UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Ingeniería en Sistemas Computacionales

COMPARATIVA DE LA EFICIENCIA DEL RENDIMIENTO DE LAS TECNOLOGÍAS REST Y GRAPHQL MEDIANTE EL CONSUMO DEL API-GITHUB EN LAS VERSIONES 3 Y 4

Trabajo de grado presentado ante la ilustre Universidad Técnica del Norte previo a la obtención del título de Ingeniero en Sistemas Computacionales

Autor:

Jimmy Alejandro Ibarra Bolaños

Director:

PhD. José Antonio Quiña Mera

Ibarra - Ecuador

2024

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE INDENTIDAD:	040163246-8		
APELLIDOS Y NOMBRES:	IBARRA BOLAÑOS JIMMY ALEJANDRO		
DIRECCIÓN:	EL ÁNGEL - CARCHI		
EMAIL:	jaibarrab@utn.edu.ec		
TELÉFONO FIJO:	062975-208	**TELÉFONO MÓVIL:**	0980291947

DATOS DE LA OBRA	
TÍTULO:	COMPARATIVA DE LA EFICIENCIA DEL RENDIMIENTO DE LAS TECNOLOGÍAS REST Y GRAPHQL MEDIANTE EL CONSUMO DEL API-GITHUB EN LAS VERSIONES 3 Y 4
AUTOR(ES):	IBARRA BOLAÑOS JIMMY ALEJANDRO
FECHA:	08/01/2020
PROGRAMA:	PREGRADO
TÍTULO POR EL QUE OPTA:	INGENIERO EN SISTEMAS COMPUTACIONALES
DIRECTOR:	PHD. ANTONIO QUIÑA
ASESOR 1:	MSC. CARPIO PINEDA
ASESOR2:	MSC. FERNANDO GARRIDO

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de esta y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 05 días del mes de septiembre de 2024

EL AUTOR:

ESTUDIANTE
Jimmy Alejandro Ibarra Bolaños
C.I: 0401632468

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN DIRECTOR

Por medio del presente, yo PhD. Antonio Quiña, certifico que el Sr. **JIMMY ALEJANDRO IBARRA BOLAÑOS**, portador de la cédula de ciudadanía Nro. 0401632468, ha trabajado en el desarrollo del proyecto de grado: **"Comparativa de la eficiencia del rendimiento de las tecnologías REST y GraphQL mediante el consumo del API-GitHub en las versiones 3 y 4"**, previo a la obtención del título de Ingeniero en Sistemas Computacionales realizado con interés profesional y responsabilidad que certifico con honor de verdad.

Atentamente

1002322384
JOSE ANTONIO
QUIÑA MERA

Firmado digitalmente por 1002322384 JOSE ANTONIO QUIÑA MERA
Fecha: 2024.09.05 14:03:39 -05'00'

PhD. Antonio Quiña

DIRECTOR DE TRABAJO DE GRADO

DEDICATORIA

A mi querida madre.

Nelly Elizabeth Bolaños Chulde

Ejemplo de esfuerzo, constancia y responsabilidad.

"Mi madre era la mujer más hermosa que he visto. Todo lo que soy se lo debo a mi madre. Atribuyo todos mis éxitos en la vida a la educación moral, intelectual y física que recibí de ella".

George Washington

Jimmy Alejandro Ibarra Bolaños

AGRADECIMIENTO

A

- Dios, por haberme concedido el don de la vida. Por guiar e iluminar mi camino. Y por ser mi fuerza para encarar el día a día.

- La Universidad Técnica del Norte y a mis profesores. Por permitirme ser parte de tan prestigiosa Casona Universitaria, por compartir sus conocimientos y su experiencia conmigo, por darme lo necesario para ser un gran profesional.

- Mi tutor, PhD. Antonio Quiña. Por haberme brindado su ayuda de manera oportuna durante la realización del presente trabajo de titulación. De igual manera a mis asesores MSc. Carpio Pineda y MSc. Fernando Garrido, por sus recomendaciones y sugerencias que han sido de gran valía.

- Mi padre, Jaime Ibarra. Por apoyarme en mis decisiones, por ayudarme cuando lo necesité y por darme sus consejos de gran valía.

- Mi madre, Nelly Bolaños. Por haber estado conmigo incondicionalmente, por haberme formado en valores para ser una persona de bien, por haberse privado de muchas comodidades para dármelas a mí, por su trabajo diario, por su infinito amor.

- Mis hermanas, Paulina y Nayeli. Por ayudarme a ser una mejor persona, por motivarme para seguir adelante siempre, por darme su cariño, por todo lo vivido.

- Mis abuelitos, Blanca Chulde y Guillermo Bolaños. Por brindarme sus consejos y sus enseñanzas, por apoyarme firmemente para cumplir cada una de mis metas.

Jimmy Alejandro Ibarra Bolaños

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

REST es un estilo arquitectónico para el diseño de servicios web que fue introducido por Roy Fielding en el año 2000, quien buscaba un enfoque más eficiente y escalable para la comunicación entre sistemas distribuidos en la web. REST promueve la separación entre el cliente y el servidor y se basa en la utilización de métodos HTTP estándar, la manipulación de recursos a través de sus URIs y la representación de estos recursos en formatos como JSON o XML.

GraphQL es un lenguaje de consulta para APIs y un tiempo de ejecución para ejecutar esas consultas con datos existentes. Fue desarrollado por Facebook en 2012, surgió como respuesta a las limitaciones de REST. GraphQL incluye conceptos avanzados como el esquema de tipos, que define la estructura de los datos disponibles y permite a los clientes y servidores evolucionar de manera independiente sin romper la compatibilidad. A diferencia de REST, donde cada recurso tiene un endpoint fijo y las respuestas pueden incluir datos no deseados, GraphQL permite a los clientes especificar exactamente qué datos necesitan en una única petición.

La mayoría de las API web actuales siguen el estilo arquitectónico REST, lo que ha contribuido a su popularidad y longevidad. Por su parte, GraphQL, desde su lanzamiento, ha sido adoptado por numerosas empresas y proyectos de código abierto, consolidándose como una alternativa robusta y eficiente a REST para el diseño de APIs. En este sentido, la idea principal del presente trabajo de titulación es comparar el rendimiento de las tecnologías REST y GraphQL, a través de un experimento controlado, con el objetivo de dar a conocer cuál de ellas es la más eficiente, teniendo en cuenta la métrica denominada tiempo medio de respuesta provista por la norma ISO/IEC 25023. Esta comparativa se la realiza exclusivamente en el contexto del consumo de datos, para ello se emplea las dos APIs de GITHUB, siendo estas la versión 3, desarrollada con arquitectura REST y la versión 4, desarrollada con arquitectura GraphQL.

Luego de haber desarrollado la comparativa de las dos arquitecturas en cuestión, se obtiene que GraphQL es 4.49 veces más eficiente en términos de tiempo de respuesta que REST. Sin embargo, en un caso de uso de los que aquí se desarrolló, REST resultó ser 1.52 veces más eficiente que GraphQL, lo cual sugiere que podría ser mejor utilizar REST en lugar de GraphQL en ciertos escenarios.

