 y redondea, es decir, la puntuación final de la consulta es: 51.

2. Luego, se debe configurar el acceso al API GraphQL de GitHub, para obtener un TOKEN de acceso personal, y acceso a la herramienta *GraphiQL Explorer* para verificar las rutas de consultas y mutaciones en el esquema de la API. Para lo cual, se realizó los siguientes pasos:

   - Creación de un TOKEN de acceso personal, de acuerdo con la documentación disponible en: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token.
   - Acceder a la herramienta *GraphiQL Explorer* de GitHub, en el siguiente enlace: `https://graphql.github.com`.
   - Acceder a la información del esquema de la API GraphQL en la sección ”Documentation Explorer”, tal como se muestra en la Figura §11.43.

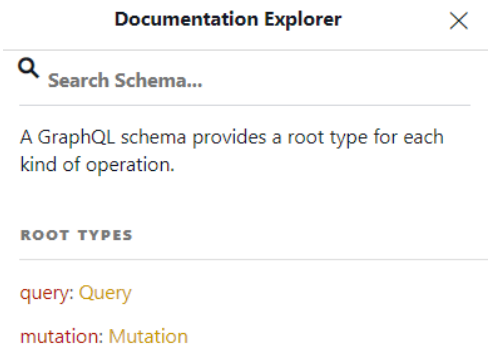


Figura 11.43: Interface del cliente GraphiQL Explorer de la API GitHub.

3. Luego de verificar las limitaciones y tener el acceso a la API GitHub, se modela las limitaciones de la API como acuerdos de nivel de servicio en un documento

SLA (.yaml) basado en el estándar *SLA4GraphQL*. La Figura §11.44 muestra el documento SLA para la API GraphQL de GitHub.

```yaml
context:
  id: github-sla
  type: plans
  version: v1.0
  api: https://api.github.com/graphql
  provider: GitHub
infrastructure:
  supervisor: https://graphql.github.com
  monitor: https://graphql.github.com
metrics:
  points:
    type: "integer"
    description: "Normalized point scale for calculating the frequency limit of API calls"
  nodes:
    type: "integer"
    description: "Nodes returned by a given request"
quotas:
  queries:
    //@first:
      _number:
        min: 1
        max: 100
        over: request
        description: "https://docs.github.com/en/graphql/overview/resource-limitations#node-limit"
      nodes:
        max: 500000
        over: request
        description: "https://docs.github.com/en/graphql/overview/resource-limitations#node-limit"
    //@last:
      _number:
        min: 1
        max: 100
        over: request
        description: "https://docs.github.com/en/graphql/overview/resource-limitations#node-limit"
      nodes:
        max: 500000
        over: request
        description: "https://docs.github.com/en/graphql/overview/resource-limitations#node-limit"
rates:
  queries:
    points:
      max: 5000
      period: hourly
      over: response
      description: https://docs.github.com/en/graphql/overview/resource-limitations#rate-limit
```

Figura 11.44: GitHub: Documento SLA

Análisis de los datos recogidos

Luego de modelar las limitaciones de la API GitHub en el documento SLA (ver figura §11.44) se procede a analizarlo de manera cualitativa y cuantitativamente. Por un lado, se realiza un análisis cualitativo del documento SLA, mediante la validación de su esquema y los acuerdos modelados; para esta validación se usa la herramienta *SLA*

GraphQL Manager. Por otro lado, se analiza cuantitativamente el documento SLA, mediante una evaluación de la idoneidad funcional, basada en las métricas de la ISO/IEC 25023 [77]. A continuación, se detallan los análisis realizados:

1. *Análisis cualitativo*. La validación del documento SLA mediante la herramienta *SLA GraphQL Manager*, se lo realiza con los siguientes pasos:

   - Creación de un proyecto NodeJS. Se comienza creando una carpeta que contendrá el proyecto; luego se abre esa carpeta en el IDE Visual Studio Code; seguidamente, mediante una terminal del IDE se inicializa el proyecto NodeJS, con el comando:
     - *npm init -y*
   - Instalación de la librería sla-graphql-manager (herramienta *SLA GraphQL Manager*), mediante una terminal del IDE, con el comando:
     - *npm i sla-graphql-manager*
   - Configuración y uso de la librería. Se crea el archivo index.js, en donde, se instancia la configuración y uso de la librería, tal como muestra la Figura §11.45.

```
1  import {instanceSLAGraphqlManager} from "sla-graphql-manager"
2  import fs from "fs"
3
4  async function app (){
5      const slaAgreement = fs.readFileSync("sla-github.yaml", "utf-8")
6      const ApiProvider = instanceSLAGraphqlManager({
7          introspectionConf:{
8              headers:{
9                  authorization: `Bearer [redacted]`,
10             },
11             url: "https://api.github.com/graphql",
12             levels: 2
13         }
14     })
15     const result = await ApiProvider.registerSLA(slaAgreement)
16     console.log(result);
17     if (result.status === "ERROR"){
18         process.exit(1)
19     }
20 }
21 app();
```

Figura 11.45: GitHub: Configuración y uso de la librería sla-graphql-manager

   - Resultado de la ejecución del programa (Validación del documento SLA). En la terminal del IDE, se ejecuta el archivo index.js del programa NodeJS con el comando:

- *node index.js*

En este proceso la librería realiza la validación del documento SLA, y devuelve un objeto con el resultado de "SUCCESS", indicando que la **validación del documento SLA del API GitHub es exitosa**, tal como muestra la Figura §11.46.

```
PS D:\Development\SLA-concepts\GitHub-SLA> node .\index.js
{
  status: 'SUCCESS',
  errors: [],
  message: 'SLA successfully registered'
}
```

Figura 11.46: GitHub: Resultado de ejecución del programa NodeJS

2. *Análisis cuantitativo*. para este análisis se realiza la evaluación de idoneidad funcional del documento SLA definido en la sección §11.4.2, utilizando la métrica "*Cobertura funcional*" y la función de medición (*10.1*). La tabla §11.2 muestra la comprobación de las limitaciones descritas en la documentación de la API de GitHub, con los acuerdos modelados en el documento SLA.

Tabla 11.2: Comprobación del modelado de las limitaciones de la API GitHub

Nro.	Limitación	Modelado	Limitaciones en el modelado
1	Argumento *first* o *last* en cualquier consulta	si (50%)	Argumentos requeridos
2	Los valores de *first* y *last* deben estar entre 1 y 100	si	
3	Solicitudes máximo de 500.000 nodos en total	si	
4	Límite de tarifa de 5.000 puntos por hora	si	

El resultado de la Tabla §11.2 evidencia, por un lado, una limitación en el modelado del acuerdo Nro. 1, el cual requiere que los argumentos *first* o *last* sean requeridos en las consultas; de acuerdo a las especificaciones de *SLA4GraphQL* no es posible modelar un acuerdo con campos o argumentos requeridos, no obstante, esta limitación se la pudo modelar de manera solapada en los acuerdos "//@first:" y "//@last:" de la figura §11.44; sin embargo, se asignó al modelado el valor de si (50%). Por otro lado, de acuerdo a la función de medición (*10.1*), se obtiene que el número de limitaciones especificadas *specifiedFunctions* = 4, y el número de limitaciones que no pudieron ser modeladas *missingFunctions* = 0.5; es decir:

Cobertura funcional = 1 - (0.5 / 4) = 88 %

Por lo tanto, de acuerdo a la escala de medición de la ISO/IEC 14598-1 [72] (ver figura §9.1), el resultado de la evaluación de ***Idoneidad Funcional*** **es** ***Satisfactorio***, en la escala que excede los requisitos.

11.4.3 Caso: API GraphQL de GitLab

Sitio oficial de la plataforma de GitLab: `https://about.gitlab.com/`

Recopilación de datos

En esta sección, se recopila las limitaciones documentadas de la API GraphQL de GitLab en un documento SLA, mediante los siguientes pasos: i) Recopilación de las limitaciones desde la documentación de la API GraphQL de GitLab. ii) Configuración de acceso a la API GraphQL de GitLab. iii) Creación del documento SLA para el API GraphQL de GitLab; tal como se describe a continuación:

1. La documentación de la API GraphQL de GitLab reporta básicamente tres limitaciones; la documentación está disponible en: `https://docs.gitlab.com/ee/api/graphql/#limits`. A continuación, se resume las limitaciones:

   - Tamaño máximo de página, es 100 registros (nodos) por página.
   - Complejidad máxima de consulta, es 200 para solicitudes no autenticadas y 250 para solicitudes autenticadas.

     La API GitLab puntúa la complejidad de las consultas, en general, cada campo de una consulta añade el valor de "uno" a la puntuación de complejidad, aunque ésta puede ser mayor o menor para determinados campos. A veces, añadir ciertos argumentos también puede aumentar la complejidad en las consultas.
   - El tiempo de espera máximo de la solicitud, es 30 segundos.

2. Configurar el acceso a la API GraphQL de GitLab para obtener un TOKEN de acceso personal y acceso a la herramienta GraphiQL Explorer para verificar las rutas de consultas y mutaciones en el esquema de la API. Para lo cual, se realizó los siguientes pasos:

- Creación de un TOKEN de acceso personal, se comienza por registrarse como usuario e ingresar a la plataforma de GitLab en: `https://gitlab.com/users/sign_in`. Luego, se crea un token personal de acceso, en el menú de usuario escoger "Preference" → "Access Tokens" → "Create personal access token".
- Acceder a la herramienta GraphiQL Explorer de GitLab, en el siguiente enlace: `https://gitlab.com/-/graphql-explorer`.
- Acceder a la información del esquema de la API GraphQL en la sección "Documentation Explorer".

3. Luego de verificar el esquema y limitaciones de la API se modela los acuerdos de nivel de servicio en un documento SLA (.yaml) basado en el estándar *SLA4GraphQL*. La Figura §11.47 muestra el documento SLA para la API GraphQL de GitLab.

Análisis de los datos recogidos

Luego de modelar las limitaciones de la API GitLab en el documento SLA (ver figura §11.47), se analiza cualitativa y cuantitativamente dicha instanciación. Por un lado, el análisis cualitativo, consiste en analizar la validación del esquema y los acuerdos modelados en el documento SLA, mediante el uso de la herramienta *SLA GraphQL Manager*. Por otro lado, el análisis cuantitativo consiste en analizar la evaluación de la idoneidad funcional del documento SLA, basada en las métricas de la ISO/IEC 25023 [77]. A continuación, se detallan los análisis realizados:

1. *Análisis cualitativo*. La validación del documento SLA mediante la herramienta *SLA GraphQL Manager*, se lo realiza en los siguientes pasos:

   - Creación de un proyecto NodeJS. Se comienza creando una carpeta que contendrá el proyecto; luego se abre esa carpeta en el IDE Visual Studio Code; seguidamente, mediante una terminal del IDE se inicializa el proyecto NodeJS, con el comando:
     - *npm init -y*
   - Instalación de la librería sla-graphql-manager (herramienta *SLA GraphQL Manager*), mediante una terminal del IDE, con el comando:
     - *npm i sla-graphql-manager*

```yaml
context:
  id: gitlab-sla
  type: plans
  version: v1.0
  api: https://gitlab.com/api/graphql
  provider: GitLab
infrastructure:
  supervisor: https://gitlab.com/-/graphql-explorer
  monitor: https://gitlab.com/-/graphql-explorer
metrics:
  authenticated_complexity:
    type: "int64"
    description: "complexity of authenticated requests"
  unauthenticated_complexity:
    type: "int64"
    description: "complexity of unauthenticated requests"
  request_timeout:
    type: "int64"
    description: "Request timeout in seconds"
quotas:
  queries:
    //edges/node:
      _count:
        max: 100
        over: response
        description: "https://docs.gitlab.com/ee/api/graphql/#limits"
    //nodes:
      _count:
        max: 100
        over: response
        description: "https://docs.gitlab.com/ee/api/graphql/#limits"
    //*:
      authenticated_complexity:
        max: 250
        over: response
        description: "https://docs.gitlab.com/ee/api/graphql/#limits"
      unauthenticated_complexity:
        max: 200
        over: response
        description: "https://docs.gitlab.com/ee/api/graphql/#limits"
      request_timeout:
        max: 30
        over: response
        description: "https://docs.gitlab.com/ee/api/graphql/#limits"
```

Figura 11.47: GitLab: Documento SLA

- Configuración y uso de la librería. Se crea el archivo index.js, en donde, se instancia la configuración y uso de la librería, tal como muestra la Figura §11.48.
- Resultado de la ejecución del programa (Validación del documento SLA). En la terminal del IDE, se ejecuta el archivo index.js del programa NodeJS con el comando:
  - *node index.js*

  En este proceso la librería realiza la validación del documento SLA, y de-

```
import {instanceSLAGraphqlManager} from "sla-graphql-manager"
import fs from "fs"

async function app (){
    const slaAgreement = fs.readFileSync("sla-gitlab.yaml", "utf-8")
    const ApiProvider = instanceSLAGraphqlManager({
        introspectionConf:{
            headers:{
                authorization: `Bearer glpat-Soq4134yzcwWqGB9eBcA`,
            },
            url: "https://gitlab.com/api/graphql",
            levels: 2
        }
    })
    const result = await ApiProvider.registerSLA(slaAgreement)
    console.log(result);
    if (result.status === "ERROR"){
        process.exit(1)
    }
}
app();
```

Figura 11.48: GitLab: Configuración y uso de la librería sla-graphql-manager

vuelve un objeto con el resultado de "SUCCESS", indicando que la **validación del documento SLA del API GitHub es exitosa**, tal como muestra la Figura §11.49.

```
PS D:\Development\SLA-concepts\GitHub-SLA> node .\index.js
{
  status: 'SUCCESS',
  errors: [],
  message: 'SLA successfully registered'
}
```

Figura 11.49: GitLab: Resultado de ejecución del programa NodeJS

2. *Análisis cuantitativo*. para este análisis se realiza la evaluación de idoneidad funcional del documento SLA definido en la sección §11.4.3, utilizando la métrica "*Cobertura funcional*" y la función de medición (*10.1*). Para lo cual, en la tabla §11.3 se realiza la comprobación de las limitaciones descritas en la documentación de la API de GitLab, con los acuerdos modelados en el documento SLA.

Tabla 11.3: Comprobación del modelado de las limitaciones de la API GitLab

Nro.	Limitación	Modelado	Limitaciones en el modelado
1	Tamaño máximo de página	si	
2	Complejidad máxima de consulta	si	

Continuación de la Tabla §11.3			
Nro.	Limitación	Modelado	Limitaciones en el modelado
3	Tiempo de espera de la solicitud	si	

En donde, de acuerdo a los resultados de la tabla §11.3 y la función de medición (*10.1*), se obtiene que el número de limitaciones especificadas *specifiedFunctions* = 3, y el número de limitaciones que no pudieron ser modeladas *missingFunctions* = 0; por lo tanto:

$$Cobertura\ funcional = 1 - (0 / 3) = 100\,\%$$

Tomando en cuenta la escala de medición de la ISO/IEC 14598-1 [72] (ver figura §9.1), el resultado de la evaluación de ***Idoneidad Funcional* es *Satisfactorio***, en la escala que excede los requisitos.

11.5 Evaluación de Idoniedad Funcional

El objetivo de la evaluación de *idoneidad funcional* es verificar el grado de funcionalidad que tiene el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) para APIs GraphQL. En este sentido, se ha aplicado el estándar **SLA4GraphQL** en tres casos de APIs GraphQL públicas, y se ha evaluado su idoneidad funcional utilizando la métrica de "Cobertura funcional" de la ISO/IEC 25023 (ver secciones §11.3, §11.4). La Tabla §11.4 muestra el resumen de los resultados del análisis cuantitativo de la idoneidad funcional de los casos: CommerceTools, GitHub, y GitLab, realizados en las evaluaciones de viabilidad y flexibilidad del estándar *SLA4GraphQL*.

Tabla 11.4: Evalución de idoneidad funcional del estándar *SLA4GraphQL*

Caso	Idoneidad Funcional
CommerceTools	89 %
GitHub	88 %
GitLab	100 %
Promedio	**92.33 %**

De acuerdo a la escala de medición de la ISO/IEC 14598-1 [72] (ver figura §9.1), el resultado de la evaluación de ***Idoneidad Funcional* del estándar *SLA4GraphQL* es *Satisfactorio***, en la escala que excede los requisitos.

11.6 Conclusiones y trabajo futuro

11.6.1 Conclusiones

En este capítulo se validó el estándar *SLA4GraphQL* propuesto en el capítulo §8 con el objetivo de contestar la pregunta de investigación **PI**$_4$: ¿En qué medida es posible modelar los acuerdos de APIs GraphQL?. En este sentido, es necesario aclarar que al inicio de este capítulo (ver sección §11.1) se relaciona la **PI**$_4$ con *SLA4GraphQL* (estándar para modelar y operativizar acuerdos de nivel de servicio (SLA) para APIs GraphQL), y se plateó tres preguntas de investigación derivadas que concretamente se contestan a continuación:

- **PI**$_{4.1}$: Viabilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar los acuerdos de nivel de servicio (SLA) de un caso en concreto?.

  En la sección §11.3, se realizó la evaluación de viabilidad mediante la instanciación del estándar *SLA4GraphQL* en un documento SLA que modeló las limitaciones del API CommerceTools. En donde, luego de realizar los análisis cualitativo y cuantitativo del documento SLA, se pudo comprobar que la instanciación de *SLA4GraphQL* fue exitosa, por lo tanto, **el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) de APIs GraphQL es viable**.

- **PI**$_{4.2}$: Flexibilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) en distintos casos concretos?.

  En la sección §11.4, se realizó la evaluación de flexibilidad mediante la instanciación del estándar *SLA4GraphQL* para las APIs GitHub y GitLab, en donde, se modeló en documentos SLA los acuerdos de nivel de servicio en base a las limitaciones documentadas de las APIs. Luego de realizar los análisis cualitativo y cuantitativo de los documentos SLA, se comprobó que la instanciación de *SLA4GraphQL* fue exitosa en varios casos en concreto, por lo tanto, **el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) de APIs GraphQL es flexible**.