ABSTRACT

REST is an architectural style for designing web services that was introduced by Roy Fielding in 2000, who was looking for a more efficient and scalable approach to communication between distributed systems on the web. REST promotes the separation between the client and the server and is based on the use of standard HTTP methods, the manipulation of resources through their URIs and the representation of these resources in formats such as JSON or XML.

GraphQL is a query language for APIs and a runtime for executing those queries against existing data. It was developed by Facebook in 2012, it emerged as a response to the limitations of REST. GraphQL includes advanced concepts such as the type schema, which defines the structure of the available data and allows clients and servers to evolve independently without breaking compatibility. Unlike REST, where each resource has a fixed endpoint and responses can include unwanted data, GraphQL allows clients to specify exactly what data they need in a single request.

Most web APIs today follow the REST architectural style, which has contributed to their popularity and longevity. For its part, GraphQL, since its launch, has been adopted by numerous companies and open-source projects, consolidating itself as a robust and efficient alternative to REST for the design of APIs. In this sense, the main idea of this degree work is to compare the performance of REST and GraphQL technologies, through a controlled experiment, with the aim of revealing which of them is the most efficient, taking into account the metric called average response time provided by the ISO/IEC 25023 standard. This comparison is carried out exclusively in the context of data consumption, for this the two GITHUB APIs are used, these being version 3, developed with REST architecture and the version 4, developed with GraphQL architecture.

After having developed the comparison of the two architectures in question, it is obtained that GraphQL is 4.49 times more efficient in terms of response time than REST. However, in one use case developed here, REST was found to be 1.52 times more efficient than GraphQL, suggesting that it might be better to use REST instead of GraphQL in certain scenarios.

INTRODUCCIÓN

PROBLEMA

ANTECEDENTES

La arquitectura de software REST fue propuesta por primera vez en el año 2000 por el Dr. Roy Thomas Fielding. REST es la abreviatura de Representational State Transfer y es un estilo arquitectónico para sistemas hipermedia distribuidos; REST es un conjunto de conceptos de diseño que se basan en las características y requerimientos del software de red, cuyo objetivo es reducir la complejidad del desarrollo, mejorar la escalabilidad de los sistemas y reducir la carga en la comunicación (Guo et al., 2018).

En 2015, la compañía Facebook publica la especificación del lenguaje GraphQL de código abierto. GraphQL es un lenguaje de consulta especialmente adecuado para modelos de datos altamente vinculados (Ulrich et al., 2019).

SITUACIÓN ACTUAL

La arquitectura REST ha llegado a ser considerablemente exitosa y debido a que es simple y fácil de usar, ha sido bastante empleada por los proveedores de servicios y la mayoría de la comunidad de desarrolladores de software. Sin embargo, REST presenta algunos problemas, particularmente en sistemas con altos requerimientos de rendimiento en tiempo real, la simplicidad de REST no sólo conlleva algunas limitaciones, sino que también reduce el desempeño en un entorno de red deficiente (Guo et al., 2018).

Por su parte GraphQL, en comparación con REST, minimiza la cantidad de transferencias de datos innecesarias y consultas individuales, reduce también la posibilidad de errores causados por consultas no válidas en el cliente y soporta modelos de datos cambiantes sin necesidad de versionar el API (Guo et al., 2018).

PROSPECTIVA

Con el desarrollo del presente estudio comparativo se pretende dar a conocer los indicadores de eficiencia de las tecnologías REST y GraphQL. De esta manera se aportará con una base teórica que permitirá decidir cuál de las dos tecnologías es mejor utilizar en el desarrollo de aplicaciones informáticas.

PLANTEAMIENTO DEL PROBLEMA

Tecnologías emergentes como GraphQL sugieren ser mejores que REST, sin embargo, en nuestro medio no existe un estudio que determine cuál de estas dos tecnologías es mejor emplear en ciertos escenarios, por ejemplo, en su capacidad de consulta de datos.

En la Figura 1, se presenta un diagrama que ilustra el contexto previamente mencionado, junto con sus causas y efectos.

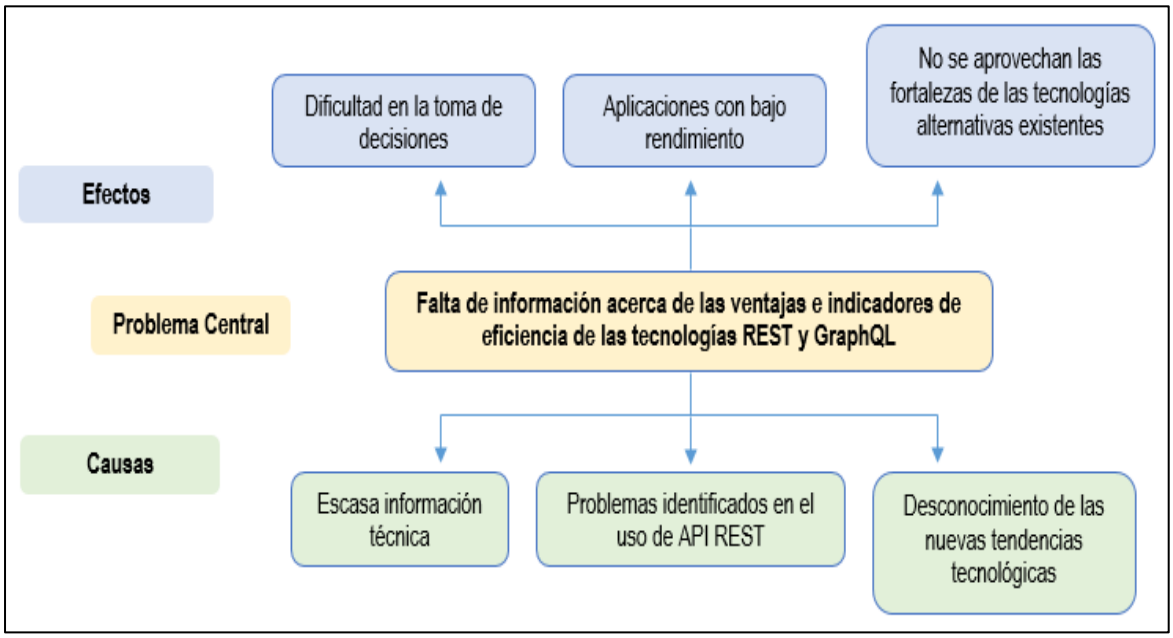


Figura 1. Árbol de Problemas

Por lo tanto, se define el problema como: ¿Cuál de las dos tecnologías sería más eficiente para el desarrollo de aplicaciones informáticas?

OBJETIVOS

OBJETIVO GENERAL

Comparar la eficiencia del rendimiento de las tecnologías REST y GraphQL mediante el consumo del API-GITHUB en las versiones 3 y 4.