- **PI**$_{4.3}$: Idoneidad Funcional. ¿En qué medida el estándar *SLA4GraphQL* proporciona funciones que cumplan con necesidades establecidas en condiciones específicas?.

  En la sección §11.5, se realizó la evaluación de idoneidad funcional del estándar *SLA4GraphQL*. Cabe indicar que esta evaluación fue parte del análisis cuantitati-

vo en las evaluaciones de viabilidad y flexibilidad, en los casos de las APIs CommerceTools, GitHub, y GitLab. En donde, el resultado global de la evaluación de idoneidad funcional indica que, **el estándar *SLA4GraphQL* es 92.33 % idóneo para modelar acuerdos de nivel de servicio (SLA) para APIs GraphQL**.

Por tal razón, se concluye que el estándar *SLA4GraphQL* es viable, flexible y tiene una idoneidad funcional del 92.33 % para modelar acuerdos de nivel de servicio (SLA) para APIs GraphQL.

11.6.2 Trabajo futuro

Como resultado del análisis cualitativo realizado en las evaluaciones de viabilidad y flexibilidad se pudo evidenciar las siguientes limitaciones del estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio, las cuales se plantean como trabajo futuro: i) Métricas con condicionales compuestas, se refiere a la posibilidad de modelar acuerdos compuestos por dos o más métricas, o por dos o más rutas. ii) Métricas con valores de campos, se refiere a la posibilidad de modelar acuerdos que se estructuren por una ruta o métrica con valores específicos de los datos de los campos de la consulta. iii) Acuerdo sin ruta, se refiere a la posibilidad de crear acuerdos que no hagan referencia a una ruta específica, y se pueda modelar comportamientos generales de la API. iv) Argumentos requeridos, se refiere que en los acuerdos se pueda modelar que uno o varios argumentos de las consultas a la API GraphQL, puedan ser descritas como requeridos. Además, se plantea realizar estudios que validen la operativización de los documentos SLA modelados para las APIs GraphQL.

Finalmente, se plantea como trabajo futuro, la posibilidad de establecer un estándar para modelar acuerdos de nivel de servicio tanto para APIs REST, como GraphQL, y puedan coexistir.

11.7 Resumen

En este capítulo se validó el estándar *SLA4GraphQL* propuesto en el capítulo §8; la validación consistió en evaluar la viabilidad, flexibilidad e idoneidad funcional del estándar. Con lo cual, se contesta la pregunta de investigación $\mathbf{PI}_4$: ¿En qué medida es posible modelar los acuerdos de APIs GraphQL?, mediante la contestación de sus preguntas derivadas: $\mathbf{PI}_{4.1}$: Viabilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL*

para modelar los acuerdos de nivel de servicio (SLA) de un caso en concreto?. Con la evaluación de viabilidad, se comprobó la viabilidad del estándar mediante la instanciación exitosa de la API CommerceTools. **PI**$_{4.2}$: Flexibilidad. ¿Ha sido posible instanciar el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) en distintos casos concretos?. Con la evaluación de flexibilidad, se pudo comprobar que la instanciación del estándar *SLA4GraphQL* fue exitosa en los casos de las APIs GitHub y GitLab. **PI**$_{4.3}$: Idoneidad Funcional. ¿En qué medida el estándar *SLA4GraphQL* proporciona funciones que cumplan con necesidades establecidas en condiciones específicas?. Con la evaluación de la idoneidad funcional del estándar aplicado en las APIs de CommerceTools, GitHub, y GitLab, se comprobó que *SLA4GraphQL* es 92.33 % idóneo para modelar acuerdos de nivel de servicio (SLA) para APIs GraphQL.

Finalmente, se concluye que el estándar *SLA4GraphQL* es viable, flexible e idóneo funcionalmente para modelar acuerdos de nivel de servicio (SLA) para APIs GraphQL.

Parte V

Conclusiones

Conclusiones y trabajos futuros

El espíritu humano debe prevalecer sobre la tecnología

Albert Einstein (1879 – 1955),
Físico alemán

Este capítulo concluye la memoria, presentando una visión general del trabajo realizado mediante el análisis de los objetivos y las conclusiones.

En la sección §12.1, se analiza el cumplimiento de los objetivos propuestos. En la sección §12.2, se resumen las publicaciones derivadas de este trabajo. Por ultimo, en la sección §12.3, se apuntan las líneas de trabajo futuro.

12.1 Consecución de objetivos

El principal objetivo de esta memoria es **mejorar la gobernanza de arquitecturas híbridas REST y GraphQL mediante contratos de niveles de servicio** (ver sección §1.3). Por lo cual, en esta sección se analiza el cumplimiento de cada uno de los objetivos parciales definidos para conformar el objetivo principal.

1. *Estudiar el estado actual de GraphQL para evaluar su madurez y tendencias.*

   En el capítulo §3 se ha establecido y validado un marco conceptual llamado paradigma GraphQL; y en el capítulo §5 se han analizado los estudios primarios acerca del paradigma GraphQL publicados desde 2015 hasta el 30 de junio del 2021. Estos estudios han permitido:

   a) Validar la implementación del paradigma GraphQL en un caso en concreto.

   b) Identificar los principales autores e instituciones que investigan sobre el paradigma GraphQL.

   c) Detectar los lugares donde se publicaron los trabajos de investigación acerca del paradigma GraphQL.

   d) Establecer la madurez de las publicaciones sobre el paradigma GraphQL, mediante la identificación de los tipos y métodos de investigación utilizados en los estudios.

   e) Verificar la tendencia de publicación acerca del paradigma GraphQL.

   f) Determinar en qué dominios, contextos y áreas del conocimiento de la ingeniería del software se realizaron los estudios acerca del paradigma GraphQL.

   g) Identificar las clasificaciones técnicas específicas del uso y aplicación del paradigma GraphQL.

2. *Verificar los efectos de los paradigmas REST y GraphQL en el proceso de desarrollo con respecto a la calidad del software*. Las condiciones y circunstancias del uso de los paradigmas, se caracterizaron con las subcaracterísticas de calidad: *Facilidad de Aprendizaje*, y *Eficacia* del producto de software; las mismas qué fueron complementadas con el estudio de la *curva de aprendizaje* de los paradigmas.

   En los capítulos §6 y §9, se ha llevado a cabo un experimento con personas que comprueba el efecto que tienen los paradigmas REST y GraphQL en el desarrollo de APIs, realizado en condiciones y circunstancias controladas en un contexto académico. Este estudio ha permitido:

a) Evaluar la *Facilidad de Aprendizaje* de los paradigmas de desarrollo de software; medido por la proporción de funciones que las personas pueden aplicar en el desarrollo de APIs, en función a las funciones explicadas. En donde, se comprobó que GraphQL presenta una ligera mejora en la *Facilidad de Aprendizaje* con respecto al paradigma REST. Sin embargo, de acuerdo a la ISO/IEC 14598 y las pruebas estadísticas de *Wald*, estas diferencias no son significativas

b) Conocer la *Eficacia* de los paradigmas en el rendimiento de las personas, medido por la proporción de tareas de desarrollo de APIs que se completan correctamente sin ayuda. En donde, GraphQL presenta una ligera mejora en la *Eficacia* con respecto al paradigma REST. Sin embargo, de acuerdo a la ISO/IEC 14598 y las pruebas estadísticas de *Wald*, estas diferencias no son significativas

c) Entender la *Curva de aprendizaje* de los paradigmas de desarrollo; medida por el tiempo de solución de los requisitos de las tareas de desarrollo de APIs, para entender la generación de aprendizaje individual que obtienen las personas cuando repiten un proceso y adquieren habilidad o eficiencia en razón de su propia experiencia. En donde, se pudo notar que el tiempo medio de la solución de los requisitos realizados con GraphQL es menor que los requisitos realizados con REST. Además, el cálculo de la curva de aprendizaje denota que los participantes aprenden más rápido el paradigma GraphQL frente al paradigma REST.

3. *Comprobar los efectos de las arquitecturas híbridas REST/GraphQL.* Los escenarios de uso de las arquitecturas se caracterizaron por un lado, por la eficiencia del rendimiento del software relativo al tiempo medio de respuesta para completar trabajos asíncronos de diferentes complejidades. Por otro lado, por el tamaño de la respuesta de consultas realizadas a las APIs desarrolladas en diferentes arquitecturas.

   En los capítulos §7 y §10 se diseña y valida un experimento computacional que compara la eficiencia de dos APIs, una desarrollada en una arquitectura REST (API REST de Mendeley), y otra en una arquitectura híbrida REST/GraphQL, esta última consiste en el desarrollo de una API GraphQL que envuelve a la API REST de Mendeley. Los principales efectos del desarrollo de la arquitectura híbrida han sido:

   a) El *tiempo medio de respuesta* de las consultas realizadas a la API GraphQL (ar-

quitectura híbrida REST/GraphQL), es menor al tiempo medio de respuesta de las consultas realizadas directamente a la API REST.

b) El *tamaño medio de respuesta* de las consultas realizadas a la API GraphQL (arquitectura híbrida REST/GraphQL), es menor al tamaño medio de respuesta de las consultas realizadas directamente a la API REST.

c) *Los escenarios más convenientes* para usar las arquitecturas híbridas REST/-GraphQL son cuando las *consultas son más complejas*, mientras más complejidad tengan las consultas, más eficientes se vuelven las arquitecturas híbridas.

d) Otros *escenarios convenientes* para usar las arquitecturas híbridas REST/GraphQL son cuando se realizan *consultas con gran cantidad de datos*, mientras más datos se consulte las arquitecturas híbridas muestran más eficiencia a comparación de las arquitecturas REST.

4. *Estandarizar el modelado y operación de acuerdos de nivel de servicio (SLA) para APIs GraphQL.*

   En el capítulo §8 se propone el estándar *SLA4GraphQL* para modelar y operativizar acuerdos de nivel de servicio (SLA) para APIs GraphQL, en donde, se propone:

   a) Una *especificación para modelar* acuerdos de nivel de servicios en documentos SLA para APIs GraphQL, tomando en cuenta las particularidades de uso y consumo de este tipo de APIs.

   b) Se establece un *ciclo de vida* para el desarrollo de APIs GraphQL basadas en SLA, y la descripción de los principales procesos para operativizar los documentos SLA.

   c) Se propone la herramienta *SLA GraphQL Manager* que realiza por un lado la validación de la especificación de modelado de los documentos SLA; y por otro lado, el registro de los documentos SLA en un repositorio de la nube como inicio de su operación.

   Luego, en el capítulo §11 se valida el estándar propuesto mediante las evaluaciones de viabilidad, flexibilidad e idoneidad funcional, en donde, muestra que:

   a) La *evaluación de viabilidad*, se realizó mediante la instanciación del estándar *SLA4GraphQL* para modelar el documento SLA de las limitaciones documentadas de la API CommerceTools. En donde, luego de realizar los análisis cualitativo y cuantitativo del documento SLA, se pudo comprobar que la

instanciación de *SLA4GraphQL* fue exitosa, por lo tanto, el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) de APIs GraphQL es viable.

b) La *evaluación de flexibilidad*, se realizó mediante la instanciación del estándar *SLA4GraphQL* para las APIs GitHub y GitLab, en donde, se modeló en documentos SLA los acuerdos de nivel de servicio en base a las limitaciones documentadas de las APIs. Luego de realizar los análisis cualitativo y cuantitativo de los documentos SLA, se comprobó que la instanciación de *SLA4GraphQL* fue exitosa en varios casos en concreto, por lo tanto, el estándar *SLA4GraphQL* para modelar acuerdos de nivel de servicio (SLA) de APIs GraphQL es flexible.

c) La *evaluación de idoneidad funcional* del estándar *SLA4GraphQL*, fue parte del análisis cuantitativo en las evaluaciones de viabilidad y flexibilidad, en los casos de las APIs CommerceTools, GitHub, y GitLab. En donde, el resultado global de la evaluación de idoneidad funcional indica que, el estándar *SLA4GraphQL* es 92.33 % idóneo para modelar acuerdos de nivel de servicio (SLA) para APIs GraphQL.

12.2 PUBLICACIONES

Los resultados parciales obtenidos durante la realización del presente trabajo de investigación han dado lugar a las publicaciones que se enumeran a continuación.

12.2.1 Revistas internacionales

- A. Quiña-Mera, P. Fernández, J. García, and A. Ruiz–Cortés.
  GraphQL: A Systematic Mapping Study.
  ACM Computing Surveys. ISNN:0360-0300, 2022.
  Índice de impacto: JCR: 10.282 (Q1); SJR: 5.09 (2021, Q1)
  Estado: [Condicionalmente aceptado], tercera revisión en proceso.

  En este trabajo se presenta un estudio de mapeo sistemático de la literatura que proporciona una visión general y específica del estado de GraphQL, para identificar tendencias y lagunas en el estudio de este paradigma. Además, se propone un marco conceptual que ilustra conceptualmente la especificación formal de

GraphQL.

- El estudio de los *efectos de los paradigmas REST y GraphQL en el desarrollo del software* aquí presentado que ha permitido verificar los efectos de los paradigmas en el proceso de desarrollo con respecto a la calidad del software. Será próximamente enviado a una revista científica.
- *Estandarizar el modelado y operación de acuerdos de nivel de servicio (SLA) para APIs GraphQL* aquí presentado que propone un estándar para modelar y operativizar SLA para APIs GraphQL. Será próximamente enviado a una revista científica.

12.2.2 Congresos internacionales

- [129] A. Quiña-Mera, P. Fernández, J. García, E. Bastidas and A. Ruiz–Cortés.
  Quality in Use Evaluation of a GraphQL Implementation.
  Multidisciplinary International Congress on Science and Technology (CIT 2021).
  Online 14-18 June 2021
  Proceedings in Book series: Lecture Notes in Networks and Systems.
  ISNN:2367-3370, 2021
  DOI: 10.1007/978-3-030-96043-8_2
  Disponible en: https://link.springer.com/chapter/10.1007/978-3-030-96043-8_2
  Índice de impacto: SJR: 0,17 (Q4)
  Estado: [Aceptado]

  En este trabajo se demuestra el funcionamiento y aplicación de implementaciones complejas de GraphQL, mediante la evaluación de la calidad en uso (basada en la serie de normas ISO/IEC 25000) de una implementación de GraphQL que automatiza el proceso de estudios de mapeo sistemático (SMS).

- A. Quiña-Mera, J. García, P. Fernández, K. Rodríguez, and A. Ruiz–Cortés.
  Effects of hybrid REST and GraphQL architectures on software quality.
  The 37th IEEE/ACM International Conference on Automated Software Engineering (ASE 2022).
  Ann Arbor, Michigan, United StatesMon, between 10 and 14 October 2022
  GII-GRIN-SCIE (GGS) Conference Rating: CORE:A++, LiveSHINE:A, MA:A
  Estado: [Enviado], fecha de notificación Fri 29 Jul 2022.

  En este trabajo se expone un experimento computacional que compara la eficiencia de dos APIs, una desarrollada en una arquitectura REST (API REST de Men-

deley), y otra en una arquitectura híbrida REST/GraphQL. Será próximamente enviado a una conferencia internacional.

- A. Quiña-Mera, J. García, P. Fernández, P. Vega, and A. Ruiz–Cortés.
  GraphQL or REST for mobile applications?.
  The International Conference on Advanced Research in Technologies, Information, Innovation and Sustainability (ARTIIS 2022).
  Santiago de Compostela, Spain, between 12th and 14th of September 2022.
  Proceedings in Book series: Communications in Computer and Information Science.
  Impact Factor: SJR: 0.21 (2021, Q4).
  Estado: [Enviado], fecha de notificación: July 5, 2022.

  En este trabajo se expone un experimento computacional que compara la eficiencia del rendimiento de tres APIs: una API GraphQL y dos API REST (una que expone consultas complejas en varios puntos finales, la otra que expone consultas complejas en un único punto final). Los resultados muestran que la calidad del software de la API desarrollada con arquitectura GraphQL es superior a la desarrollada con arquitectura REST.

12.2.3 Participación en proyectos de investigación

- Director del proyecto de investigación interno de la Facultad de Ingeniería en Ciencias Aplicadas, Universidad Técnica del Norte. Gobierno de arquitecturas híbridas REST y GraphQL mediante contratos (Segunda fase). Duración 12 meses. Integrantes: A. Quiña-Mera, M. Rea, P. Fernández, J. García, and A. Ruiz–Cortés. Fecha inicio: 21 diciembre 2018, Fecha fin: 21-diciembre-2019. Aprobado con resolución HCD Nro. UTN-FICA-2019-0061.

- Director del proyecto de investigación interno de la Facultad de Ingeniería en Ciencias Aplicadas, Universidad Técnica del Norte. Gobierno de arquitecturas híbridas REST y GraphQL mediante contratos (Segunda fase). Duración 12 meses. Integrantes: A. Quiña-Mera, M. Rea, C. Guevara, P. Fernández, J. García, and A. Ruiz–Cortés. Fecha inicio: 01 febrero 2020, Fecha fin: 31-enero-2021. Aprobado con resolución HCD Nro. UTN-SJFICA-2020-0314-M.

- Director del proyecto de investigación interno de la Facultad de Ingeniería en Ciencias Aplicadas, Universidad Técnica del Norte. Gobierno de arquitecturas

híbridas REST y GraphQL mediante contratos (Tercera fase). Duración 12 meses. Integrantes: A. Quiña-Mera, M. Rea, C. Guevara, P. Fernández, J. García, and A. Ruiz–Cortés. Fecha inicio: 01 febrero 2021, Fecha fin: 31-julio-2022 (prórroga de 4 meses). Aprobado con resolución HCD Nro. UTN-SJFICA-2021-0144-M.

12.3 TRABAJO FUTURO

Una vez finalizado el presente estudio de investigación, se ha evidenciado varias oportunidades de estudio, y nuevas ideas las cuales se proponen como trabajo futuro a mediano y largo plazo. Entre las cuáles se destacan las siguientes:

- *Componentes GraphQL*, se propone validar la implementación de la operación *"Subscription"*, debido a que este componente quedó fuera del estudio de la validación de la implementación de GraphQL; Además, se sugiere realizar más estudios que muestren evidencia empírica acerca del comportamiento de las implementaciones de GraphQL.