OBJETIVOS ESPECÍFICOS

- Establecer una línea base conceptual para identificar las métricas de la norma ISO/IEC 25023 para la comparativa de las versiones 3 y 4 del API-GITHUB.
- Desarrollar una prueba de concepto para consumir las versiones 3 y 4 del API-GITHUB utilizando la metodología SCRUM.

- Verificar los resultados de la comparativa según las métricas de la norma ISO/IEC 25023.

ALCANCE

En este trabajo de investigación se realizará una comparativa de la eficiencia del rendimiento, basada en la norma ISO/IEC 25023, de las tecnologías REST y GraphQL. El estudio comparativo consistirá de los siguientes aspectos:

- Establecer las métricas de eficiencia según la norma ISO/IEC 25023.
- Desarrollar una prueba de concepto que consistirá en hacer una aplicación para consumir el API-GITHUB versión 3 y versión 4 (dichas versiones están implementadas con la tecnología REST y GraphQL respectivamente) y a través de casos de uso probar la funcionalidad de las dos APIs con el fin de obtener valores para las métricas de la norma ISO/IEC 25023.
- Realizar la comparativa de las tecnologías REST y GraphQL.
- Verificar los resultados en base a las métricas previamente establecidas.

Cabe recalcar que una prueba de concepto es una implementación, generalmente resumida o incompleta, de un método o idea y se realiza con la finalidad de verificar que el concepto o teoría en cuestión es susceptible de ser explotada de una manera útil; la prueba de concepto se considera normalmente un paso importante en el proceso de crear un prototipo realmente operativo (Fundación Sadosky, 2014).

JUSTIFICACIÓN

El presente proyecto está dentro del marco de los Objetivos de Desarrollo Sostenible planteados por la Organización de las Naciones Unidas:

9. b Apoyar el desarrollo de tecnologías, la investigación y la innovación nacionales en los países en desarrollo, incluso garantizando un entorno normativo propicio a la diversificación industrial y la adición de valor a los productos básicos, entre otras cosas (Naciones Unidas, 2019).

9. c Aumentar significativamente el acceso a la tecnología de la información y las comunicaciones y esforzarse por proporcionar acceso universal y asequible a Internet en los países menos adelantados de aquí a 2020 (Naciones Unidas, 2019).

JUSTIFICACIÓN SOCIAL

El presente trabajo de investigación será un aporte para las personas que están interesadas en las tecnologías REST y GraphQL y dará a conocer indicadores de eficiencia del desempeño de las mismas.

Comunidades científicas y tecnológicas como Api Gurú y similares se verán beneficiadas con esta investigación ya que será un instrumento que les permitirá tomar decisiones respecto a cuál tecnología utilizar para desarrollar sus proyectos de software.

JUSTIFICACIÓN ECONÓMICA

Mediante la comparativa es posible aportar a la comunidad tecnológica de manera que ellos puedan utilizar la herramienta más eficiente y puedan aprovechar sus recursos como tiempo, talento humano y costos.

JUSTIFICACIÓN TÉCNICA

Se llevará a cabo una prueba de concepto que efectuará la comparativa de las dos tecnologías en cuestión a través del consumo del API pública de GITHUB en las versiones 3 y 4, mismas que están implementadas con REST y GraphQL respectivamente.

Previa revisión en las bases bibliográficas Scopus, IEEE Explore, DBLP, Google Scholar se pudo comprobar que no existe un trabajo con un enfoque similar al que se propone, por lo cual se estaría haciendo un aporte inédito en la comunidad científica y tecnológica.

CONTEXTO

Luego de haber revisado los proyectos de titulación que se han realizado en la Universidad Técnica del Norte no se encontraron estudios similares donde se aborde las dos tecnologías que se propone en el estudio actual, pero sí existen trabajos en los cuales se utiliza servicios REST en la implementación de sus aplicativos. Además, se hizo una búsqueda en los repositorios digitales de varias otras universidades del país y se obtuvo un hallazgo, perteneciente a estudiantes de la Escuela Superior Politécnica de Chimborazo, donde se efectúa un estudio comparativo de los servicios web basados en REST y SOAP.

La Tabla 1, contiene mayores detalles y diferencias entre el proyecto que se propone en el presente documento y el antes mencionado.

Tabla 1. Trabajos de titulación relacionados

Fecha de publicación	Título	Autor(es)	Diferencias con el estudio actual
15-ene-2019	Estudio de integración de los frameworks angular 4 y YII2, orientado a servicios Rest, que permitan la gestión y control de inventarios para mejorar la productividad en la empresa induxion	Gudiño Quinteros, Andrés Rubén	Se realiza una implementación de servicios REST en el aplicativo, utilizando PHP como lenguaje de Backend. Aporte: en el estudio actual se realizará una comparativa entre REST y GraphQL y se utilizará, tentativamente, el framework Angular 8 para el desarrollo del aplicativo cliente.
ago-2018	Desarrollo de una aplicación web y móvil para la gestión de inventario y pedidos utilizando servicios RESTful.	Díaz Arrieta, Ronald Henry; Ortiz Vinueza, Jorge Luis	Se implementa servicios REST en el aplicativo. Aporte: en el estudio actual se darán a conocer métricas de rendimiento de las tecnologías REST y GraphQL.
14-mar-2016	Estudio Comparativo de los Servicios Web Restfull Jersey y SOAP JAX-WS para el Desarrollo de una Aplicación Android con Wikitude Aplicada a la Gestión de Información Geolocalizada del Turismo de la Provincia de Chimborazo.	Hidalgo Macas, Lorena Nataly.; Jiménez Acaro, Milton Edison.	La comparativa no se basa en normas ISO. SOAP es una tecnología antigua, por tanto, el estudio no es de actualidad. Aporte: el estudio actual tendrá base en la norma ISO/IEC 25023 y aborda temas de actualidad como lo son REST y GraphQL.

CAPÍTULO 1

Marco Teórico

1.1. Servicios Web

Según el World Wide Web Consortium (W3C), un servicio web es una aplicación software identificada a través de un URI cuyas interfaces se pueden definir, describir y descubrir mediante documentos XML; los servicios web hacen posible la interoperación de sistemas distribuidos heterogéneos con independencia de las plataformas hardware y software empleadas (Malla, 2014).

Se puede pensar en los servicios web como una arquitectura conceptual y tecnológica que permite que distintos servicios se describan, publiquen, descubran y utilicen a través de sistemas distribuidos, empleando la infraestructura de Internet; todo esto se sustenta en una gran cantidad de recomendaciones y estándares, donde XML forma el principal soporte de estas tecnologías (Malla, 2014).

En definitiva, los servicios web son un caso particular de arquitectura orientada a servicios (SOA). Estas arquitecturas orientadas a servicios en el fondo no son más que arquitecturas distribuidas, sin embargo, presentan algunas desventajas como que su rendimiento es generalmente bastante inferior al de otras arquitecturas distribuidas clásicas, y que necesitan varias tecnologías y especificaciones adicionales para funcionar de manera adecuada (Malla, 2014).

El hecho de que el desarrollo de estas tecnologías vaya ligado a procesos de estandarización es sumamente importante y alentador, pero no es suficiente sólo con disponer de estándares, sino que también es necesario disponer de herramientas que los implementen (Malla, 2014).

1.2. Arquitectura Orientada a Servicios (SOA)

SOA es una forma de tecnología arquitectónica que sigue el principio de orientación a servicios. Este concepto de orientación a servicios se enfoca en resolver grandes problemas al dividirlos en pequeños conjuntos de servicios para posteriormente tratar problemas específicos (Tee et al., 2019).

La arquitectura orientada a servicios (SOA) es uno de los estilos arquitectónicos más recientes cuyo objetivo es lograr un acoplamiento flexible entre los agentes de software que interactúan; SOA es un modelo de arquitectura de

sistemas basado en servicios web; un servicio es una unidad de trabajo creada por un proveedor de servicios; en SOA, los recursos se ponen a disposición de otros participantes en la red como servicios independientes a los que se accede de manera estandarizada (Cai et al., 2019).

SOA se enfoca en diseñar la arquitectura de una aplicación de manera que sea posible reutilizar componentes existentes. En SOA, el proceso interno está disponible como un servicio independiente al que se puede acceder sin conocimiento de la implementación de la plataforma tecnológica detrás de él (Aqsha et al., 2019).

1.3. API: Interfaz de programación de aplicaciones

API es un acrónimo en inglés que significa Application Programming Interface, traducido como Interfaz de Programación de Aplicaciones; este término se refiere a las funciones y métodos proporcionados por una biblioteca de programación específica, que actúan como una capa de abstracción; en otras palabras, permite que otro software pueda utilizar esas funciones y métodos (Quinaluiza, 2018).

La Interfaz de Programación de Aplicaciones (API) es un conjunto de funciones, métodos o procedimientos que han sido creados para ser consumidos o utilizados por otro software de forma segura; una API facilita la comunicación entre aplicaciones en formato estándar para el intercambio de datos; las APIs pueden habilitar el acceso a sus recursos a aplicaciones de terceros de forma controlada y segura (Lara, 2021).

En la Figura 2, se observa la interacción de una API con aplicaciones y servicios externos.

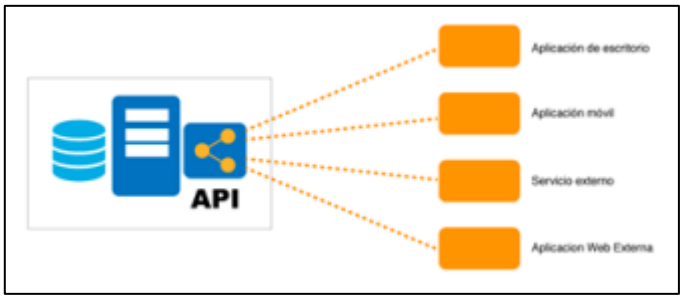


Figura 2. Esquema de una API
Fuente: (Lara, 2021)

1.4. REST: Estilo arquitectónico

Representational State Transfer (REST), es un estilo arquitectónico para el desarrollo de servicios web; en REST, el cliente envía una solicitud y el servidor

devuelve la respuesta al cliente mediante el uso de recursos; el lenguaje de REST se basa en sustantivos y verbos; los servicios web REST no dependen de ningún protocolo, pero casi todos los servicios REST hacen uso de verbos HTTP (get, post, put, delete) para regular las operaciones con los recursos; los recursos pueden ser videos, páginas web, imágenes, cualquier cosa que pueda permitirse en un sistema informático (Soni & Ranga, 2019).

Por tanto, REST es un método popular para proporcionar interoperabilidad entre un cliente y un servidor utilizando el protocolo de transferencia de hipertexto (HTTP) y un formato de intercambio común, por ejemplo, JavaScript Object Notation (JSON) (Tarkowska et al., 2018).

En la Figura 3, se observa la arquitectura de los servicios web basados en REST.

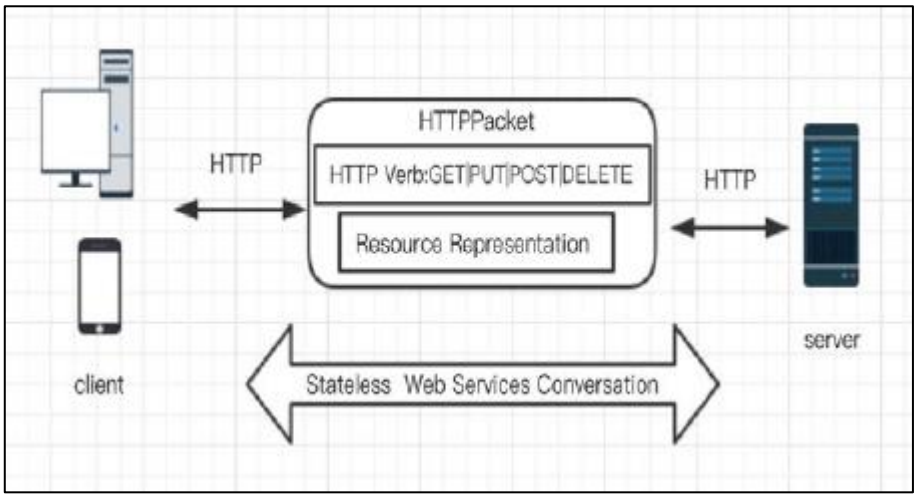


Figura 3. Arquitectura de servicios web REST
Fuente: (Guo et al., 2018)

1.5. GraphQL: Lenguaje de consultas de APIs

GraphQL es un lenguaje de consultas de la capa de aplicación desarrollado por Facebook. En octubre de 2015, esta compañía liberó GraphQL bajo la modalidad de código abierto y ha venido trabajando en él internamente desde 2012. En la actualidad, la mayoría de las aplicaciones de Facebook utilizan este lenguaje de consultas como mecanismo de obtención de datos, lo cual resulta en cientos de miles de millones de llamadas a sus APIs GraphQL al día (Nogatz & Seipel, 2017).

GraphQL ofrece una interfaz unificada entre el cliente y el servidor para la obtención y manipulación de datos; usando el sistema de tipos de GraphQL, es posible manejar datos de varias fuentes y combinar, por ejemplo, bases de datos

relacionales con bases de datos NoSQL; a diferencia de REST, GraphQL proporciona un único API endpoint y admite consultas flexibles sobre datos vinculados (Nogatz & Seipel, 2017).

Desde que GraphQL fue lanzado para uso público ha sido adoptado por empresas de diversos sectores como la tecnología (GitHub), entretenimiento (Netflix), finanzas (PayPal), viajes (KLM), entre otros. GraphQL es compatible con múltiples lenguajes de programación, por ejemplo, JavaScript, Python, Java, Golang, Ruby, entre otros (Priyatna et al., 2019).

El modo de interacción entre diferentes clientes y el servidor GraphQL se muestra en la Figura 4.