- *Efectos de los paradigmas de desarrollo*, se incentiva realizar la replicación del experimento para comparar los efectos de los paradigmas REST y GraphQL en el proceso de desarrollo con respecto a la calidad del software, en otras poblaciones de participantes de pregrado y posgrado en el contexto académico, como también replicar el experimento en el contexto industrial. También, se promueve la experimentación para comparar otras características de la calidad interna, externa y de uso del producto de software entre los paradigmas REST y GraphQL. Finalmente, se promueve el estudio comparativo en diferentes dominios de la arquitectura de software como por ejemplo en el lado de proveedor y consumidor del servicio de APIs, como también la eficiencia a nivel de acceso de datos con los paradigmas REST y GraphQL.

- *Efectos de los paradigmas híbridos REST/GraphQL*, se promueve complementar el experimento del efecto de paradigmas híbridos con nuevos casos de uso que manipulen operaciones como mutaciones. Además, se incentiva a la comunidad científica a realizar estudios similares con arquitecturas híbridas con otras APIs públicas de preferencia que estén aplicadas en la práctica industrial de la ingeniería de software. También, se promueve la experimentación para comparar arquitecturas híbridas REST y GraphQL con otras características de la calidad

del producto de software. Además, se promueve la realización de estudios de arquitecturas híbridas de GraphQL con otras tecnologías como SOAP, SPARQL o Falcor.

- *Modelado y operación de acuerdos de nivel de servicio (SLA) para APIs GraphQL*, luego de la validación del estándar *SLA4GraphQL* se evidenció las siguientes oportunidades de mejora para modelar documentos SLA: i) Métricas con condicionales compuestas, es la posibilidad de estructurar métricas con dos o más rutas y/o métricas. ii) Métricas con valores de campos, es la posibilidad de modelar acuerdos que se estructuren por una ruta o métrica con valores específicos de los datos de los campos de la consulta. iii) Acuerdo sin ruta, es la posibilidad de crear acuerdos que no hagan referencia a una ruta específica, y se pueda modelar comportamientos generales de la API. iv) Argumentos requeridos, se refiere que en los acuerdos se pueda modelar uno o varios argumentos como requeridos en las consultas a la API GraphQL. Además, se plantea realizar estudios que validen la operativización de los documentos SLA modelados para las APIs GraphQL. Finalmente, se plantea la posibilidad de establecer un estándar para modelar acuerdos de nivel de servicio tanto para APIs REST, como GraphQL, y puedan coexistir.

Parte VI

Anexos

Estudios primarios del Mapeo Sistemático de GraphQL

En este anexo se muestra el conjunto de estudios primarios que fueron mapeados en el *Estudio de Mapeo Sistemático* (SMS, por sus siglas en inglés de Systematic Mapping Study) acerca GraphQL realizado en el capítulo §5.

Tabla A.1: Estudios primarios del SMS

Comienzo de tabla		
No.	Título	Referencia
E01	An empirical analysis of GraphQL API schemas in open code repositories and package registries	[86]
E02	An initial analysis of facebook's GraphQL language	[61]
E03	Advanced Data Fetching with GraphQL: Case Bakery Service	[160]
E04	API Design in Distributed Systems: A Comparison between GraphQL and REST	[35]
E05	Bioactivity-explorer: a web application for interactive visualization and exploration of bioactivity data	[104]
E06	Bridges between GraphQL and RDF	[159]
E07	Context-Aware Access to Heterogeneous Resources Through On-the-Fly Mashups	[27]
E08	Defining Schemas for Property Graphs by Using the GraphQL Schema Definition Language	[60]
E09	Design and Implementation of Real-Time Management System Architecture based on GraphQL	[57]
E10	Detecting Cycles in GraphQL Schemas	[106]
E11	Efficient Architecture Design for Software as a Service in Cloud Environments	[145]
E12	Experiences on Migrating RESTful Web Services to GraphQL	[174]
E13	Generating GraphQL-Wrappers for REST(-like) APIs	[183]
E14	GraphQL and DC-WSN-Based Cloud of Things	[132]

Continuación de la Tabla §A.1		
No.	Título	Referencia
E15	GraphQL for archival metadata: An overview of the EHRI GraphQL API	[13]
E16	A model-driven framework for data-driven applications in serverless cloud computing	[140]
E17	Implementing GraphQL as a query language for deductive databases in SWI-Prolog using DCGs, quasi quotations, and dicts	[113]
E18	Improving the OEEU's Data-driven Technological Ecosystem's Interoperability with GraphQL	[171]
E19	On the Role of Context in the Design of Mobile Mashups	[16]
E20	QL4MDR: a GraphQL query language for ISO 11179-based metadata repositories	[168]
E21	Towards a UML and IFML Mapping to GraphQL	[136]
E22	Using GraphQL for Content Delivery in Kentico Cloud	[17]
E23	Lion: listen online. Using GraphQL as a mediator for data integration and ingestion	[31]
E24	Migrating to GraphQL: A Practical Assessment	[10]
E25	morph-GraphQL: GraphQL Servers Generation from R2RML Mappings (Sese).	[126]
E26	Querying heterogeneous linked building data with context-expanded GraphQL queries	[179]
E27	Rendering real-time dashboards using a GraphQL- based UI Architecture	[34]
E28	Semantics and Complexity of GraphQL	[62]
E29	Performance analysis of Web Services: Comparison between RESTful & GraphQL web services	[65]
E30	An Empirical Study of GraphQL Schemas	[182]
E31	A GraphQL approach to Healthcare Information Exchange with HL7 FHIR	[110]
E32	A Link Generator for Increasing the Utility of OpenAPI-to-GraphQL Translations	[97]
E33	A mechanized formalization of GraphQL	[29]
E34	Analysis of GraphQL performance: a case study	[41]
E35	Block Affordances for GraphQL in MIT App Inventor	[18]
E36	Build a GraphQL application with Node. js and React	[166]

Continuación de la Tabla §A.1		
No.	Título	Referencia
E37	Comparative Analysis Between Standards Oriented to Web Services: SOAP, REST and GRAPHQL	[141]
E38	Design of Hybrid Application Based on GraphQL for Efficient Query for PHR	[79]
E39	Deviation testing: A test case generation technique for GraphQL APIs	[109]
E40	Empirical study on the usage of graph query languages in open source Java projects	[146]
E41	Evaluating execution strategies of GraphQL queries	[137]
E42	Exploiting Declarative Mapping Rules for Generating GraphQL Servers with Morph-GraphQL	[22]
E43	Exploring the quality attribute and performance implications of using GraphQL in a data-fetching API	[181]
E44	A Performance Comparison of Auto-Generated GraphQL Server Implementations	[100]
E45	GraphQL as modern access to jBPM process engine	[59]
E46	GraphQL Federation: A Model-Based Approach	[156]
E47	A principled approach to GraphQL query cost analysis	[19]
E48	GraphQL Schema Generation for Data-Intensive Web APIs	[40]
E49	GraphQL: The API Design Revolution	[135]
E50	How fast GraphQL is compared to REST APIs	[114]
E51	Implementation and evaluation of a graphql-based web application for project follow up	[151]
E52	Implementing graphql in existing REST api	[99]
E53	Microservice architecture patterns with GraphQL	[165]
E54	News API implementation with serverless GraphQL	[4]
E55	Performance Analysis of GraphQL and RESTful in SIM LP2M of the Hasanuddin University	[63]
E56	REST vs GraphQL: A Controlled Experiment	[11]
E57	Social Media Intelligence and Learning Environment: an Open Source Framework for Social Media Data Collection, Analysis and Curation	[177]
E58	Sustainable IoT sensing applications development through graphQL-based abstraction layer	[85]
E59	Transformation of REST API to GraphQL for OpenTOSCA	[49]

Continuación de la Tabla §A.1		
No.	Título	Referencia
E60	An ontological metamodel for cyber-physical system safety, security, and resilience coengineering	[5]
E61	An Overview of GraphQL: Core Features and Architecture	[154]
E62	Automatic bootstrapping of GraphQL endpoints for RDF triple stores	[54]
E63	Automatic Property-based Testing of GraphQL APIs	[84]
E64	Building a modern data archive with React, GraphQL, and friends	[66]
E65	Combination of CityJSON with PostgreSQL, MongoDB and GraphQL	[155]
E66	Comparative Analysis of Rest and GraphQL Technology on Nodejs-Based Api Development	[158]
E67	Comparative analysis of web application performance in case of using Rest versus GraphQL	[173]
E68	Development of a centralized system for data storage and processing on operation modes and reliability indicators of power equipment	[157]
E69	FGPE Gamification Service: A GraphQL Service to Gamify Online Education	[117]
E70	GraphQL for building microservices	[127]
E71	GraphQL Interface for OPC UA	[68]
E72	GraphQL-based generic and domain specific query interfaces for the JValue ODS	[176]
E73	Guaranteeing Type Consistency in Collective Adaptive Systems	[142]
E74	Identification and Evaluation of a Process for Transitioning from REST APIs to GraphQL APIs in the Context of Microservices Architecture	[55]
E75	Migrating from REST to GraphQL having long-term supported clients	[95]
E76	MVP Architecture Model with Single Endpoint Access for Displaying COVID 19 Patients Information Dynamically	[149]
E77	Performance comparison: Between GraphQL, REST & SOAP	[38]
E78	Performance Measurement of GraphQL API in Home ESS Data Server	[102]

Continuación de la Tabla §A.1		
No.	Título	Referencia
E79	RCSB Protein Data Bank: Architectural Advances Towards Integrated Searching and Efficient Access to Macromolecular Structure Data from the PDB Archive	[138]
E80	REST API vs GraphQL: A literature and experimental study	[1]
E81	Student Behavior Report Management System on Somsri.IO	[20]
E82	The Web service development with React, GraphQL and Apollo	[150]
E83	Tokocabai marketplace application based on web using extreme programming method	[130]
E84	Zincbindpredict—prediction of zinc binding sites in proteins	[70]
Fin de tabla		

Efectos de las arquitecturas REST y GraphQL: Paquete de laboratorio

Este anexo muestra el paquete de laboratorio utilizado en el experimento de *"Facilidad de aprendizaje entre GraphQL y REST"* diseñado en el capítulo §6, y validado en el capítulo §9. En este apartado, se despliega la estructura y el conjunto de artefactos que conforman el paquete de laboratorio, en donde, en lo posible se imprime el artefacto o parte de este, y en otros casos se describe su propósito y se indica su ubicación en la estructura de carpetas del paquete de laboratorio publicado en un repositorio digital de Zenodo[1]: [128].

La estructura del anexo se conforma por la sección §B.1 que muestra los artefactos de las tareas de formación. La sección §B.2 muestra los artefactos de las tareas experimentales; y por último la sección §B.3 muestra los artefactos de las tareas de análisis del experimento.

B.1 Entrenamiento

En esta sección, se detalla los artefactos usados en las tareas de formación del experimento descritas en las tablas §6.2 y §6.3.

B.1.1 Encuesta demográfica/diagnóstica

En esta fase del experimento se diseñó una encuesta para levantar información demográfica de los participantes, y para diagnosticar el conocimiento y percepción de la facilidad de uso, consumo y aprendizaje de los paradigmas REST y GraphQL de desarrollo de APIs. La Tabla §B.1 muestra las preguntas de la encuesta; conformada por las 12 primeras preguntas acerca de información demográfica, y el resto de información acerca de los paradigmas de desarrollo de APIs, en donde, concretamente las preguntas 18, 19, 22, 23, 24, y 25 se refieren a la facilidad de uso, consumo, y aprendizaje de los paradigmas.

[1] https://zenodo.org/

Tabla B.1: Encuesta demográfica/diagnóstica

Nro.	Pregunta
1	Nombre completo
2	Correo electrónico personal
3	País
4	Ciudad
5	Dirección
6	Teléfono
7	Edad
8	¿Cuál es su definición étnica?
9	¿Cuál es su género?
10	¿Cuál es su estado civil?
11	¿Cuántos hijos tiene?
12	¿Cuál es su situación laboral actual?
13	¿Conoce acerca de las tecnologías API's (Application Programming Interfaces)?
14	¿Cuál es su nivel de conocimiento sobre API REST?
15	¿Cuál es su nivel de conocimiento sobre GraphQL?
16	¿Ha usado API REST?
17	¿Ha usado GraphQL?
18	¿Qué tan fácil es usar REST?
19	¿Qué tan fácil es usar GraphQL?
20	¿Ha consumido algún API REST?
21	¿Ha consumido algún servicio GraphQL?
22	¿Qué tan fácil es consumir un API REST?
23	¿Qué tan fácil es consumir un API GraphQL?
24	¿Qué tan fácil es aprender REST?
25	¿Qué tan fácil es aprender GraphQL?
26	¿Cuál API prefiere?

Este artefacto se operativizó en un formulario privado de Microsoft Forms 365[2], que no tiene acceso público. Por este motivo se presenta la encuesta completa en un archivo .pdf en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\01 encuesta-demográfica-diagnóstica\encuesta-apis.pdf"*.

B.1.2 Introducción teórica del ambiente de desarrollo

Es una presentación teórica acerca de la arquitectura orientada a microservicios y de las tecnologías usadas en el ambiente de desarrollo, que son la base para el desarrollo de las tareas de formación y las tareas experimentales. Este artefacto es un archivo de

[2] https://www.microsoft.com/es-ww/microsoft-365/online-surveys-polls-quizzes

Microsoft Power Point 365[3] que está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\02 introducción-teórica\01 introducción-ambiente-desarrollo.pptx"*.

B.1.3 Introducción teórica al paradigma REST

Es una presentación teórica acerca del estilo arquitectónico REST que sirvió como base para el desarrollo de APIs REST en las tareas de formación y tareas experimentales. Este artefacto es un archivo de Microsoft Power Point 365 que está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\02 introducción-teórica\02 introducción-rest.pptx"*.

B.1.4 Introducción teórica al paradigma GraphQL

Es una presentación teórica acerca de GraphQL como lenguaje de consulta y tiempo de ejecución de APIs, que sirvió como base para el desarrollo de APIs GraphQL en las tareas de formación y tareas experimentales. Este artefacto es un archivo de Microsoft Power Point 365 que está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\02 introducción-teórica\03 introducción-graphql.pptx"*.

B.1.5 Requisitos: Gestión de Pizzas

Los artefactos de las tareas de requisitos a lo largo del experimento concretamente especifican requisitos de proyectos de desarrollo de APIs. En donde, cada caso contiene cuatro artefactos, de los cuales: las copias de seguridad y archivos de scripts de la base de datos se generaron desde la herramienta de administración *pgAdmin 4* del motor de la base de datos *PostgreSQL*; los diagramas realacionales son archivos de gráficos con extensión PNG generados desde la herramienta Online *Diagrams.net (draw.io)*[4]; y los documentos de requisitos y registro de solución son archivos de Microsoft Excel 365[5].

A continuación, se muestra los requisitos de operaciones básicas CRUD, y diseño de base de datos para administrar Pizzas; que concretamente fueron utilizadas en las siguientes tareas de formación (ver Tabla §6.2): ST02- Ejemplo de creación de proyecto

[3] https://www.microsoft.com/es-ww/microsoft-365/powerpoint
[4] https://www.diagrams.net/
[5] https://www.microsoft.com/es-ww/microsoft-365/excel

de desarrollo, conexión y consulta de base de datos; ST06-Ejemplo de desarrollo de un API REST; y ST10-Ejemplo de desarrollo de un API GraphQL.

Copia de seguridad de la base de datos

Es un archivo binario con extensión .backup que contiene una copia de seguridad de la base de datos de Pizza, usado para que los participantes del experimento crearan esta base de datos en sus ambientes de desarrollo, y puedan desarrollar los ejemplos mostrados en las clases síncronas de la formación. El artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\01 gestión-pizzas\pizza.backup"*.

Script estructura de la BDD y datos de prueba

Es un archivo con extensión .sql que contiene scripts de creación de la estructura, e inserción de datos de prueba de la base de datos Pizza, en lenguaje de consulta estructurado (SQL, por sus siglas en inglés de Structured Query Language). El objetivo de este archivo es brindar a los participantes del experimento una alternativa de creación de la base de datos; o en el caso que el archivo pizza.backup no les funcionó por alguna razón técnica. Este artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\01 gestión-pizzas\pizza.sql"*.

A continuación, se muestra el contenido del archivo pizza.sql:

```
CREATE TABLE pizza
(
    id serial NOT NULL,
    name text NOT NULL,
    origin text NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE ingredient
(
    id serial NOT NULL,
    name text NOT NULL,
    calories integer NOT NULL,
    PRIMARY KEY (id)
);
CREATE TABLE pizza_ingredient
```

```
(
    id serial NOT NULL,
    pizza_id integer NOT NULL,
    ingredient_id integer NOT NULL,
    FOREIGN KEY (ingredient_id) REFERENCES ingredient (id),
    FOREIGN KEY (pizza_id) REFERENCES pizza (id)
);
INSERT INTO pizza(id, name, origin) VALUES (1, 'Pizza Napolitana', 'Italia');
INSERT INTO pizza(id, name, origin) VALUES (2, 'Pizza Carbonara', 'Italia');
INSERT INTO ingredient(id, name, calories) VALUES (1, 'Queso mozzarella', 280);
INSERT INTO ingredient(id, name, calories) VALUES (2, 'Salsa de tomate', 29);
INSERT INTO ingredient(id, name, calories) VALUES (3, 'Anchoas', 10);
INSERT INTO ingredient(id, name, calories) VALUES (4, 'Queso parmesano', 431);
INSERT INTO ingredient(id, name, calories) VALUES (5, 'Cebollas', 20);
INSERT INTO ingredient(id, name, calories) VALUES (6, 'Bacon', 541);
INSERT INTO pizza_ingredient(pizza_id, ingredient_id) VALUES (1, 1);
INSERT INTO pizza_ingredient(pizza_id, ingredient_id) VALUES (1, 2);
INSERT INTO pizza_ingredient(pizza_id, ingredient_id) VALUES (1, 3);
INSERT INTO pizza_ingredient(pizza_id, ingredient_id) VALUES (2, 2);
INSERT INTO pizza_ingredient(pizza_id, ingredient_id) VALUES (2, 4);
INSERT INTO pizza_ingredient(pizza_id, ingredient_id) VALUES (2, 5);
INSERT INTO pizza_ingredient(pizza_id, ingredient_id) VALUES (2, 6);
```

Diagrama relacional de la base de datos

Es un gráfico con extensión .png que contiene el diagrama relacional de la estructura de la base de datos Pizza (ver figura §B.1), que sirvió como complemento para especificar los requisitos de la gestión de Pizza en las tareas de formación. Este artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\01 gestión-pizzas\pizza-diagrama.png"*.