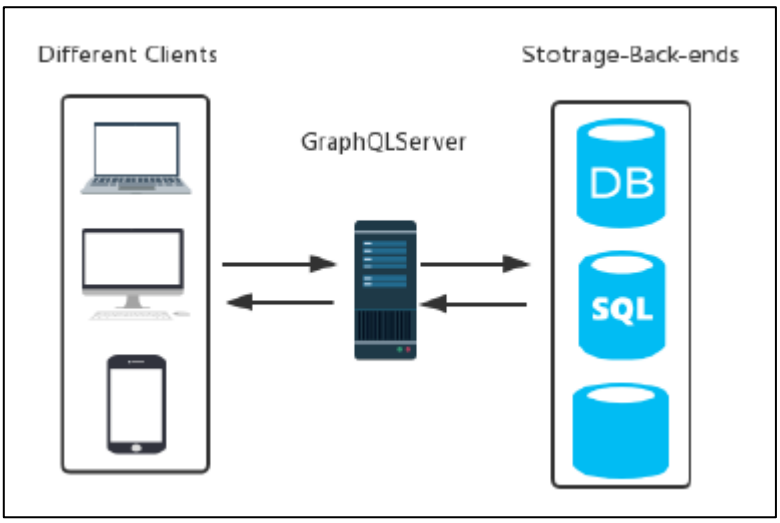


Figura 4. Modo de interacción del servidor GraphQL
Fuente: (Guo et al., 2018)

En la Figura 4, se puede observar que diferentes aplicaciones cliente pueden enviar peticiones al servidor GraphQL, verificar y recopilar información de diferentes fuentes de datos para ejecutar consultas.

El modo de interacción GraphQL consta de los siguientes dos mensajes:

a) Una solicitud, que debe proporcionarse en un lenguaje de consulta bien definido y enviarse como una cadena simple al único endpoint GraphQL.
b) Una respuesta del servidor GraphQL especificada como un documento JSON.

Una consulta GraphQL es jerárquica y estructurada, lo que proporciona una sola entidad con todas las relaciones desde un API endpoint. La estructura de la consulta representa los datos que se espera sean retornados. En general, cada nivel

de consulta GraphQL corresponde a un tipo específico y se pueden anidar o incluso definirse de forma recursiva (Guo et al., 2018).

1.6. API-GitHub

GitHub es un servicio de alojamiento en línea para repositorios Git, con 24 millones de usuarios en 200 países y 67 millones de repositorios en el año 2017. Desde su lanzamiento inicial en 2008, GitHub ha tenido un notable crecimiento en cuanto a popularidad (Russell et al., 2018).

GitHub también proporciona un mecanismo en línea para acceder, compartir y discutir millones de repositorios. Los repositorios públicos de GitHub sólo representan un subconjunto de todos los repositorios de Git (Krohn & Weninger, 2019). Por lo tanto, GitHub se está convirtiendo en una de las fuentes de artefactos de software más importantes en Internet (Kalliamvakou et al., 2014).

GitHub dispone de dos APIs oficiales: la versión 3, conocida como API REST, y la versión 4, que es la API GraphQL. Ambas APIs retornan sus resultados en formato JSON, lo que facilita la interoperabilidad con una amplia gama de aplicaciones y lenguajes de programación (Mombach & Valente, 2018).

Ambas APIs proveen una amplia gama de datos relacionados con los repositorios (repositories), usuarios (users), incidencias (issues), peticiones de cambios (pull requests), entre otros. Por ejemplo, los desarrolladores pueden utilizar estas APIs para obtener información sobre el estado de un repositorio, los detalles de los commits, la lista de colaboradores, y el historial de incidencias y pull requests (Mombach & Valente, 2018).

Las APIs de GitHub son herramientas poderosas que permiten a los desarrolladores interactuar de manera programática con la plataforma, facilitando la creación de aplicaciones y servicios que se integran perfectamente con los flujos de trabajo de desarrollo modernos.

1.7. Investigación experimental en la Ingeniería de Software

La experimentación en la ingeniería de software es un campo que se enfoca en la recolección y análisis de datos para mejorar la eficiencia y la calidad del software; se basa en principios empíricos y científicos que permiten evaluar nuevas tecnologías, herramientas y metodologías de desarrollo (Wohlin et al., 2012).

Los entornos de experimentación permiten a los desarrolladores tomar decisiones informadas sobre diversos aspectos del proceso de desarrollo, como la selección de métodos de prueba, la estimación de costos y la evaluación de la satisfacción del cliente (Wohlin et al., 2012).

La guía de Wohlin et al. (2012) proporciona un marco estructurado para llevar a cabo experimentos en el ámbito de la ingeniería de software, asegurando que los procedimientos y metodologías sean rigurosos y válidos. Describe detalladamente el proceso experimental, esto incluye la definición clara de los objetivos del experimento, la selección de variables a evaluar, el diseño de los experimentos, la recolección de datos y el análisis de los resultados.

1.8. Norma ISO/IEC 25023

La finalidad de la norma ISO/IEC 25023 es establecer métricas de calidad para evaluar cuantitativamente la calidad del sistema y del producto de software de acuerdo a las características y subcaracterísticas definidas en la norma ISO/IEC 25010 (López, 2023).

El modelo de calidad, establecido en la norma ISO/IEC 25010, proporciona las siguientes características y subcaracterísticas de calidad:

1. Eficacia
2. Eficiencia
3. Satisfacción
    3.1. Utilidad
    3.2. Confianza
    3.3. Placer (experiencia del usuario)
    3.4. Confort (ergonómico)
4. Libertad de riesgo
    4.1. Reducción de riesgos económicos
    4.2. Reducción de riesgos de salud y seguridad
    4.3. Reducción de riesgos ambientales
5. Cobertura del contexto
    5.1. Integridad del contexto
    5.2. Flexibilidad (ISO/IEC_25022, 2016)

De las características de calidad anteriormente mencionadas, en este trabajo de investigación se abordará la *eficiencia.*

Sabiendo que la norma ISO/IEC 25023 está diseñada para satisfacer las necesidades de los usuarios con respecto a la calidad del sistema y del producto de software, las métricas que define esta norma para la característica de *eficiencia* forman parte de los atributos de calidad del rendimiento y son las siguientes: comportamiento del tiempo, utilización de recursos y capacidad (Véliz, 2023).

Las métricas de eficiencia del rendimiento se emplean para evaluar el rendimiento relativo a la cantidad de recursos utilizados bajo condiciones establecidas; los recursos pueden incluir otros productos de software, la configuración de software y hardware del sistema y los materiales (por ejemplo, papel de impresión, medios de almacenamiento) (INTECO, 2020).

Dentro de las métricas del comportamiento del tiempo está aquella conocida como *tiempo medio de respuesta*; esta métrica mide el tiempo promedio empleado para completar un trabajo o un proceso asíncrono, es fundamental para evaluar la eficiencia y la rapidez (INTECO, 2020). El cálculo del tiempo medio de respuesta se realiza utilizando la siguiente fórmula:

$$X = \sum_{i=1}^{n} (B_i = A_i)/n \qquad (1)$$

En la cual:

- A_i representa el tiempo de comienzo de un trabajo i
- B_i representa el tiempo de finalización del trabajo i
- n representa la cantidad de medidas (INTECO, 2020)

1.9. Metodología Ágil SCRUM

El marco de trabajo Scrum es considerado uno de los primeros métodos ágiles que surgieron cuando un grupo de individuos acuñó el Manifiesto Ágil en 2001. Facilita el proceso de Gestión de Proyectos y trae consigo principios y valores ágiles. Scrum es adecuado para manejar proyectos de software (Ganesh & Narayanan, 2019).

Scrum es básicamente un enfoque orientado a las personas. Promueve la colaboración, la sincronización del equipo y otros artefactos relacionados que conducen a la participación del proyecto de los respectivos *stakeholders*. Scrum establece el rol de *Propietario del producto*, que puede ser el Cliente o el representante del Cliente. Además, promueve la auto organización entre los

miembros del equipo, lo cual resulta en la mejora del proceso de software y, por lo tanto, aumenta la productividad (Ganesh & Narayanan, 2019).

Scrum fomenta la creatividad de cada miembro del equipo, permitiéndoles a todos asumir el papel que desean durante el proyecto. Cada persona puede ser un desarrollador, un tester, un diseñador, dependiendo de las responsabilidades que asuma entre dos sprints. Teniendo en cuenta la diversidad de los proyectos de TI, tiene sentido comprender en qué tipo de proyectos se debería usar la metodología Scrum y dónde existe la necesidad de una gestión y control más explícitos (Nicula & Ghimii, 2019).

Las metodologías ágiles son una combinación de modelos de proceso iterativos e incrementales con enfoque en la adaptabilidad de los procesos y la satisfacción del cliente mediante la entrega rápida de un producto de software funcional; los métodos ágiles dividen el producto en pequeños bloques incrementales (Nicula & Ghimii, 2019).

Estos bloques se trabajan en iteraciones, cada iteración generalmente dura aproximadamente de una a tres semanas e involucra equipos interfuncionales que trabajan simultáneamente en varias áreas como planificación, análisis de requerimientos, diseño, codificación, pruebas unitarias y pruebas de aceptación; al final de la iteración, se muestra un producto funcional al cliente y a los *stakeholders* importantes; las metodologías ágiles se basan en los métodos de desarrollo de software adaptativos, mientras que los modelos tradicionales como el modelo en cascada se basan en un enfoque predictivo (Nicula & Ghimii, 2019).

Los equipos predictivos en los modelos tradicionales generalmente trabajan con una planificación detallada y tienen un pronóstico completo de las tareas y características exactas que se entregarán en los próximos meses o durante el ciclo de vida del producto; los métodos predictivos dependen completamente del análisis de requerimientos y la planificación realizada al comienzo del ciclo; cualquier cambio que se incorpore pasará por una estricta gestión de control de cambios y priorización; las metodologías ágiles utilizan un enfoque adaptativo donde no hay una planificación detallada y hay claridad en las tareas futuras sólo con respecto a qué características deben desarrollarse (Nicula & Ghimii, 2019).

Existe un desarrollo basado en características y el equipo se adapta dinámicamente a los requerimientos cambiantes del producto; el producto se prueba con mucha frecuencia, a través de las iteraciones *release*, minimizando el riesgo de

fallas importantes en el futuro; la interacción con el cliente es la columna vertebral de esta metodología ágil, y la comunicación abierta con documentación mínima son las características típicas de un entorno de desarrollo ágil; los equipos ágiles trabajan en estrecha colaboración entre ellos y se sitúan con mayor frecuencia en la misma ubicación geográfica (Nicula & Ghimii, 2019).

Scrum es actualmente uno de los métodos ágiles más estudiados y utilizados, tanto por su novedad como por el supuesto de que este método puede mejorar la productividad del proyecto; el método utiliza un proceso empírico basado en la flexibilidad, la adaptabilidad y la productividad, diferente de la rígida y muy centrada en la manera de planificar el desarrollo con el modelo en cascada; Scrum pone especial énfasis en la comunicación y la colaboración, el funcionamiento del software, la auto organización del equipo y la flexibilidad para adaptarse a las realidades de negocios emergentes (Nicula & Ghimii, 2019).

Establece actividades de gestión de alta frecuencia para rastrear problemas en tiempo real durante el proceso de desarrollo de un proyecto; en la gestión tradicional, existe una fuerte mentalidad de comando y control con roles y responsabilidades claramente definidos para cada miembro del equipo, mientras que, en muchas metodologías ágiles, especialmente en Scrum, no hay un líder real; se requieren mecanismos básicos de colaboración para que funcione y el entorno debe configurarse de modo que las personas tengan un incentivo para ayudarse mutuamente a fin de optimizarlo todo, en lugar de optimizar su nicho individual; de lo contrario, las empresas y los individuos no podrán obtener todos los beneficios de Scrum (Nicula & Ghimii, 2019).

En la Figura 5 se observa a Scrum como un proceso, donde toma un proyecto y lo divide en ciclos regulares conocidos como *Sprints*; los *sprints* generalmente duran entre 1 y 2 semanas e involucran equipos funcionales de 5 a 9 personas; cada *sprint* abarca algún segmento de trabajo que el equipo ha acordado realizar; los equipos que trabajan juntos en cada *sprint* son generalmente multifuncionales y autogestionados. Toman una determinada tarea que se ha establecido en el *"product backlog"*, similar a una lista de objetivos para el producto y lo llevan a una fase de entrega (Nicula & Ghimii, 2019).

Un aspecto determinante de Scrum es que cada sprint termina en algún tipo de entrega, ya sea una idea formulada, una simulación útil o un componente completamente diseñado; esto permite bucles de retroalimentación iterativos en el

proceso de diseño y evita que se avance en la dirección incorrecta (Nicula & Ghimii, 2019).

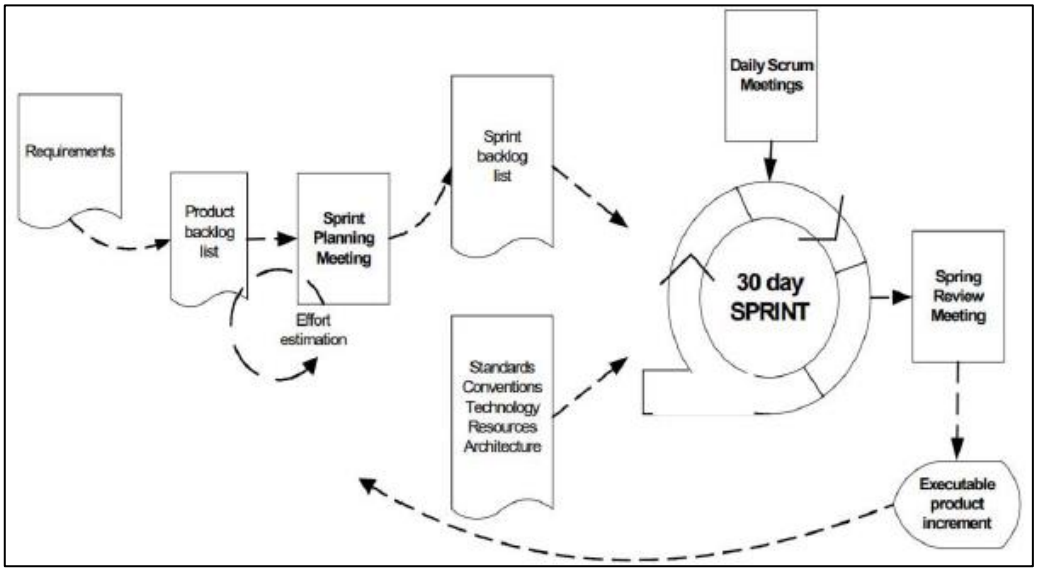


Figura 5. Modelo Scrum
Fuente: (Nicula & Ghimii, 2019)