B.1.6 Requisitos: Gestión de Libros

Los artefactos de esta tarea muestran los requisitos de operaciones básicas CRUD, y diseño de base de datos para administrar Libros; que concretamente fueron utilizadas en las siguientes tareas de formación (ver Tabla §6.2): ST03- Taller práctico de creación de un proyecto y consulta de datos; ST07-Taller práctico de creación de un API REST;

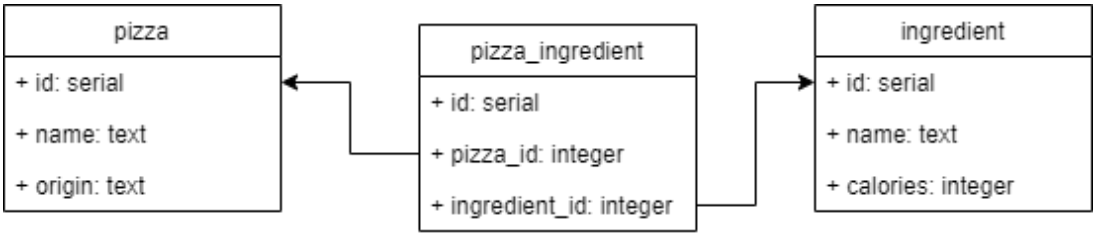


Figura B.1: Diagrama relacional de la base de datos Pizza

y ST11-Taller práctico de creación de un API GraphQL.

Copia de seguridad de la base de datos

Es un archivo binario con extensión .backup que contiene una copia de seguridad de la base de datos de Libros, usado para que los participantes del experimento crearan esta base de datos en sus ambientes de desarrollo, y puedan desarrollar los talleres establecidos en las clases síncronas de formación. El artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\02 gestión-libros\libros.backup"*.

Script estructura de la bdd y datos de prueba

Es un archivo con extensión .sql que contiene scripts de creación de la estructura, e inserción de datos de prueba de la base de datos Libros, en lenguaje de consulta SQL. El objetivo de este archivo es brindar a los participantes del experimento una alternativa de creación de la base de datos. Este artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\01 gestión-libros\libros.sql"*.

A continuación, se muestra el contenido del archivo *libros.sql*:

```
CREATE TABLE autor
(
    aut_id serial NOT NULL,
    aut_nombre text NOT NULL,
    aut_pais text ,
    PRIMARY KEY (aut_id)
);
CREATE TABLE libro
(
    lib_id serial NOT NULL,
```

```
    lib_titulo text NOT NULL,
    lib_editorial text ,
    PRIMARY KEY (lib_id)
);
CREATE TABLE autor_libro
(
    aut_lib_id serial NOT NULL,
    aut_id integer NOT NULL,
    lib_id integer NOT NULL,
    aut_lib_autor_principal boolean,
    PRIMARY KEY (aut_id, lib_id),
    FOREIGN KEY (aut_id) REFERENCES autor (aut_id),
    FOREIGN KEY (lib_id) REFERENCES libro (lib_id)
);
INSERT INTO autor(aut_id, aut_nombre, aut_pais) VALUES (1, 'Mario Piattini', 'Argentina');
INSERT INTO autor(aut_id, aut_nombre, aut_pais) VALUES (2, 'Javier Garzas', 'España');
INSERT INTO libro(lib_id, lib_titulo, lib_editorial) VALUES (1, 'Calidad del producto y proceso software', 'Ra-Ma');
INSERT INTO libro(lib_id, lib_titulo, lib_editorial) VALUES (2, 'Calidad de Sistemas de Información', 'Ra-Ma');
INSERT INTO libro(lib_id, lib_titulo, lib_editorial) VALUES (3, 'Fábricas de software', 'Ra-Ma');
INSERT INTO autor_libro(aut_id, lib_id, aut_lib_autor_principal) VALUES (1, 1, true);
INSERT INTO autor_libro(aut_id, lib_id, aut_lib_autor_principal) VALUES (1, 2, true);
INSERT INTO autor_libro(aut_id, lib_id, aut_lib_autor_principal) VALUES (2, 3, true);
INSERT INTO autor_libro(aut_id, lib_id, aut_lib_autor_principal) VALUES (1, 3, false);
```

Diagrama relacional de la base de datos

Es un gráfico con extención .png que contiene el diagrama relacional de la estructura de la base de datos Libros (ver figura §B.2), que sirvió como complemento para especificar los requisitos de la gestión de Libros en las tareas de formación. Este artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\02 gestión-libros\libros-diagrama.png"*.

Requisitos y registro del tiempo de solución

Es una hoja de cálculo que contiene los requisitos para el desarrollo de un API, consta de diez operaciones de datos CRUD. Además, contiene una sección para registrar la hora de inicio, fin, y el porcentaje de solución de cada requisito. Este artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01*

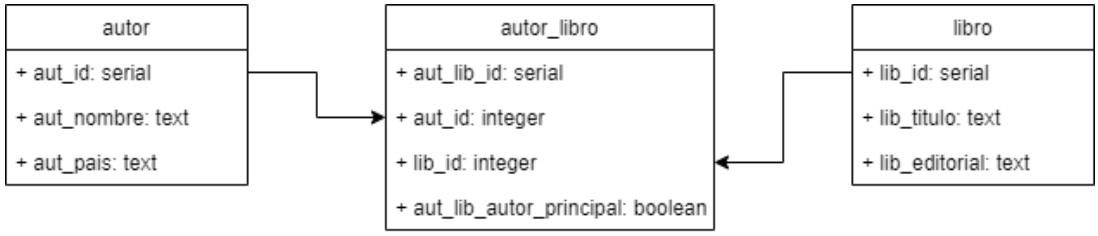


Figura B.2: Diagrama relacional de la base de datos Libros

Formación\03 requisitos-tareas\02 gestión-libros\libros-requisitos-registro.sql".

En la Figura §B.3, se muestra la captura de pantalla de una parte del archivo *libros-requisitos-registro.xlsm*.

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE SOFTWARE
IBARRA - ECUADOR
TALER "Libros"

INDICACIONES:

1	Realice una captura de pantalla por cada tarea resuelta del presente taller. La captura debe incluir el escritorio completo, que evidencie el código fuente en el IDE y el programa ejecutándose. Pegar la captura en las hojas de cada tarea, ejemplo: captura de pantalla de la tarea 1 pegar en la hoja "T-1".
2	Rellene la tabla de tareas.
3	Realice las conlcusiones de la tarea.
4	Cargue el informe y código fuente del taller en la actividad de la asignatura en el portafolio estudiante.

TAREAS: **Nota**: presione (Ctrl + h) para insertar la hora

Nro	Descripción de la tarea	Inicio	Fin	Total	Finalizó	Porcentaj
1	Insertar datos autor			0:00:00	No	0,00%
2	Actualizar datos autor			0:00:00	No	0,00%
3	Eliminar datos autor			0:00:00	No	0,00%
4	Consulta de datos de autores			0:00:00	No	0,00%
5	Consulta de datos autor por id			0:00:00	No	0,00%
6	Asignar un autor a un libro			0:00:00	No	0,00%
7	Eliminar un autor de un libro			0:00:00	No	0,00%
8	Consulta de libros y sus autores			0:00:00	No	0,00%
9	Consulta de autores de un libro			0:00:00	No	0,00%
10	Consulta cuantos autores principales tienen los libros			0:00:00	No	0,00%
			Total	0:00:00		

Figura B.3: Requisitos y registro de solución de la gestión de Libros

B.1.7 Requisitos: Gestión de Blogs

Los artefactos de esta tarea muestran los requisitos de operaciones básicas CRUD, y diseño de base de datos para administrar Blogs; que concretamente fueron utilizadas en las siguientes tareas de formación (ver Tabla §6.2): ST04- Deber de creación de un

proyecto y consulta de datos; ST08-Deber de creación de un API REST; y ST12-Deber de creación de un API GraphQL.

Copia de seguridad de la base de datos

Es un archivo binario con extensión .backup que contiene una copia de seguridad de la base de datos de Blogs, usado para que los participantes del experimento crearan esta base de datos en sus ambientes de desarrollo, y puedan desarrollar los deberes en las horas asíncronas de trabajo establecidas para la formación. El artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\03 gestión-blogs\blogs.backup"*.

Script estructura de la bdd y datos de prueba

Es un archivo con extensión .sql que contiene scripts de creación de la estructura, e inserción de datos de prueba de la base de datos Blogs, en lenguaje de consulta SQL. El objetivo de este archivo es brindar a los participantes del experimento una alternativa de creación de la base de datos. Este artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\01 gestión-blogs\blogs.sql"*.

A continuación, se muestra el contenido del archivo *blogs.sql*:

```
CREATE TABLE autor
(
    aut_id SERIAL not null,
    aut_usuario TEXT not null,
    aut_nombre TEXT not null,
    primary key (aut_id)
);
CREATE TABLE categoria (
    cat_id SERIAL not null,
    cat_titulo TEXT not null,
    primary key (cat_id)
);
CREATE TABLE publicacion (
    pub_id SERIAL not null,
    cat_id INTEGER null,
    aut_id INTEGER null,
    pub_titulo TEXT not null,
```

```
    pub_descripcion TEXT not null,
    primary key (pub_id),
    FOREIGN KEY (aut_id) REFERENCES autor (aut_id),
    FOREIGN KEY (cat_id) REFERENCES categoria (cat_id)
);
CREATE TABLE comentario (
    com_id SERIAL not null,
    pub_id INTEGER null,
    aut_id INTEGER null,
    com_descripcion TEXT not null,
    primary key (com_id),
    FOREIGN KEY (aut_id) REFERENCES autor (aut_id),
    FOREIGN KEY (pub_id) REFERENCES publicacion (pub_id)
);
CREATE TABLE reaccion (
    rea_id SERIAL not null,
    com_id INTEGER not null,
    aut_id INTEGER not null,
    rea_like BOOLEAN not null,
    primary key (com_id, aut_id),
    FOREIGN KEY (aut_id) REFERENCES autor (aut_id),
    FOREIGN KEY (com_id) REFERENCES comentario (com_id)
);
INSERT INTO autor (aut_usuario, aut_nombre) VALUES ('Ponchito12','Francisco Orellana');
INSERT INTO autor (aut_usuario, aut_nombre) VALUES ('MarioXD','Mario Venegas');
INSERT INTO autor (aut_usuario, aut_nombre) VALUES ('Karlita20','Carla Guerra');
INSERT INTO categoria (cat_titulo) VALUES ('Software');
INSERT INTO categoria (cat_titulo) VALUES ('Mecanica');
INSERT INTO categoria (cat_titulo) VALUES ('Moda');
INSERT INTO categoria (cat_titulo) VALUES ('Gastronomia');
INSERT INTO publicacion (cat_id, aut_id, pub_titulo, pub_descripcion) VALUES (1,2,'API REST y GraphQL','¿Cuál le gusta mas?');
INSERT INTO publicacion (cat_id, aut_id, pub_titulo, pub_descripcion) VALUES (1,3,'Java','¿Cuanta experiencia tienen?');
INSERT INTO publicacion (cat_id, aut_id, pub_titulo, pub_descripcion) VALUES (2,2,'Carros electricos', 'Donde se llena la bate-
ria');
INSERT INTO publicacion (cat_id, aut_id, pub_titulo, pub_descripcion) VALUES (4,3,'El mejor hornado','¿Donde es el mejor hor-
nado?');
INSERT INTO comentario (pub_id, aut_id, com_descripcion) VALUES (1,1,'REST');
```

```
INSERT INTO comentario (pub_id, aut_id, com_descripcion) VALUES (1,2,'No lo se');
INSERT INTO comentario (pub_id, aut_id, com_descripcion) VALUES (4,3,'Otavalo');
INSERT INTO reaccion (com_id, aut_id, rea_like) VALUES (1,2,true);
INSERT INTO reaccion (com_id, aut_id, rea_like) VALUES (3,2,true);
INSERT INTO reaccion (com_id, aut_id, rea_like) VALUES (2,3,true);
```

Diagrama relacional de la base de datos

Es un gráfico con extención .png que contiene el diagrama relacional de la estructura de la base de datos Blogs (ver figura §B.4), que sirvió como complemento para especificar los requisitos de la gestión de Libros en las tareas de formación. Este artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01 Formación\03 requisitos-tareas\03 gestión-blogs\blogs-diagrama.png"*.

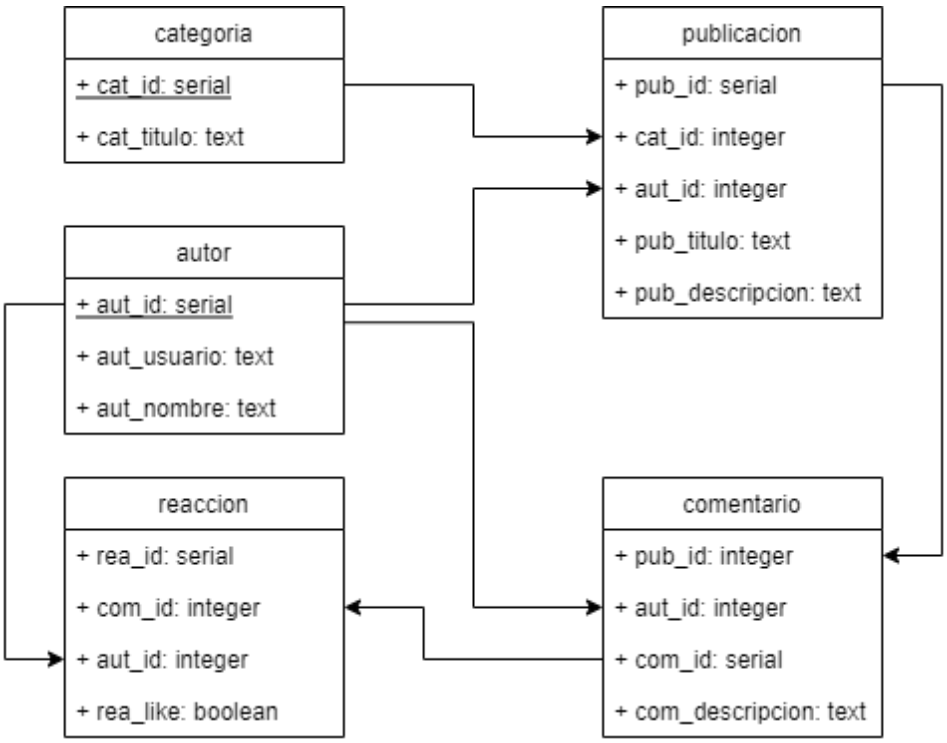


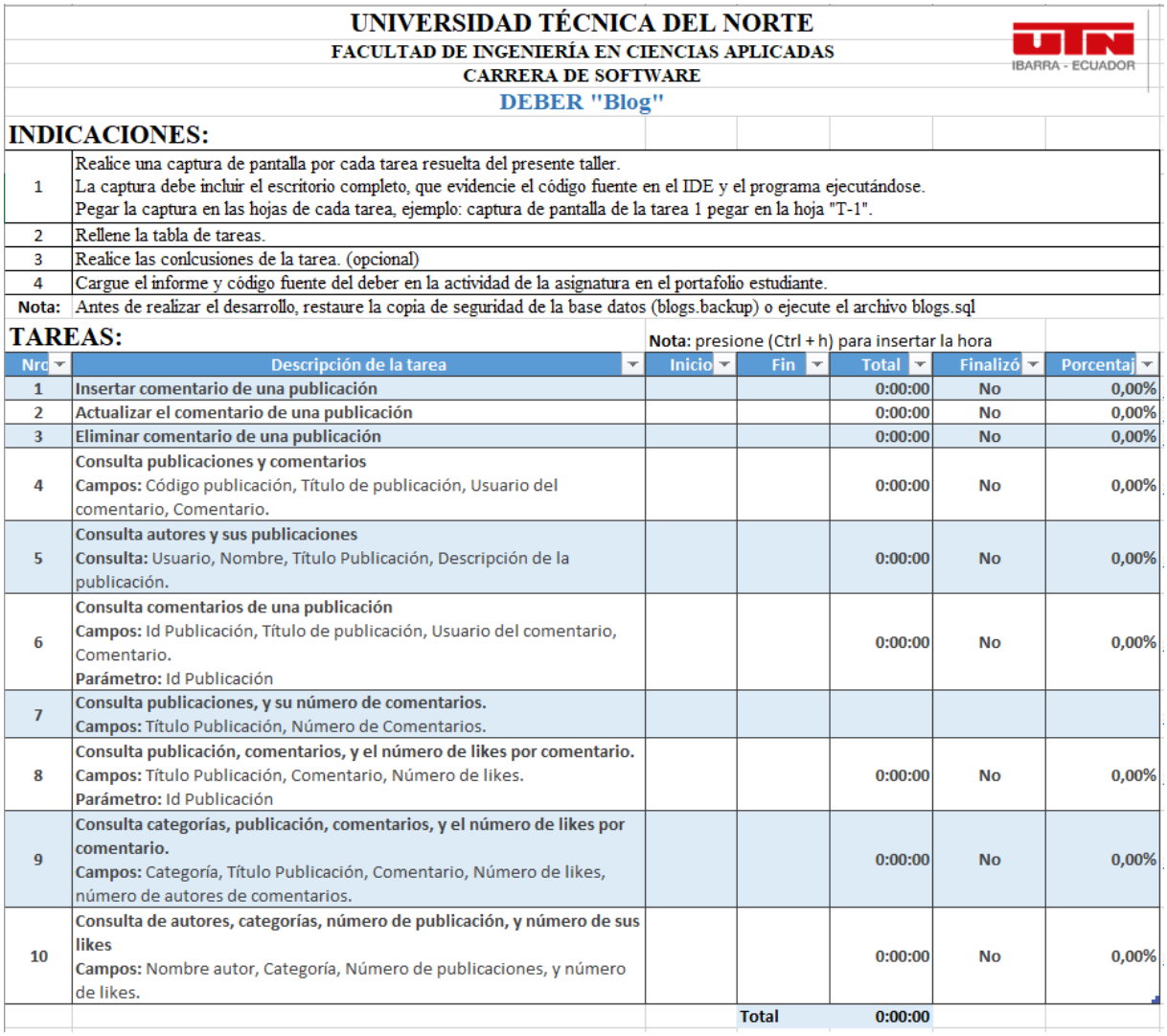
Figura B.4: Diagrama relacional de la base de datos Blogs

Requisitos y registro del tiempo de solución

Es una hoja de cálculo que contiene los requisitos para el desarrollo de un API, consta de diez operaciones de datos CRUD. Además, contiene una sección para registrar la hora de inicio, fin, y el porcentaje de solución de cada requisito. Este artefacto está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\01*

Formación\03 requisitos-tareas\03 gestión-blogs\blogs-requisitos-registro.sql".

En la Figura §B.5, se muestra la captura de pantalla de una parte del archivo *blogs-requisitos-registro.xlsm*.



UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE SOFTWARE
IBARRA - ECUADOR
DEBER "Blog"

INDICACIONES:

1	Realice una captura de pantalla por cada tarea resuelta del presente taller. La captura debe incluir el escritorio completo, que evidencie el código fuente en el IDE y el programa ejecutándose. Pegar la captura en las hojas de cada tarea, ejemplo: captura de pantalla de la tarea 1 pegar en la hoja "T-1".
2	Rellene la tabla de tareas.
3	Realice las conlcusiones de la tarea. (opcional)
4	Cargue el informe y código fuente del deber en la actividad de la asignatura en el portafolio estudiante.
Nota:	Antes de realizar el desarrollo, restaure la copia de seguridad de la base datos (blogs.backup) o ejecute el archivo blogs.sql

TAREAS: **Nota:** presione (Ctrl + h) para insertar la hora

Nro	Descripción de la tarea	Inicio	Fin	Total	Finalizó	Porcentaj
1	**Insertar comentario de una publicación**			0:00:00	No	0,00%
2	**Actualizar el comentario de una publicación**			0:00:00	No	0,00%
3	**Eliminar comentario de una publicación**			0:00:00	No	0,00%
4	**Consulta publicaciones y comentarios** **Campos:** Código publicación, Título de publicación, Usuario del comentario, Comentario.			0:00:00	No	0,00%
5	**Consulta autores y sus publicaciones** **Consulta:** Usuario, Nombre, Título Publicación, Descripción de la publicación.			0:00:00	No	0,00%
6	**Consulta comentarios de una publicación** **Campos:** Id Publicación, Título de publicación, Usuario del comentario, Comentario. **Parámetro:** Id Publicación			0:00:00	No	0,00%
7	**Consulta publicaciones, y su número de comentarios.** **Campos:** Título Publicación, Número de Comentarios.					
8	**Consulta publicación, comentarios, y el número de likes por comentario.** **Campos:** Título Publicación, Comentario, Número de likes. **Parámetro:** Id Publicación			0:00:00	No	0,00%
9	**Consulta categorías, publicación, comentarios, y el número de likes por comentario.** **Campos:** Categoría, Título Publicación, Comentario, Número de likes, número de autores de comentarios.			0:00:00	No	0,00%
10	**Consulta de autores, categorías, número de publicación, y número de sus likes** **Campos:** Nombre autor, Categoría, Número de publicaciones, y número de likes.			0:00:00	No	0,00%
			Total	0:00:00		

Figura B.5: Requisitos y registro de solución de la gestión de Blogs

B.2 EXPERIMENTO

En esta sección, se detalla los artefactos usados en las tareas experimentales del experimento descritas en la sección §6.2.7. Estas tareas consisten en especificar los requisitos de tres casos de proyectos de desarrollo de APIs. Cada caso contiene cuatro ar-

tefactos, de los cuales: las copias de seguridad y archivos de scripts de la base de datos se generan desde la herramienta de administración *pgAdmin 4* del motor de la base de datos *PostgreSQL*; los diagramas realacionales son archivos de gráficos con extensión PNG generados desde la herramienta Online *Diagrams.net (draw.io)*; y los documentos de requisitos y registro de solución son archivos de Microsoft Excel 365.

B.2.1 Requisitos: Gestión de Eventos

Esta tarea experimental contiene los requisitos y diseño de base de datos para el desarrollo de un API (REST o GraphQL) que contenga las operaciones básicas CRUD de la gestión de Eventos.

Copia de seguridad de la base de datos

Es un archivo con extensión .backup que contiene una copia de seguridad de la base de datos de Eventos, usado por los participantes del experimento para crear la base de datos en sus ambientes de desarrollo. El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\01 gestión-eventos\eventos.backup"*.

Script estructura de la bdd y datos de prueba

Es un archivo con extensión .sql que contiene los scripts de creación de la estructura de la base de datos Eventos, e inserción de datos de prueba en lenguaje SQL. El objetivo de este archivo es brindar a los participantes una alternativa de creación de la base de datos. El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\01 gestión-eventos\eventos.sql"*.

A continuación, se muestra el contenido del archivo *eventos.sql*:

```
CREATE TABLE participante (
    par_id SERIAL not null,
    par_nombre TEXT not null,
    par_cedula TEXT not null,
    par_correo TEXT not null,
    PRIMARY KEY (par_id)
);
CREATE TABLE sala (
    sal_id SERIAL not null,
```

```
    sal_nombre TEXT not null,
    sal_descripcion TEXT not null,
    sal_estado BOOLEAN not null,
    PRIMARY KEY (sal_id)
);
CREATE TABLE evento (
    eve_id SERIAL not null,
    sal_id INTEGER not null,
    eve_nombre TEXT not null,
    eve_costo DECIMAL(8,2) not null,
    PRIMARY KEY (eve_id),
    FOREIGN KEY (sal_id) references sala (sal_id)
);
CREATE TABLE evento_participante (
    eve_id INTEGER not null,
    par_id INTEGER not null,
    evepar_cantidad DECIMAL(8,2) not null,
    PRIMARY KEY (eve_id, par_id),
    FOREIGN KEY (eve_id) references evento (eve_id),
    FOREIGN KEY (par_id) references participante (par_id)
);
```

```
INSERT INTO sala (sal_nombre, sal_descripcion, sal_estado) values ('Gala Club', 'Una opción diferente para realizar sus eventos sociales, en un ambiente de distinción, con excelente gastronomía y el servicio cordial', true);
INSERT INTO sala (sal_nombre, sal_descripcion, sal_estado) values ('Hotel Ajaví', 'Tiene un area de 560m2 y capacidad para 600 personas en montaje tipo teatro, 450 personas con sillas para fiesta. El salón esta lujosamente alfombrado, tiene aire acondicionado, iluminación sectorizada, sistema de sonido de potencia y de palabra. Su división plegable lo convierte en 2 salones de 210m2 y 350m2 con ingresos independientes.', true);
INSERT INTO sala (sal_nombre, sal_descripcion, sal_estado) values ('Alborada', 'Planificación de Bodas, Atención de Recepciones, Eventos al aire libre, Fiestas Infantiles, Seminarios, Conferencias, servicio de alimentos a domicilio.',true);
INSERT INTO sala (sal_nombre, sal_descripcion, sal_estado) values ('Hosteria Rancho de Carolina', 'Con capacidad para 250 personas, apto para toda clase de compromisos. Disfrute aquí de la más exquisita cocina nacional e internacional.', true);
INSERT INTO sala (sal_nombre, sal_descripcion, sal_estado) values ('Hotel Imperio del Sol', 'Discoteka privada. Karaoke. Parqueadero privado con guardia de seguridad 24 horas. Paraiso de Imbabura a solo 5 minutos de Ibarra. Un sitio acogedor rodeado de los más hermosos y variados paisajes, situado a orillas de la histórica y cálida laguna de Yahuarcocha.', true);
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Daniel Guerra', '1004100901','mguerra@utn.edu.ec');
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Ricardo Avila', '1004100902','ravila@utn.edu.ec');
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Dennis Chicaiza', '1004100903','dchicaiza@utn.edu.ec');
```

```
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Ervin Cabascango', '1004100904','dcabascango@utn.edu.ec');
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Ronal Moreira', '1004100905','rmoreira@utn.edu.ec');
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Gabriel Garzón', '1004100906','ggarzon@utn.edu.ec');
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Daniel Mejia', '1004100907','dmejia@utn.edu.ec');
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Fabio Checa', '1004100908','fcheca@utn.edu.ec');
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Edward Elric', '1004100909','eelric@utn.edu.ec');
INSERT INTO participante (par_nombre, par_cedula, par_correo) values ('Dark Alchemist', '1004100900','dalchemist@utn.edu.ec');
INSERT INTO evento (sal_id, eve_nombre, eve_costo) values (1, 'Grado Colegio Sanchez y Cifuentes', 200.50);
INSERT INTO evento (sal_id, eve_nombre, eve_costo) values (2, 'Boda Juan Rigoberto', 405.00);
INSERT INTO evento (sal_id, eve_nombre, eve_costo) values (3, 'Grado UTN Software', 106.66);
INSERT INTO evento_participante (eve_id, par_id, evepar_cantidad) values (1, 1, 2);
INSERT INTO evento_participante (eve_id, par_id, evepar_cantidad) values (1, 2, 2);
INSERT INTO evento_participante (eve_id, par_id, evepar_cantidad) values (2, 3, 2);
INSERT INTO evento_participante (eve_id, par_id, evepar_cantidad) values (2, 4, 2);
INSERT INTO evento_participante (eve_id, par_id, evepar_cantidad) values (3, 6, 2);
INSERT INTO evento_participante (eve_id, par_id, evepar_cantidad) values (3, 10, 2);
```

Diagrama relacional de la base de datos

Es un archivo con extensión .png que contiene el diagrama relacional de la estructura de la base de datos Eventos (ver figura §6.2), que sirvió como complemento para especificar los requisitos para la Gestión de Eventos en la tarea experimental. El archivo está disponible en la siguiente dirección del paquete de laboratorio: El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\01 gestión-eventos\eventos-diagrama.png"*.

Requisitos y registro del tiempo de solución

Es una hoja de cálculo que contiene los requisitos para el desarrollo de un API, consta de diez operaciones de datos CRUD. Además, contiene una sección para registrar la hora de inicio, fin, y el porcentaje de solución de cada requisito. En este sentido, se construyó dos artefactos con el mismo contenido, pero diferenciando el paradigma (API-REST / API-GraphQL) en el título de las tareas. Los artefactos están disponibles en las siguientes direcciones del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\01 gestión-eventos\eventos-requisitos-registro-rest.xlsm"*, y *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\01 gestión-eventos\eventos-requisitos- registro-graphql.xlsm"*.

En la Figura §B.6, se muestra la captura de pantalla de una parte del archivo *eventos-requisitos-registro-rest.xlsm*.

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE SOFTWARE
IBARRA - ECUADOR

CASO "Eventos"

INDICACIONES:

1	Realice una captura de pantalla por cada tarea resuelta de la actividad. La captura debe incluir el escritorio completo, que evidencie el código fuente en el IDE y el programa ejecutándose. Pegar la captura en las hojas de cada tarea, ejemplo: captura de pantalla de la tarea 1 pegar en la hoja "T-1".
2	Rellene la tabla de tareas con los ***tiempos de las tareas*** .
3	Usar la aplicación "***aTube Catcher***" para guardar el video del desarrollo de la actividad y luego cargar el video en Microsoft Teams.
4	Registre sus ***observaciones*** acerca de la actividad.
5	Cargue el informe y código fuente (sin node_modules) de la actividad en el ***portafolio estudiante*** .

TAREAS (API-REST): **Nota**: presione (Ctrl + h) para insertar la hora

Nro	Descripción de la tarea	Inicio	Fin	Total	Finalizó	Porcentaj
1	**Insertar un registro de evento**			0:00:00	No	0,00%
2	**Actualizar un registro de evento**			0:00:00	No	0,00%
3	**Eliminar un registro de evento**			0:00:00	No	0,00%
4	**Consultar los registros de los eventos**			0:00:00	No	0,00%
5	**Consultar el registro de un evento, filtrado por el campo:eve_id**			0:00:00	No	0,00%
6	**Asignar un participante a un evento**			0:00:00	No	0,00%
7	**Eliminar un participante de un evento**			0:00:00	No	0,00%
8	**Consulta compuesta:** Consultar las salas y sus eventos **tablas:** sala, evento **campos:** sal_id, sal_nombre, eve_nombre, y eve_costo			0:00:00	No	0,00%
9	**Consulta compuesta:** Consultar un evento y sus participantes **tablas:** evento, evento_participante, participante **campos:** eve_nombre, par_cedula, par_nombre, y eve_par_pago			0:00:00	No	0,00%
10	**Consulta compuesta:** Consultar las salas y eventos de un participante **tablas:** sala, evento, evento_participante, participante **campos:** sal_nombre, eve_nombre, eve_costo, par_nombre, y eve_par_pago **parámetro:** par_id			0:00:00	No	0,00%

Figura B.6: Requisitos y registro de solución de la gestión de eventos

B.2.2 Requisitos: Gestión de Facturas

Esta tarea experimental contiene los requisitos y diseño de base de datos para el desarrollo de un API (REST o GraphQL) que contenga las operaciones básicas CRUD de la gestión de Facturas.

Copia de seguridad de la base de datos

Es un archivo con extensión .backup que contiene una copia de seguridad de la base de datos de Facturas, usado por los participantes del experimento para crear la base

de datos en sus ambientes de desarrollo. El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\02 gestión-facturas\facturas.backup"*.

Script estructura de la bdd y datos de prueba

Es un archivo con extensión .sql que contiene los scripts de creación de la estructura de la base de datos Facturas, e inserción de datos de prueba en lenguaje SQL. El objetivo de este archivo es brindar a los participantes una alternativa de creación de la base de datos. El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\02 gestión-facturas\facturas.sql"*.

A continuación, se muestra el contenido del archivo facturas.sql:

```
CREATE TABLE cliente (
    cli_id SERIAL not null,
    cli_nombre TEXT not null,
    cli_correo TEXT not null,
    cli_estado BOOLEAN not null,
    PRIMARY KEY (cli_id)
);
CREATE TABLE producto (
    pro_id SERIAL not null,
    pro_nombre TEXT not null,
    pro_pvp DECIMAL(8,2) not null,
    pro_estado BOOLEAN not null,
    PRIMARY KEY (pro_id)
);
CREATE TABLE factura (
    fac_id SERIAL not null,
    cli_id INTEGER null,
    fac_fecha TEXT not null,
    PRIMARY KEY (fac_id),
    FOREIGN KEY (cli_id) references cliente (cli_id)
);
CREATE TABLE factura_producto (
    pro_id INTEGER not null,
    fac_id INTEGER not null,
```

```
    facpro_cantidad INTEGER not null,
    facpro_pvp DECIMAL(8,2) not null,
    PRIMARY KEY (pro_id, fac_id),
    FOREIGN KEY (fac_id) references factura (fac_id),
    FOREIGN KEY (pro_id) references producto (pro_id)
);
insert into cliente (cli_nombre, cli_correo, cli_estado) values ('Mario Guerrero', 'mguerrero@utn.edu.ec', true);
insert into cliente (cli_nombre, cli_correo, cli_estado) values ('Daniel Guerra', 'dguerra@utn.edu.ec', true);
insert into cliente (cli_nombre, cli_correo, cli_estado) values ('Ricardo Avila', 'ravila@utn.edu.ec', true);
insert into cliente (cli_nombre, cli_correo, cli_estado) values ('Gabriel Garzón', 'ggarzon@utn.edu.ec', true);
insert into cliente (cli_nombre, cli_correo, cli_estado) values ('Dark Alchemist', 'dalchemist@utn.edu.ec', true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Leche', 0.85, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Tallarin', 0.75, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Pan integral', 1.49, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Cocoa', 0.95, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Atún', 1.00, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Caldo maggi', 1.55, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Cereal kelloggs', 1.00, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Aceite', 1.89, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Canguil', 0.40, true);
insert into producto (pro_nombre, pro_pvp, pro_estado) values ('Gelatina', 2.10, true);
insert into factura (cli_id, fac_fecha) values (1, '05/06/2021');
insert into factura (cli_id, fac_fecha) values (3, '04/06/2021');
insert into factura (cli_id, fac_fecha) values (5, '03/06/2021');
insert into factura_producto (pro_id, fac_id, facpro_cantidad, facpro_pvp) values (1, 1, 2, 0.90);
insert into factura_producto (pro_id, fac_id, facpro_cantidad, facpro_pvp) values (2, 1, 1, 0.80);
insert into factura_producto (pro_id, fac_id, facpro_cantidad, facpro_pvp) values (3, 2, 1, 1.60);
insert into factura_producto (pro_id, fac_id, facpro_cantidad, facpro_pvp) values (4, 2, 1, 1.00);
insert into factura_producto (pro_id, fac_id, facpro_cantidad, facpro_pvp) values (5, 3, 1, 1.10);
insert into factura_producto (pro_id, fac_id, facpro_cantidad, facpro_pvp) values (6, 3, 1, 1.60);
```

Diagrama relacional de la base de datos

Es un archivo con extensión .png que contiene el diagrama relacional de la estructura de la base de datos Facturas, (ver figura §6.3), que sirvió como complemento de los requisitos de software del caso Gestión de Facturas en la tarea experimental. El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-*

laboratorio\02 Experimento\01 requisitos-tareas\02 gestión-facturas\facturas-diagrama.png".