Dentro de cada Sprint hay un mini proyecto en cascada ya que cada incremento de producto ejecutable necesita requisitos mínimos, diseño, implementación y pruebas. Lo que hace a Scrum diferente de cualquier otra metodología es la combinación entre los principios en los que se basan todas las metodologías ágiles y el número limitado de roles dentro del equipo (Nicula & Ghimii, 2019).

Uno de los principios del desarrollo de software ágil menciona que "Las mejores arquitecturas, requisitos y diseños surgen de los equipos autoorganizados". Por lo tanto, los equipos Scrum tienen la libertad de elegir cómo realizar su trabajo, en lugar de ser dirigidos por otras partes interesadas o gerentes. Los miembros son facultados por la organización para estructurar y administrar su propio trabajo (Nicula & Ghimii, 2019).

Los equipos autogestionados ofrecen ventajas potenciales sobre los equipos administrados tradicionalmente porque otorgan autoridad para la toma de decisiones al nivel de problemas operativos y aumentan la velocidad y precisión de la resolución de problemas (Nicula & Ghimii, 2019).

CAPÍTULO 2

Desarrollo

En este capítulo se describen las etapas de desarrollo de la metodología ágil Scrum, aplicadas en el proceso de implementación de una prueba de concepto, misma que consiste en un cliente web que consume los servicios de la API REST y API GraphQL de GitHub, con el propósito de realizar un experimento que permita comparar estas dos tecnologías en su capacidad de consulta de datos, donde el principal factor a tener en cuenta será el tiempo de respuesta. Para la experimentación se empleará la metodología de Wohlin.

En la Figura 6, se da a conocer de manera gráfica las diferentes etapas de la metodología Scrum.

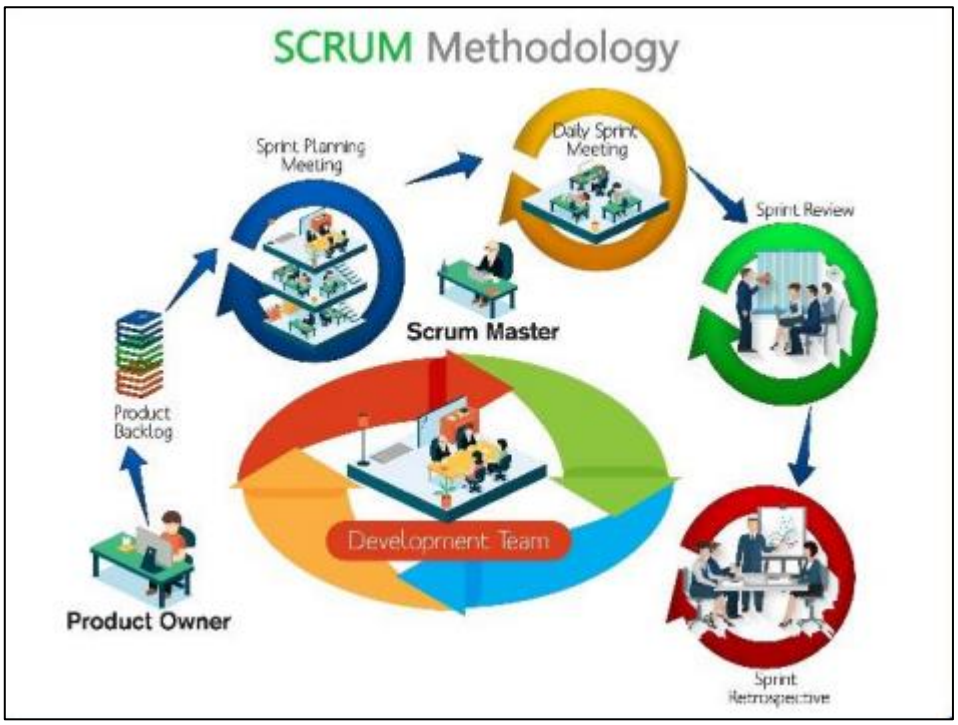


Figura 6. Metodología Scrum
Fuente: (Gómez-Sierra & López-Bustamante, 2019)

2.1. Análisis

2.1.1. Equipo de Scrum

En este punto se procede a definir el equipo de Scrum que se encargará de desarrollar el proyecto. La Tabla 2, detalla los roles y responsabilidades de cada integrante del grupo de trabajo.

Tabla 2. Roles en el proyecto

Integrante	Rol	Descripción
PhD. Antonio Quiña	Product Owner	Responsable de definir y evaluar los requisitos del proyecto, director de este trabajo de titulación y docente investigador de la Universidad Técnica del Norte
Jimmy Ibarra	Scrum Master Jefe del proyecto	Responsable del avance del proyecto y tesista
Jimmy Ibarra	Development Team Equipo de desarrollo	Responsable del desarrollo del software y tesista

2.1.2. Especificación de Requisitos

Según la metodología de desarrollo ágil Scrum, los requerimientos se definen utilizando historias de usuario. Estas historias representan descripciones breves y específicas de las funcionalidades que el sistema debe proporcionar desde la perspectiva del usuario final (Scrum Manager, 2014).

Cada historia de usuario se centra en una necesidad particular del usuario y se escribe en un lenguaje sencillo de manera que todos los miembros del equipo puedan entender (Scrum Manager, 2014).

En el contexto de este proyecto, las historias de usuario han sido elaboradas en colaboración entre el Product Owner y el Scrum Master; el formato de historia de usuario que se utilizó es una adaptación de Scrum Manager.

Desde la Tabla 3 hasta la Tabla 10, se presentan las historias de usuario que se han definido para este proyecto.