Requisitos y registro del tiempo de solución

Es una hoja de cálculo que contiene los requisitos para el desarrollo de un API, consta de diez operaciones de datos CRUD. Además, contiene una sección para registrar la hora de inicio, fin, y el porcentaje de solución de cada requisito. En este sentido, se construyó dos artefactos con el mismo contenido, pero diferenciando el paradigma (API-REST / API-GraphQL) en el título de las tareas. Los artefactos están disponibles en las siguientes direcciones del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\01 gestión-facturas\facturas-requisitos-registro-rest.xlsm"*, y *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\02 gestión-facturas\facturas-requisitos- registro-graphql.xlsm"*.

En la Figura §B.7, se muestra la captura de pantalla de una parte del archivo *facturas-requisitos-registro-rest.xlsm*.

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE SOFTWARE
IBARRA - ECUADOR
CASO "Facturas"

INDICACIONES:

1	Realice una captura de pantalla por cada tarea resuelta de la actividad. La captura debe incluir el escritorio completo, que evidencie el código fuente en el IDE y el programa ejecutándose. Pegar la captura en las hojas de cada tarea, ejemplo: captura de pantalla de la tarea 1 pegar en la hoja "T-1".
2	Rellene la tabla de tareas con los ***tiempos de las tareas*** .
3	Usar la aplicación "***aTube Catcher***" para guardar el video del desarrollo de la actividad y luego cargar el video en Microsoft Teams.
4	Registre sus ***observaciones*** acerca de la actividad.
5	Cargue el informe y código fuente (sin node_modules) de la actividad en el ***portafolio estudiante*** .

TAREAS (API-REST): **Nota:** presione (Ctrl + h) para insertar la hora

Nro	Descripción de la tarea	Inicio	Fin	Total	Finalizó	Porcentaj
1	**Insertar un registro de factura**			0:00:00	No	0,00%
2	**Actualizar un registro de factura**			0:00:00	No	0,00%
3	**Eliminar un registro de factura**			0:00:00	No	0,00%
4	**Consultar los registros de las facturas**			0:00:00	No	0,00%
5	**Consultar el registro de una factura, filtrado por el campo: fac_id**			0:00:00	No	0,00%
6	**Asignar un producto a una factura**			0:00:00	No	0,00%
7	**Eliminar un producto de una factura**			0:00:00	No	0,00%
8	**Consulta compuesta:** Consultar los clientes y sus facturas **tablas:** cliente, factura **campos:** cli_id, cli_nombre, fac_id, y fac_fecha			0:00:00	No	0,00%
9	**Consulta compuesta:** Consultar una factura y sus productos **tablas:** factura, factura_producto, producto **campos:** fac_id, pro_id, pro_nombre, y facpro_cantidad **parámetro:** fac_id			0:00:00	No	0,00%
10	cliente **tablas:** cliente, factura, factura_producto, producto **campos:** cli_nombre, fac_id, pro_nombre, pro_pvp, y facpro_cantidad **parámetro:** cli_id			0:00:00	No	0,00%

Figura B.7: Requisitos y registro de solución de la gestión de facturas

B.2.3 Requisitos: Gestión de Notas

Esta tarea experimental contiene los requisitos y diseño de base de datos para el desarrollo de un API (REST o GraphQL) que contenga las operaciones básicas CRUD de la gestión de Notas.

Copia de seguridad de la base de datos

Es un archivo con extensión .backup que contiene una copia de seguridad de la base de datos de Notas, usado por los participantes del experimento para crear la base de datos en sus ambientes de desarrollo. El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\03 gestión-notas\notas.backup"*.

Script estructura de la bdd y datos de prueba

Es un archivo con extensión .sql que contiene los scripts de creación de la estructura de la base de datos Notas, e inserción de datos de prueba en lenguaje SQL. El objetivo de este archivo es brindar a los participantes una alternativa de creación de la base de datos. El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\03 gestión-notas\notas.sql"*.

A continuación, se muestra el contenido del archivo *notas.sql*:

```
CREATE TABLE estudiante (
    est_id SERIAL not null,
    est_nombre TEXT not null,
    est_cedula TEXT not null,
    est_correo TEXT not null,
    PRIMARY KEY (est_id)
);
CREATE TABLE materia (
    mat_id SERIAL not null,
    mat_nombre TEXT not null,
    mat_nivel TEXT not null,
    mat_paralelo TEXT not null,
    mat_estado BOOLEAN not null,
    PRIMARY KEY (mat_id)
);
CREATE TABLE actividad (
```

```
    act_id SERIAL not null,
    mat_id INTEGER null,
    act_nombre TEXT not null,
    act_descripcion TEXT not null,
    PRIMARY KEY (act_id),
    FOREIGN KEY (mat_id) references materia (mat_id)
);
CREATE TABLE actividad_nota (
    act_id INTEGER not null,
    est_id INTEGER not null,
    actnot_nota DECIMAL(8,2) not null,
    PRIMARY KEY (act_id, est_id),
    FOREIGN KEY (act_id) references actividad (act_id),
    FOREIGN KEY (est_id) references estudiante (est_id)
);
insert into materia (mat_nombre, mat_nivel, mat_paralelo, mat_estado) values ('Programación Orientada Objetos', 'Primero', 'A',true);
insert into materia (mat_nombre, mat_nivel, mat_paralelo, mat_estado) values ('Base de Datos', 'Tercero', 'A',true);
insert into materia (mat_nombre, mat_nivel, mat_paralelo, mat_estado) values ('Desarrollo de software para MIPYMES', 'Quinto',
'A',true);
insert into materia (mat_nombre, mat_nivel, mat_paralelo, mat_estado) values ('Construcción de software', 'Quinto', 'B',true);
insert into materia (mat_nombre, mat_nivel, mat_paralelo, mat_estado) values ('Inteligencia artificial', 'Sexto', 'B',true);
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Daniel Guerra', '1004100901','mguerra@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Ricardo Avila', '1004100902','ravila@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Dennis Chicaiza', '1004100903','dchicaiza@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Ervin Cabascango', '1004100904','dcabascango@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Ronal Moreira', '1004100905','rmoreira@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Gabriel Garzón', '1004100906','ggarzon@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Daniel Mejia', '1004100907','dmejia@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Fabio Checa', '1004100908','fcheca@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Edward Elric', '1004100909','eelric@utn.edu.ec');
insert into estudiante (est_nombre, est_cedula, est_correo) values ('Dark Alchemist', '1004100900','dalchemist@utn.edu.ec');
insert into actividad (mat_id, act_nombre, act_descripcion) values (1,'Objetos en Java','Declaración y acceso a funciones de un ob-
jeto');
insert into actividad (mat_id, act_nombre, act_descripcion) values (2,'Diseño Base de Datos','Diseño en Power Designer y exporta-
ción script');
insert into actividad (mat_id, act_nombre, act_descripcion) values (5,'Redes neuronales','Explicación y ejercicio con CNN');
insert into actividad_nota (act_id, est_id, actnot_nota) values (1, 1, 8.00);
```

```
insert into actividad_nota (act_id, est_id, actnot_nota) values (1, 2, 6.50);
insert into actividad_nota (act_id, est_id, actnot_nota) values (2, 3, 7.00);
insert into actividad_nota (act_id, est_id, actnot_nota) values (2, 4, 5.00);
insert into actividad_nota (act_id, est_id, actnot_nota) values (3, 6, 7.70);
insert into actividad_nota (act_id, est_id, actnot_nota) values (3, 10, 10.00);
```

Diagrama relacional de la base de datos

Es un archivo con extensión .png que contiene el diagrama relacional de la estructura de la base de datos Notas, (ver figura §6.4), que sirvió como complemento de los requisitos de software del caso Gestión de Notas en la tarea experimental. El archivo está disponible en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\03 gestión-notas\notas-diagrama.png"*.

Requisitos y registro del tiempo de solución

Es una hoja de cálculo que contiene los requisitos para el desarrollo de un API, consta de diez operaciones de datos CRUD. Además, contiene una sección para registrar la hora de inicio, fin, y el porcentaje de solución de cada requisito. En este sentido, se construyó dos artefactos con el mismo contenido, pero diferenciando el paradigma (API-REST / API-GraphQL) en el título de las tareas. Los artefactos están disponibles en las siguientes direcciones del paquete de laboratorio: *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\01 gestión-notas\notas-requisitos-registro-rest.xlsm"*, y *"Paquete-de-laboratorio\02 Experimento\01 requisitos-tareas\02 gestión-notas\notas-requisitos-registro-graphql.xlsm"*.

En la Figura §B.8, se muestra la captura de pantalla de una parte del archivo *notas-requisitos-registro-rest.xlsm*.

B.3 Análisis

B.3.1 Encuesta de Autoevaluación

En esta sección se utilizó la misma encuesta del Anexo §B.1.1 pero sin la información demográfica. La encuesta se ejecutó luego de realizar el experimento, con la finalidad que los participantes auto evalúen su percepción de facilidad de uso, consumo y aprendizaje de los paradigmas estudiados.

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE SOFTWARE
IBARRA - ECUADOR

CASO "Notas"

INDICACIONES:

1	Realice una captura de pantalla por cada tarea resuelta de la actividad. La captura debe incluir el escritorio completo, que evidencie el código fuente en el IDE y el programa ejecutándose. Pegar la captura en las hojas de cada tarea, ejemplo: captura de pantalla de la tarea 1 pegar en la hoja "T-1".
2	Rellene la tabla de tareas con los ***tiempos de las tareas*** .
3	Usar la aplicación "***aTube Catcher***" para guardar el video del desarrollo de la actividad y luego cargar el video en Microsoft Teams.
4	Registre sus ***observaciones*** acerca de la actividad.
5	Cargue el informe y código fuente (sin node_modules) de la actividad en el ***portafolio estudiante*** .

TAREAS (API-REST):

Nota: presione (Ctrl + h) para insertar la hora

Nro	Descripción de la tarea	Inicio	Fin	Total	Finalizó	Porcentaj
1	**Insertar un registro de actividad**			0:00:00	No	0,00%
2	**Actualizar un registro de actividad**			0:00:00	No	0,00%
3	**Eliminar un registro de actividad**			0:00:00	No	0,00%
4	**Consultar los registros de las actividades**			0:00:00	No	0,00%
5	**Consultar el registro de una actividad, filtrado por el campo: act_id**			0:00:00	No	0,00%
6	**Asignar un estudiante a una actividad**			0:00:00	No	0,00%
7	**Eliminar un estudiante de una actividad**			0:00:00	No	0,00%
8	**Consulta compuesta:** Consultar las materias y sus actividades **tablas:** materia, actividad **campos:** mat_id, mat_nombre, act_nombre, y act_descripcion			0:00:00	No	0,00%
9	**Consulta compuesta:** Consultar una actividad y la nota de los estudiantes **tablas:** actividad, actividad_nota, estudiante **campos:** act_nombre, est_cedula, est_nombre, y actnot_nota **parámetro:** act_id			0:00:00	No	0,00%
10	**Consulta compuesta:** Consultar las materias, actividades y notas, de un estudiante **tablas:** materia, actividad, actividad_nota, estudiante **campos:** mat_nombre, act_nombre, act_descripcion, est_nombre, y			0:00:00	No	0,00%

Figura B.8: Requisitos y registro de solución de la gestión de notas

Este artefacto se operativizó en un formulario privado de Microsoft Forms 365, que no tiene acceso público. Por este motivo se presenta la encuesta en un archivo .pdf en la siguiente dirección del paquete de laboratorio: *"Paquete-de-laboratorio\03 Análisis\01 encuesta-autoevaluación\encuesta-auto-evaluación.pdf"*.

Bibliografía

Al final de cada referencia bibliográfica aparece un enlace a cada página en que se cita dicha referencia.

[1] Tobias Andersson, Håkan Reinholdsson, Fredrik Stridh, and Dawit Mengistu. *REST API vs GraphQL: A literature and experimental study*. PhD thesis, Kristianstad University, 2021. (pages 79, 84, 223).

[2] Hilary Arksey and Lisa O'Malley. Scoping studies: towards a methodological framework. *International journal of social research methodology*, 8(1):19–32, 2005. (page 20).

[3] Hilary Arksey and Lisa O'Malley. Scoping studies: Towards a methodological framework. *International Journal of Social Research Methodology: Theory and Practice*, 8(1):19–32, 2005. (page 63).

[4] Ilmari Autio, Petri Vuorimaa, and Virpi Huhtinen. *News API implementation with serverless GraphQL*. PhD thesis, Aalto University, Espoo, 2020. (pages 79, 221).

[5] Georgios Bakirtzis, Tim Sherburne, Stephen Adams, Barry Horowitz, Peter Beling, and Cody Fleming. An ontological metamodel for cyber-physical system safety, security, and resilience coengineering. *Software and Systems Modeling*, page 25, 2021. (pages 79, 222).

[6] Victor R Basili. Software modeling and measurement: the goal/question/metric paradigm. *Encyclopedia of Software Engineering*, 1994. (pages 94, 110).

[7] Saša Baškarada, Vivian Nguyen, and Andy Koronios. Architecting Microservices: Practical Opportunities and Challenges. *Journal of Computer Information Systems*, 00(00):1–9, 2018. (page 4).

[8] Jorge Biolchini, Paula Gomes Mian, Ana Candida Cruz Natali, and Guilherme Horta Travassos. Systematic review in software engineering. *System engineering and computer science department COPPE/UFRJ, Technical Report ES*, 679(05):45, 2005. (page 20).

[9] Pierre Bourque and Richard E. Fairley. *SWEBOK v.3 - Guide to the Software Engineering - Body of Knowledge.* IEEE Computer Society, New Jersey, version 3. edition, 2014. (page 70).

[10] Gleison Brito, Thais Mombach, and Marco Tulio Valente. Migrating to GraphQL: A Practical Assessment. In *SANER 2019 - Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering*, pages 140–150, Hangzhou, 2019. IEEE. (pages 6, 78, 79, 84, 85, 220).

[11] Gleison Brito and Marco Tulio Valente. REST vs GraphQL: A controlled experiment. In *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020*, pages 81–91, Salvador, 3 2020. Institute of Electrical and Electronics Engineers Inc. (pages 29, 78, 79, 84, 85, 221).

[12] John Brooke. SUS: A Retrospective. *Journal of Usability Studies*, 8(2):29–40, 2013. (page 45).

[13] Mike Bryant. GraphQL for archival metadata: An overview of the EHRI GraphQL API. In *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, volume 2018-Janua, pages 2225–2230, Boston, 12 2017. Institute of Electrical and Electronics Engineers Inc. (pages 78, 79, 220).

[14] Reinhard Budde, Karlheinz Kautz, Karin Kuhlenkamp, and Heinz Züllighoven. *Prototyping An Approach to Evolutionary System Development*. Springer-Verlag Berlin Heidelberg, Berlin, 1992. (page 70).

[15] David Budgen, Mark Turner, Pearl Brereton, and Barbara A Kitchenham. Using mapping studies in software engineering. In *Ppig*, volume 8, pages 195–204, 2008. (page 20).

[16] Valerio Cassani, Stefano Gianelli, Maristella Matera, Riccardo Medana, Elisa Quintarelli, Letizia Tanca, and Vittorio Zaccaria. On the Role of Context in the Design of Mobile Mashups. In Gaedke M. Daniel F., editor, *2nd International Rapid Mashup Challenge, RMC 2016*, volume 696, pages 108–128. Springer Verlag, Lugano, 2017. (pages 78, 79, 220).

[17] David Čechák and Bruno Rossi. *Using GraphQL for Content Delivery in Kentico Cloud*. PhD thesis, Masaryk University, 2017. (pages 79, 84, 85, 220).

[18] Lujing CEN and Evan PATTON. Block Affordances for GraphQL in MIT App Inventor. In *International Conference on Computational Thinking Education 2019*, pages

147–150, Hong Kong, 2019. The Education University of Hong Kong. (pages 78, 79, 85, 220).

[19] Alan Cha, Erik Wittern, Guillaume Baudart, James Davis, Louis Mandel, and Jim Laredo. A principled approach to GraphQL query cost analysis. In *ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 257–268, New York, 2020. Association for Computing Machinery. (pages 78, 79, 85, 221).

[20] Natchaporn Chaengmongkol, Somnuek Sinthupuan, Attawit Changkamanon, and Snit Sitti. Student Behavior Report Management System on Somsri.IO. In *2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 581–584, Chiang Mai, 2021. IEEE. (pages 78, 79, 223).

[21] B Chase Richard, Jacobs F Robert, and J Aquilano Nicholas. *Administración de operaciones-Producción y cadena de suministros*. Mc. Graw Hill, Mexico, 12 edition, 2009. (page 96).

[22] David Chaves-Fraga, Freddy Priyatna, Ahmad Alobaid, and Oscar Corcho. Exploiting Declarative Mapping Rules for Generating GraphQL Servers with Morph-GraphQL. *International Journal of Software Engineering and Knowledge Engineering*, 30(6):785–803, 2020. (pages 79, 221).

[23] Clarivate. Journal Citation Reports™: Reference Guide, 2021. (page 68).

[24] Clarivate. Quartiles in JCR on the InCites Platform, 2021. (page 68).

[25] Commercetools. Limits — HTTP API — commercetools, 2021. (pages 14, 168).

[26] Nelly Condori-Fernandez, Maya Daneva, Klaas Sikkel, Roel Wieringa, Oscar Dieste, and Oscar Pastor. A systematic mapping study on empirical evaluation of software requirements specifications techniques. *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, 1:502–505, 2009. (page 70).

[27] Florian Daniel, Maristella Matera, Elisa Quintarelli, Letizia Tanca, and Vittorio Zaccaria. Context-aware access to heterogeneous resources through on-the-fly mashups. In Reijers H.A. Krogstie J., editor, *30th International Conference on Advanced Information Systems Engineering, CAiSE 2018*, volume 10816 LNCS, pages 119–134, Tallinn, 2018. Springer Verlag. (pages 78, 219).

[28] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, 150:77–97, 2019. (page 4).

[29] Tomás Díaz, Federico Olmedo, and Éric Tanter. A Mechanized Formalization of GraphQL. In *CPP 2020 - Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, co-located with POPL 2020*, pages 201–214, New Orleans, 2020. Association for Computing Machinery, Inc. (pages 78, 79, 220).

[30] África Domingo, Jorge Echeverría, Óscar Pastor, and Carlos Cetina. Evaluating the influence of scope on feature location. *Information and Software Technology*, 2021. (page 108).

[31] James Dustyn and Kevin Chen-Chuan Chang. *Lion: listen online. Using GraphQL as a mediator for data integration and ingestion*. PhD thesis, University of Illinois at Urbana-Champaign, 2018. (pages 79, 220).

[32] Tore Dyba, Torgeir Dingsoyr, and Geir Hanssen. Applying Systematic Reviews to Diverse Study Types: An Experience Report. In *First International Symposium on Empirical Software Engineering and Measurement*, pages 225–234, Washington, 2007. IEEE. (pages 8, 65).

[33] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting Empirical Methods for Software Engineering Research. *Guide to Advanced Empirical Software Engineering*, 1(2007934531):285–311, 2008. (pages 21, 69, 70).

[34] Naresh Eeda and Nazim Madhavji. *Rendering real-time dashboards using a GraphQL- based UI Architecture*. PhD thesis, Western University, 2018. (pages 79, 220).

[35] Thomas Eizinger, Löwenstein Bernhard, and Lukasz Juszczyk. *API Design in Distributed Systems: A Comparison between GraphQL and REST*. PhD thesis, University of Applied Sciences Technikum Wien, 2017. (pages 84, 219).

[36] Elsevier. FAQs - Journal Metrics - Scopus.com, 2017. (page 68).

[37] Robert Engel, Shashank Rajamoni, Bryant Chen, Heiko Ludwig, and Alexander Keller. ysla: reusable and configurable slas for large-scale sla management. In

2018 IEEE 4th international conference on collaboration and internet computing (CIC), pages 317–325. IEEE, 2018. (page 53).

[38] Pontus Erlandsson, Joakim Remes, Niclas Ståhl, and Yacine Atif. *Performance comparison: Between GraphQL, REST & SOAP*. PhD thesis, Iniversity of Skovde, 2020. (pages 79, 84, 222).

[39] Inc. Facebook. GraphQL — A query language for your API, 2016. (page 30).

[40] Carles Farré, Jovan Varga, and Robert Almar. GraphQL Schema Generation for Data-Intensive Web APIs. In *9th International Conference on Model and Data Engineering, MEDI 2019*, volume 11815 LNCS, pages 184–194, Toulouse, 2019. Springer. (pages 78, 79, 221).

[41] Mafalda Ferreira, Isabel Silva, and Sousa José. *Analysis of GraphQL performance: a case study*. PhD thesis, Instituto Superior de Engenharia do Porto, 2019. (pages 6, 79, 84, 220).

[42] Roy T Fielding and Richard N Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002. (page 28).

[43] Antonio Gamez-Diaz, Pablo Fernandez, and Antonio Ruiz-Cortes. An analysis of restful apis offerings in the industry. In *International Conference on Service-Oriented Computing*, pages 589–604. Springer, 2017. (pages 52, 53).

[44] Antonio Gamez-Diaz, Pablo Fernandez, and Antonio Ruiz-Cortes. Automating sla-driven api development with sla4oai. In *International Conference on Service-Oriented Computing*, pages 20–35. Springer, 2019. (page 53).

[45] Antonio Gamez-Diaz, Pablo Fernandez, and Antonio Ruiz-Cortés. Governify for APIs: SLA-driven ecosystem for API governance. In *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1120–1123, 2019. (page 53).

[46] Antonio Gamez-Diaz, Pablo Fernandez, Antonio Ruiz-Cortés, Pedro J. Molina, Nikhil Kolekar, Prithpal Bhogill, Madhurranjan Mohaan, and Francisco Méndez. The role of limitations and SLAs in the API industry. In *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1006–1014, New

York, aug 2019. Association for Computing Machinery, Inc. (pages 14, 52, 53, 55, 120, 126).

[47] José María García, Pablo Fernández, Carlos Pedrinaci, Manuel Resinas, Jorge Cardoso, and Antonio Ruiz-Cortés. Modeling service level agreements with linked usdl agreement. *IEEE Transactions on Services Computing*, 10(1):52–65, 2016. (page 53).

[48] Marcela Genero, Antonio Cruz-Lemus, and Mario Piattini. *Métodos de investigación en ingeniería del software*. Ra-Ma, Bogota, 2014. (page 20).

[49] Eyob Ghebremicael, Frank Leymann, and Kálmán Képes. *Transformation of REST API to GraphQL for OpenTOSCA*. PhD thesis, University of Stuttgart, Stuttgart, 2017. (pages 6, 79, 221).

[50] GII-GRIN-SCIE (GGS). The GII-GRIN-SCIE (GGS) Conference Rating - Search the GGS Rating 2018, 2018. (page 68).

[51] GitHub Inc. Resource limitations - GitHub Docs, 2021. (pages 14, 168, 192).

[52] GitHub Inc. Git Guide, 2022. (page 168).

[53] GitLab. GraphQL API Tutorial — Toptal, 2021. (pages 14, 168).

[54] Lars Gleim, Tim Holzheim, István Koren, and Stefan Decker. Automatic bootstrapping of GraphQL endpoints for RDF triple stores. In *CEUR Workshop Proceedings*, volume 2722, pages 119–134, Virtual Conference, 2020. CEUR-WS. (pages 77, 79, 222).

[55] Berke Gözneli, Florian Matthes, and Gloria Bondel. *Identification and Evaluation of a Process for Transitioning from REST APIs to GraphQL APIs in the Context of Microservices Architecture*. PhD thesis, Technical University of Munich, 2020. (pages 79, 84, 222).

[56] GRANDstack. neo4j-graphql.js User Guide — GRANDstack, 2021. (page 44).

[57] Ying Guo, Fang Deng, and Xiudong Yang. Design and Implementation of Real-Time Management System Architecture based on GraphQL. In *2018 2nd Annual International Conference on Cloud Technology and Communication Engineering, CTCE 2018*, volume 466, page 9, Nanjing, 12 2018. Institute of Physics Publishing. (pages 29, 78, 79, 84, 219).

[58] Mohammad Kazem Haki and Maia Wentland Forte. Proposal of a service oriented architecture governance model to serve as a practical framework for business-it alignment. In *4th International Conference on New Trends in Information Science and Service Science*, pages 410–417. IEEE, 2010. (page 7).

[59] Dominik Hanák and Martin Večeřa. *GraphQL as modern access to jBPM process engine*. PhD thesis, Masaryk University, Brno, 2019. (pages 79, 84, 85, 221).

[60] Olaf Hartig and Jan Hidders. Defining Schemas for Property Graphs by using the GraphQL Schema Definition Language. In *2nd ACM SIGMOD Joint International Workshop on Graph Data Management Experiences and Systems and Network Data Analytics, GRADES-NDA 2019, co-located with the ACM SIGMOD International Conference on Management of Data 2019*, pages 1–11, Amsterdam, 2019. Association for Computing Machinery. (pages 77, 79, 219).

[61] Olaf Hartig and Jorge Pérez. An initial analysis of facebook's GraphQL language. In *11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, AMW 2017*, volume 1912, page 10, Montevideo, 2017. CEUR-WS. (pages 77, 219).

[62] Olaf Hartig and Jorge Pérez. Semantics and Complexity of GraphQL. In *WWW2018*, pages 1155–1164, Lyon, 2018. IW3C2. (pages 78, 79, 85, 220).

[63] Dewi Ayu Hartina, Armin Lawi, and Leonard Enrico Panggabean. Performance Analysis of GraphQL and RESTful in SIM LP2M of the Hasanuddin University. In *The 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT) 2018*, pages 237–240, Indonesia, 2018. Institute of Electrical and Electronics Engineers Inc. (pages 78, 79, 84, 221).

[64] Huahai He and Ambuj K Singh. Graphs-at-a-time: Query Language and Access Methods for Graph Databases. In *2008 ACM SIGMOD International Conference on Management of Data 2008, SIGMOD'08*, pages 405–417, Vancouver, 2008. ACM Sigmod. (page 66).

[65] Arnar Helgason, Mikael Berndtsson, and Yacine Atif. *Performance analysis of Web Services: Comparison between RESTful & GraphQL web services*. PhD thesis, University of Skövde, 2017. (pages 79, 84, 220).

[66] Christian Hettlage, Lucian Botha, Nhlavutelo Macebele, Moses Mogotsi, Sifiso Myeza, Encarni Romero Colmenero, Rosalind Skelton, Petri Väisänen, and Lonwabo Zaula. Building a modern data archive with React, GraphQL, and friends.

In *SPIE Astronomical Telescopes and Instrumentation*, page 10, Conference online, 2020. SPIE Proceedings. (pages 78, 79, 222).

[67] Alan Hevner and Samir Chatterjee. Design Science Research in Information Systems. In *Integrated Series in Information Systems*, chapter 2, pages 9–22. Springer, Boston, 39 edition, 2010. (page 22).

[68] Jani Hietala, Riku Ala-Laurinaho, Juuso Autiosalo, and Heikki Laaki. GraphQL Interface for OPC UA. In *Proceedings - 2020 IEEE Conference on Industrial Cyberphysical Systems, ICPS 2020*, pages 149–155, Tampere, 2020. IEEE. (pages 6, 78, 79, 84, 222).

[69] Thomas Hunter II. *Advanced Microservices: A Hands-on Approach to Microservice Infrastructure and Tooling*. Apress, San Francisco, 2017. (page 4).

[70] Sam M. Ireland and Andrew C.R. Martin. Zincbindpredict—prediction of zinc binding sites in proteins. *Molecules*, 26(4):1–11, 2021. (pages 79, 223).

[71] Research Group ISA. SLA for Open API Initiative Specification (Research Version) — SLA4OAI-ResearchSpecification, 2021. (pages 14, 53, 120, 121, 122).

[72] ISO/IEC. *ISO/IEC 14598-1 Information technology — Software product evaluation — Part 1: General overview*, volume 1. International Organization for Standardization, Geneva, Switzerland, 1 edition, 1999. (pages 145, 146, 190, 197, 201).

[73] ISO/IEC. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Evaluation process*. International Organization for Standardization, 1 edition, 2011. (pages 40, 48).

[74] ISO/IEC. *ISO/IEC 25000:2014 "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE"*, volume 1. International Organization for Standardization, Geneva, Switzerland, 2 edition, 2014. (pages 95, 111).

[75] ISO/IEC. *NTE INEN-ISO/IEC 25010*. International Organization for Standardization, 1 edition, 2015. (pages 40, 45, 94, 95, 110, 111, 152).

[76] ISO/IEC. *ISO/IEC 25022:2016 "Systems and software engineering — Systems and software quality requirements and evaluation (SQuaRE) — Measurement of quality in use"*, volume 1. International Organization for Standardization, Geneva, Switzerland, 1 edition, 2016. (pages 40, 45, 48, 95, 96, 110, 111, 112, 149, 152, 159, 163).

[77] ISO/IEC. *ISO/IEC 25023:2016 "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality"*, volume 1. International Organization for Standardization, Geneva, Switzerland, 1 edition, 2016. (pages 95, 96, 152, 170, 186, 188, 195, 198).

[78] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. Reporting experiments in software engineering. In *Guide to advanced empirical software engineering*, pages 201–228. Springer, 2008. (pages 21, 97).

[79] Dong-cheol Jeon, Liuhaoyang, and Heejoung Hwang. Design of Hybrid Application Based on GraphQL for Efficient Query for PHR. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 381–383, Jeju Island, 10 2019. Institute of Electrical and Electronics Engineers Inc. (pages 78, 79, 221).

[80] Json.org. JSON, 2009. (page 31).

[81] Juan Mendoza. RPubs - Alfa de Cronbach - Psicometría con R, 2018. (page 47).

[82] Natalia Juristo and Ana M Moreno. *Basics of software engineering experimentation*. Springer Science & Business Media, 2013. (page 21).

[83] Evrim Itir Karac, Burak Turhan, and Natalia Juristo. A controlled experiment with novice developers on the impact of task description granularity on software quality in test-driven development. *IEEE Transactions on Software Engineering*, 2019. (pages 97, 108).

[84] Stefan Karlsson, Adnan Causevic, and Daniel Sundmark. Automatic Property-based Testing of GraphQL APIs. In *IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 1–10, Madrid, 2021. IEEE. (pages 78, 79, 222).

[85] Raees Khan and Adnan Noor Mian. Sustainable IoT sensing applications development through graphQL-based abstraction layer. *Electronics (Switzerland)*, 9(4):23, 4 2020. (pages 79, 84, 221).

[86] Yun Wan Kim, Mariano Consens, and Olaf Hartig. An Empirical Analysis of GraphQL API Schemas in Open Code Repositories and Package Registries. In *13th Alberto Mendelzon International Workshop on Foundations of Data Management, AMW 2019*, pages 1–5, Asuncion, 2019. CEUR-WS. (pages 77, 84, 85, 219).

[87] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004. (page 20).

[88] Barbara Kitchenham and Pearl Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049–2075, 2013. (pages 8, 41, 65, 67).

[89] Barbara Kitchenham and Pearl Brereton. A systematic review of systematic review process research in software engineering. *Information and software technology*, 55(12):2049–2075, 2013. (page 20).

[90] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering, 2007. (page 20).

[91] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering*, 28(8):721–734, 2002. (page 21).

[92] Antti Knutas, Arash Hajikhani, Juho Salminen, Jouni Ikonen, and Jari Porras. Cloud-based bibliometric analysis service for systematic mapping studies. *ACM International Conference Proceeding Series*, 1008:184–191, 2015. (page 41).

[93] Christian Kohl, Emma J. McIntosh, Stefan Unger, Neal R. Haddaway, Steffen Kecke, Joachim Schiemann, and Ralf Wilhelm. Online tools supporting the conduct and reporting of systematic reviews and systematic maps: A case study on CADIMA and review of existing tools. *Environmental Evidence*, 7(1):1–17, 2018. (pages 41, 42).

[94] Yousri Kouki, Frederico Alvares De Oliveira, Simon Dupont, and Thomas Ledoux. A language support for cloud elasticity management. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 206–215. IEEE, 2014. (page 53).

[95] Vadim Kozhevnikov and Donat Shergalis. Migrating from REST to GraphQL having long-term supported clients. *Theoretical And Applied Science*, 93(February):180–185, 2001. (pages 79, 84, 222).

[96] Marco Kuhrmann, Daniel Méndez Fernández, and Maya Daneva. On the pragmatic design of literature studies in software engineering: an experience-based

guideline. *Empirical Software Engineering*, 22(6):2852–2891, 2017. (pages 8, 63, 65, 72).

[97] Dominik Adam Kus, István Koren, and Ralf Klamma. A Link Generator for Increasing the Utility of OpenAPI-to-GraphQL Translations. In *WWW2020*, page 4, Taipei, 5 2020. Association for Computing Machinery (ACM) Digital Library. (pages 78, 79, 85, 220).

[98] Miguel A. Laguna and Yania Crespo. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming*, 78(8):1010–1034, 2013. (page 65).

[99] Binod Lama and Miguel Valero. *Implementing Graphql in Existing REST API*. PhD thesis, Universitat Politécnica de Catalunya, Barcelona, 2019. (pages 79, 221).

[100] Markus Larsson, David Ångström, Sijin Cheng, and Olaf Hartig. *A Performance Comparison of Auto-Generated GraphQL Server Implementations*. PhD thesis, Linköping University, 2020. (pages 79, 84, 221).

[101] Byron Lee. Introducing the GraphQL Foundation – Lee Byron – Medium, 2018. (pages 6, 8).

[102] Eunggi Lee, Kiwoong Kwon, and Jungmee Yun. Performance Measurement of GraphQL API in Home ESS Data Server. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, volume 2020-Octob, pages 1929–1931, Jeju, 2020. IEEE. (pages 78, 79, 84, 222).

[103] Shanshan Li, He Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, Jidong Ge, and Zhihao Shan. A dataflow-driven approach to identifying microservices from monolithic applications. *Journal of Systems and Software*, 157:16, 2019. (page 4).

[104] Lu Liang, Chunfeng Ma, Tengfei Du, Yufei Zhao, Xiaoyong Zhao, Mengmeng Liu, Zhonghua Wang, and Jianping Lin. Bioactivity-explorer: a web application for interactive visualization and exploration of bioactivity data. *Journal of Cheminformatics*, 11(1):1–6, 2019. (pages 79, 219).

[105] Chi Y. Lin, Tarek Abdel-Hamid, and Joseph S. Sherif. Software-Engineering Process Simulation model (SEPS). *Journal of Systems and Software*, 38(3):263–277, 1997. (page 70).

[106] Jonas Lind, Kieron Soames, Patrick Lambrix, and Olaf Hartig. *Detecting Cycles in GraphQL Schemas*. PhD thesis, Linköping University, 2018. (pages 79, 219).

[107] Mhairi McNeill. About RStudio - RStudio, 2020. (page 46).

[108] Peter M Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards and Technology, Gaithersburg, 2011. (page 4).

[109] Daniela Meneses, Alison Fernandez, Andreina Cota, Juan Sandoval, Milton Mamani, Alexandre Bergel, and Stephane Ducasse. Deviation Testing: A Test Case Generation Technique for GraphQL APIs. In *11th International Workshop on Smalltalk Technologies (IWST)*, page 9, New York, 2018. Association for Computing Machinery. (pages 77, 79, 85, 221).

[110] Suresh Mukhiya, Fazle Rabbi, Violet Pun, Adrian Rutle, and Yngve Lamo. A graphql approach to healthcare information exchange with hl7 fhir. In *The 9th International Conference on Current and Future Trends of Information and The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH 2019) Communication Technologies in Healthcare*, volume 160, pages 338–345, Coimbra, 2019. Elsevier B.V. (pages 5, 7, 78, 79, 84, 220).

[111] Neo4j. Cypher Query Language - Developer Guides, 2020. (page 44).

[112] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. An analysis of public rest web service apis. *IEEE Transactions on Services Computing*, 2018. (pages 52, 53).

[113] Falco Nogatz and Dietmar Seipel. Implementing GraphQL as a query language for deductive databases in SWI-Prolog using DCGs, quasi quotations, and dicts. In *30th Workshop on (Constraint) Logic Programming, WLP 2016 and 29th Workshop on (Constraint) Logic Programming, WLP 2015*, volume 234, pages 42–56, Leipzig, 2017. Open Publishing Association. (pages 29, 77, 84, 220).

[114] Camille Oggier. *How fast GraphQL is compared to REST APIs*. PhD thesis, Haaga-Helia University of Applied Sciences, Helsinki, 2020. (pages 79, 84, 221).

[115] Olin Emily and The Linux Foundation. The Linux Foundation Announces Intent to Form New Foundation to Support GraphQL - The Linux Foundation, 2018. (pages 6, 8).

[116] Arsenio Paez. Grey literature: An important resource in systematic reviews. *Journal of Evidence-Based Medicine*, 10(May):1–8, 2017. (pages 66, 67).

[117] José Carlos Paiva, Alicja Haraszczuk, Ricardo Queirós, José Paulo Leal, Jakub Swacha, and Sokol Kosta. FGPE Gamification Service: A GraphQL Service to Gamify Online Education. In *World Conference on Information Systems and Technologies WorldCIST 2021*, volume AISC 1368, pages 480–489, Terceira Island, 2021. Springer, Cham. (pages 78, 79, 222).

[118] Cesare Pautasso. Restful web services: principles, patterns, emerging technologies. In *Web Services Foundations*, pages 31–51. Springer, 2014. (page 28).

[119] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of management information systems*, 24(3):45–77, 2007. (pages 13, 23, 39, 166).

[120] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008*, page 11, Bari, 2008. BCS Learning and Development Ltd. (pages 20, 63).

[121] Kai Petersen and Cigdem Gencel. Worldviews, research methods, and their relationship to validity in empirical software engineering research. In *Proceedings - Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement, IWSM-MENSURA 2013*, pages 81–89, Ankara, 2013. IEEE Computer Society. (pages 72, 73).

[122] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015. (pages 20, 42, 63, 65, 67, 68, 69, 72, 73, 80).

[123] Mark Petticrew and Helen Roberts. *Systematic reviews in the social sciences: A practical guide*. John Wiley & Sons, 2008. (page 20).

[124] Prisma. Execution — GraphQL, 2017. (pages 37, 39).

[125] Prisma. GraphQL vs REST - A comparison, 2017. (page 5).

[126] Freddy Priyatna, David Chaves-Fraga, Ahmad Alobaid, and Oscar Corcho. Morph-GraphQL: GraphQL servers generation from R2RML mappings (SESE). In *31st International Conference on Software Engineering and Knowledge Engineering, SEKE 2019.*, volume 2019-July, pages 291–296, Lisbon, 2019. Knowledge Systems Institute Graduate School. (pages 78, 79, 220).

[127] Ossi Puustinen. *GraphQL for building microservices*. PhD thesis, Tampere University, 2020. (pages 79, 222).

[128] Antonio Quiña-Mera, Pablo Fernandez, José María García, and Antonio Ruiz-Cortés. Efectos de las arquitecturas REST y GraphQL: Paquete de laboratorio, apr 2022. (page 225).

[129] Antonio Quiña-Mera, Pablo Fernández, José María García, Edwin Bastidas, and Antonio Ruiz-Cortés. Quality in use evaluation of a GraphQL implementation. In *Proceedings - XVI Multidisciplinary International Congress on Science and Technology, CIT 2021*, page 0, 6330 Cham, Switzerland, 2021. Springer Nature Switzerland AG. (pages 40, 212).

[130] Meuthia Rachmaniah, Maya Maharani Krismanti, and Muhammad Idzhar Darissalam. Tokocabai marketplace application based on web using extreme programming method. In *2020 International Conference on Computer Science and Its Application in Agriculture, ICOSICA 2020*, pages 1–7, Bogor, 2020. IEEE. (pages 78, 79, 223).

[131] Damaraju Raghavarao and Lakshmi Padgett. *Repeated measurements and cross-over designs*. John Wiley & Sons, 2014. (page 97).

[132] Saqib Rasool, Raees Khan, and Adnan Noor Mian. GraphQL and DC-WSN-Based Cloud of Things. *IT Professional*, 21(1):59–66, 1 2019. (pages 79, 219).

[133] Muhammad Raza, Farookh Khadeer, Omar Khadeer, and Ming Zhao. A comparative analysis of machine learning models for quality pillar assessment of SaaS services by multi-class text classification of users ' reviews. *Future Generation Computer Systems*, 101:341–371, 2019. (page 4).

[134] Leonard Richardson, Mike Amundsen, Michael Amundsen, and Sam Ruby. *RESTful Web APIs: Services for a Changing World*. " O'Reilly Media, Inc.", 2013. (page 28).

[135] Aleksi Ritsilä. *GraphQL: The API Design Revolution*. PhD thesis, Haaga-Helia University of Applied Sciences, Helsinki, 2017. (pages 79, 84, 221).

[136] Roberto Rodriguez-Echeverria, Javier Cánovas, and Jordi Cabot. Towards a UML and IFML Mapping to GraphQL. In Wimmer M. Garrigos I., editor, *17th International Conference on Web Engineering, ICWE 2017*, volume 10544 LNCS, pages 149–155, Rome, 2018. Springer Verlag. (pages 78, 79, 220).

[137] Piotr Roksela, Marek Konieczny, and Slawomir Zielinski. Evaluating execution strategies of GraphQL queries. In N. Herencsar, editor, *2020 43rd International Conference on Telecommunications and Signal Processing, TSP 2020*, pages 640–644, Milan, 2020. Institute of Electrical and Electronics Engineers Inc. (pages 78, 79, 221).

[138] Yana Rose, Jose M. Duarte, Robert Lowe, Joan Segura, Chunxiao Bi, Charmi Bhikadiya, Li Chen, Alexander S. Rose, Sebastian Bittrich, Stephen K. Burley, and John D. Westbrook. RCSB Protein Data Bank: Architectural Advances Towards Integrated Searching and Efficient Access to Macromolecular Structure Data from the PDB Archive. *Journal of Molecular Biology*, 433(11):166704, 2021. (pages 79, 223).

[139] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012. (pages 166, 167).

[140] Fatima Samea, Farooque Azam, Muhammad Rashid, Muhammad Waseem Anwar, Wasi Haider Butt, and Abdul Wahab Muzaffar. A model-driven framework for data-driven applications in serverless cloud computing. *PLoS ONE*, 15(8 August):1–32, 2020. (pages 79, 220).

[141] Jaime Sayago, Evelin Flores, and Andres Recalde. Comparative Analysis Between Standards Oriented to Web Services: SOAP, REST and GRAPHQL. In *International Conference on Applied Technologies*, volume 1193 CCIS, pages 286–300, Quito, 2019. Springer. (pages 78, 79, 84, 221).

[142] Jonas Schurmann, Tim Tegeler, and Bernhard Steffen. Guaranteeing Type Consistency in Collective Adaptive Systems. In *9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020*, pages 311–328, Rhodes, 2020. Springer Nature Switzerland. (pages 78, 79, 85, 222).

[143] SCImago. SJR - About Us, 2012. (page 68).

[144] SCImago. SJR - Help, 2019. (page 68).

[145] Pavel Seda, Pavel Masek, Jindriska Sedova, Milos Seda, Jan Krejci, and Jiri Hosek. Efficient Architecture Design for Software as a Service in Cloud Environments. In *10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops, ICUMT 2018*, pages 317–322, Moscow, 11 2018. IEEE Computer Society. (pages 78, 79, 219).

[146] Philipp Seifer, Johannes Härtel, Martin Leinberger, Ralf Lämmel, and Steffen Staab. Empirical study on the usage of graph query languages in open source Java projects. In *SLE 2019 - Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering, co-located with SPLASH 2019*, pages 152–166, Athens, 10 2019. Association for Computing Machinery, Inc. (pages 6, 78, 79, 84, 85, 221).

[147] Klaas Sijtsma. On the use, the misuse, and the very limited usefulness of cronbach's alpha. *Psychometrika*, 74(1):107–120, 2009. (page 47).

[148] Janice Singer. Using the american psychological association (apa) style guidelines to report experimental results. In *Workshop on Empirical Studies in Software Engineering (WSESE 1999)*, pages 71–75. Citeseer, 1999. (page 21).

[149] Akansha Singh and N. Jeyanthi. MVP Architecture Model with Single Endpoint Access for Displaying COVID 19 Patients Information Dynamically. In *Proceedings - 2020 12th International Conference on Computational Intelligence and Communication Networks, CICN 2020*, pages 471–476, Bhimtal, 2020. IEEE. (pages 78, 79, 84, 222).

[150] Dmitry Sklyarov, Ari Rantala, and Oy Movya. *The Web service development with React , GraphQL and Apollo*. PhD thesis, JAMK University of Applied Sciences, 2020. (pages 79, 223).

[151] Jeremias Snellman, Timo Mantere, Petri Välisuo, and Tim Wallin Gambi. *Implementation and Evaluation of a Graphql-Based Web Application for Project Follow Up*. PhD thesis, University of Vaasa, Vaasa, 2019. (pages 79, 85, 221).

[152] M. Solari and S. Vegas. Classifying and analysing replication packages for software engineering experimentation. In *7th International Conference on Product Focused Software Process Improvement (PROFES 2006)-Workshop Series in Empirical Software Engineering (WSESE)*, 2006. (page 171).

[153] Facebook Open Source. Schemas and Types — GraphQL, 2015. (pages 34, 36, 37).

[154] Vlatko Spasev, Ivica Dimitrovski, and Ivan Kitanovski. An Overview of GraphQL : Core Features and Architecture. In *ICT Innovations Conference 2020*, page 14, Skopje, 2020. Springer. (pages 78, 79, 222).

[155] Karin Staring, Stelios Vitalis, Linda Brink, and Balázs Dukai. *Combination of CityJSON with PostgreSQL, MongoDB and GraphQL*. PhD thesis, Delft University ofTechnology, 2020. (pages 79, 84, 222).

[156] Patrick Stünkel, Ole von Bargen, Adrian Rutle, and Yngve Lamo. GraphQL Federation: A Model-Based Approach. *The Journal of Object Technology*, 19(2):21, 2020. (pages 79, 84, 221).

[157] Mahsud M. Sultanov, Yulia A. Gorban, Aleksey A. Smirnov, and Viktor A. Yurov. Development of a centralized system for data storage and processing on operation modes and reliability indicators of power equipment. In *Proceedings of the 3rd 2021 International Youth Conference on Radio Electronics, Electrical and Power Engineering, REEPE 2021*, pages 3–7, Moscow, 2021. IEEE. (pages 78, 79, 222).

[158] Gede Susrama, Mas Diyasa, Gideon Setya Budiwitjaksono, Hafidz Amarul, and Ilham Ade. Comparative Analysis of Rest and GraphQL Technology on Nodejs-Based Api Development. In *5th International Seminar of Research Month 2020*, volume 2021, pages 43–52, Zoom/Live Streaming Youtube, 2021. Nusantara Science and Technology Proceedings. (pages 78, 79, 84, 222).

[159] Ruben Taelman, Miel Vander Sande, and Ruben Verborgh. Bridges between GraphQL and RDF. In *W3C Workshop on Web Standardization for Graph Data. W3C*, pages 4–7, Berlin, 2019. W3C. (pages 77, 219).

[160] Teemu Taskula, Eero Hyvonen, Janne Kario, and Keski-Luopa Jukka. *Advanced Data Fetching with GraphQL: Case Bakery Service*. PhD thesis, Aalto University, 2019. (pages 79, 84, 219).

[161] Samir Tata, Mohamed Mohamed, Takashi Sakairi, Nagapramod Mandagere, Obinna Anya, and Heiko Ludwig. rsla: A service level agreement language for cloud services. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 415–422. IEEE, 2016. (page 53).

[162] The GraphQL Foundation. GraphQL, 2018. (pages 28, 29, 30, 32, 35, 37, 38, 39).

[163] Johannes Thönes. Microservices. *IEEE Software*, 32:4, 2015. (page 4).

[164] István Tóth-Király, Gábor Orosz, Edina Dombi, Balázs Jagodics, Dávid Farkas, and Camille Amoura. Cross-cultural comparative examination of the Academic Motivation Scale using exploratory structural equation modeling. *Personality and Individual Differences*, 106:130–135, 2017. (page 47).

[165] Ville Touronen, Jussi Kangasharju, and Matti Luukkainen. *Microservice architecture patterns with GraphQL*. PhD thesis, University of Helsinki, Helsinki, 2019. (pages 79, 221).

[166] Tri Tran and Janne Salonen. *Build a GraphQL application with Node. js and React*. PhD thesis, Metropolia University of Applied Sciences, 2019. (pages 79, 220).

[167] Wei Tek Tsai, Xiao Ying Bai, and Yu Huang. Software-as-a-service (SaaS): Perspectives and challenges. *Science China Information Sciences*, 57(5):1–15, 2014. (page 4).

[168] H Ulrich, J Kern, D. Tas, A. K. Kock-Schoppenhauer, F. Ückert, J. Ingenerf, and M. Lablans. QL 4 MDR: A GraphQL query language for ISO 11179-based metadata repositories. *BMC Medical Informatics and Decision Making*, 19(1):7, 2019. (pages 79, 220).

[169] Rafael Brundo Uriarte, Francesco Tiezzi, and Rocco De Nicola. Slac: A formal service-level-agreement language for cloud computing. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 419–426. IEEE, 2014. (page 53).

[170] R. Varadan, K. Channabasavaiah, S. Simpson, K. Holley, and A. Allam. Increasing business flexibility and soa adoption through effective soa governance. *IBM Systems Journal*, 47(3):473–488, 2008. (page 7).

[171] Andrea Vazquez-Ingelmo, Juan Cruz-Benito, and Francisco García-Penalvo. Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL. In *5th International Conference on Technological Ecosystem for Enhancing Multiculturality, TEEM 2017*, volume Part F1322, page 8, Cadiz, 10 2017. Association for Computing Machinery. (pages 78, 79, 84, 220).

[172] Mario G Piattini Velthuis. *Auditoría de Tecnologías y Sistemas de Información*. Grupo Editorial RA-MA, 2008. (page 52).

[173] Milena Vesić and Nenad Kojić. Comparative Analysis of Web Application Performance In Case of Using REST Versus GraphQL. In *Fourth International Scientific Conference ITEMA 2020*, pages 1–9, ONLINE-Virtual, 2020. (pages 29, 78, 79, 84, 222).

[174] Maximilian Vogel, Sebastian Weber, and Christian Zirpins. Experiences on migrating RESTful Web Services to GraphQL. In *ICSOC Workshops 2017*, page 283–295, Malaga, 2018. Springer Verlag. (pages 5, 6, 7, 78, 79, 84, 219).

[175] R. H. Von Alan, S. T March, J. Park, and S. Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004. (pages 11, 39).

[176] Kai Von-Rönne and Dirk Riehle. *GraphQL-based generic and domain specific query interfaces for the JValue ODS*. PhD thesis, Friedrich-Alexander University Erlangen-Nürnberg, 2020. (pages 79, 222).

[177] Chen Wang, Luigi Marini, Chieh Li Chin, Nickolas Vance, Curtis Donelson, Pascal Meunier, and Joseph T. Yun. Social media intelligence and learning environment: An open source framework for social media data Collection, Analysis and Curation. In *IEEE 15th International Conference on eScience, eScience 2019*, pages 252–261, San Diego, 9 2019. IEEE Computer Society. (pages 78, 79, 221).

[178] Shaohua Wang, Iman Keivanloo, and Ying Zou. How do developers react to RESTful API evolution? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8831:245–259, 2014. (page 4).

[179] Jeroen Maurits Werbrouck, Madhumitha Senthilvel, Jakob Beetz, and Pieter Pauwels. Querying heterogeneous linked building data with context-expanded GraphQL queries? In *CEUR Workshop Proceedings*, volume 2389, pages 21–34, Lisbon, 2019. CEUR-WS. (pages 77, 79, 84, 220).

[180] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requirements Engineering*, 11(1):102–107, 2006. (pages 68, 69).

[181] Daniel Wikander, Johan Holmberg, and Mia Persson. *Exploring the quality attribute and performance implications of using GraphQL in a data-fetching API*. PhD thesis, Malmö University, Malmö, 2020. (pages 79, 84, 221).

[182] Erik Wittern, Alan Cha, James Davis, Guillaume Baudart, and Louis Mandel. An Empirical Study of GraphQL Schemas. In *17th International Conference on Service-Oriented Computing, ICSOC 2019*, pages 3–19, Toulouse, 2019. Springer. (pages 78, 79, 84, 85, 220).

[183] Erik Wittern, Alan Cha, and Jim A. Laredo. Generating GraphQL-wrappers for REST(-like) APIs. In *18th International Conference on Web Engineering, ICWE 2018*, volume 10845 LNCS, pages 65–83, Caceres, 2018. Springer Verlag. (pages 6, 7, 78, 79, 85, 219).

[184] Claes Wohlin. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *18th International Conference on Evaluation and Assessment in Software Engineering*, page 10, London, 2014. Association for Computing Machinery. (page 65).

[185] Claes Wohlin, Per Runeson, Paulo Anselmo Da Mota Silveira Neto, Emelie Engström, Ivan Do Carmo Machado, and Eduardo Santana De Almeida. On the reliability of mapping studies in software engineering. *Journal of Systems and Software*, 86(10):2594–2610, 2013. (page 68).

[186] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 1 edition, 2012. (pages 11, 21, 69, 70, 118, 150, 161).

[187] M. Zelkowitz and D. Wallace. Experimental Models for Validating Technology. *Computer*, 31(5):23–31, 1998. (page 97).

[188] Olaf Zimmermann. Microservices tenets: Agile approach to service development and deployment. *Computer Science - Research and Development*, 32(3-4):301–310, 2017. (pages 4, 7).