Tabla 3. Historia Usuario 1

Historias de Usuario		
Código: H.U.01	**Usuario:** Administrador	
Denominación: Gestión del token para acceder a los recursos		
Prioridad: Alta	**Dependencias:** Ninguna	**Estimación:** 5h

Continúa …

Descripción:
Se requiere obtener un token para tener acceso a los servicios de las APIs REST y GraphQL de GitHub.
Condiciones de aceptación:

Tabla 4. Historia Usuario 2

Historias de Usuario		
Código: H.U.02	**Usuario:** Administrador	
Denominación: Gestión del recurso usuarios (users)		
Prioridad: Alta	**Dependencias:** H.U.01	**Estimación:** 5h
Descripción: Se requiere obtener los usuarios registrados en GitHub desde las APIs REST y GraphQL. Esta funcionalidad debe ser capaz de recuperar la cantidad de datos solicitada, estableciendo así el nivel 1 de consumo. **Condiciones de aceptación:** - Tener una opción que permita elegir el nivel 1 de consumo. - Tener una opción que permita elegir la cantidad de registros que se desea recuperar desde las APIs. - La recuperación exitosa de la lista de usuarios, tanto desde la API-REST como desde la API-GraphQL, se entenderá como una respuesta correcta, lo que permitirá visualizar en pantalla los tiempos de respuesta de cada API en milisegundos (ms). - Renderizar también en pantalla el identificador del nivel de consumo, el número de repetición y la cantidad de registros recuperados desde las APIs.		

Tabla 5. Historia Usuario 3

Historias de Usuario		
Código: H.U.03	**Usuario:** Administrador	
Denominación: Gestión del recurso repositorios (repositories)		
Prioridad: Alta	**Dependencias:** H.U.02	**Estimación:** 6h
Descripción: Se requiere obtener los repositorios de los usuarios registrados en GitHub desde las APIs REST y GraphQL. Debe contener los repositorios de un usuario específico. Esta funcionalidad debe ser capaz de recuperar la cantidad de datos solicitada, estableciendo así el nivel 2 de consumo.		

Continúa …

Condiciones de aceptación:

- Tener una opción que permita elegir el nivel 2 de consumo.
- Tener una opción que permita elegir la cantidad de registros que se desea recuperar desde las APIs.
- La recuperación exitosa de la lista de repositorios, tanto desde la API-REST como desde la API-GraphQL, se entenderá como una respuesta correcta, lo que permitirá visualizar en pantalla los tiempos de respuesta de cada API en milisegundos (ms).
- Renderizar también en pantalla el identificador del nivel de consumo, el número de repetición y la cantidad de registros recuperados desde las APIs.

Tabla 6. Historia Usuario 4

Historias de Usuario		
Código: H.U.04	**Usuario:** Administrador	
Denominación: Gestión del recurso ramas (branches)		
Prioridad: Alta	**Dependencias:** H.U.03	**Estimación:** 7h

Descripción:

Se requiere obtener las ramas de los repositorios de los usuarios registrados en GitHub desde las APIs REST y GraphQL. Debe contener las ramas de un repositorio específico. Esta funcionalidad debe ser capaz de recuperar la cantidad de datos solicitada, estableciendo así el nivel 3 de consumo.

Condiciones de aceptación:

- Tener una opción que permita elegir el nivel 3 de consumo.
- Tener una opción que permita elegir la cantidad de registros que se desea recuperar desde las APIs.
- La recuperación exitosa de la lista de ramas, tanto desde la API-REST como desde la API-GraphQL, se entenderá como una respuesta correcta, lo que permitirá visualizar en pantalla los tiempos de respuesta de cada API en milisegundos (ms).
- Renderizar también en pantalla el identificador del nivel de consumo, el número de repetición y la cantidad de registros recuperados desde las APIs.

Tabla 7. Historia Usuario 5

Historias de Usuario		
Código: H.U.05	**Usuario:** Administrador	
Denominación: Gestión de menús de selección		
Prioridad: Media	**Dependencias:** Ninguna	**Estimación:** 5h

Continúa …

Descripción:
Se requiere elegir desde la interfaz gráfica el nivel de consumo, es decir, el identificador del caso de uso requerido. También se debe seleccionar la cantidad de registros que se desea recuperar desde las APIs REST y GraphQL de GitHub.

Condiciones de aceptación:

- Las opciones para elegir el nivel de consumo son: Nivel 1, Nivel 2 y Nivel 3.
- La cantidad de registros que se puede seleccionar para cada nivel de consumo es de 1, 10, 100, 1000 y 10000 registros.

Tabla 8. Historia Usuario 6

Historias de Usuario		
Código: H.U.06	**Usuario:** Administrador	
Denominación: Gestión del nivel 1 de consumo		
Prioridad: Alta	**Dependencias:** H.U.01, H.U.02	**Estimación:** 10h

Descripción:
Se requiere ejecutar el nivel 1 de consumo para obtener la cantidad de registros que haya sido especificada tanto desde la API REST como desde la API GraphQL de GitHub. Adicionalmente, se requiere conocer el tiempo de respuesta de ambas APIs, mismo que posteriormente permitirá llevar a cabo la comparación.

Condiciones de aceptación:

- Renderizar en pantalla el tiempo de respuesta de las dos APIs.
- La unidad de medida del tiempo será milisegundos (ms).
- El nivel 1 de consumo hará uso de la petición (request) que ha sido establecida en la historia de usuario 2 (H.U.02).

Tabla 9. Historia Usuario 7

Historias de Usuario		
Código: H.U.07	**Usuario:** Administrador	
Denominación: Gestión del nivel 2 de consumo		
Prioridad: Alta	**Dependencias:** H.U.01, H.U.02, H.U.03	**Estimación:** 15h

Descripción:
Se requiere ejecutar el nivel 2 de consumo para obtener la cantidad de registros que haya sido especificada tanto desde la API REST como desde la API GraphQL de GitHub.

Continúa …

Adicionalmente, se requiere conocer el tiempo de respuesta de ambas APIs, mismo que posteriormente permitirá llevar a cabo la comparación.

Condiciones de aceptación:

- Renderizar en pantalla el tiempo de respuesta de las dos APIs.
- La unidad de medida del tiempo será milisegundos (ms).
- El nivel 2 de consumo hará uso de la petición (request) que ha sido establecida en la historia de usuario 3 (H.U.03) con los datos que arrojará previamente la ejecución de la historia de usuario 2 (H.U.02).

Tabla 10. Historia Usuario 8

Historias de Usuario		
Código: H.U.08	**Usuario:** Administrador	
Denominación: Gestión del nivel 3 de consumo		
Prioridad: Alta	**Dependencias:** H.U.01, H.U.02, H.U.03, H.U.04	**Estimación:** 20h

Descripción:

Se requiere ejecutar el nivel 3 de consumo para obtener la cantidad de registros que haya sido especificada tanto desde la API REST como desde la API GraphQL de GitHub. Adicionalmente, se requiere conocer el tiempo de respuesta de ambas APIs, mismo que posteriormente permitirá llevar a cabo la comparación.

Condiciones de aceptación:

- Renderizar en pantalla el tiempo de respuesta de las dos APIs.
- La unidad de medida del tiempo será milisegundos (ms).
- El nivel 3 de consumo hará uso de la petición (request) que ha sido establecida en la historia de usuario 4 (H.U.04) con los datos que arrojará previamente la ejecución de las historias de usuario 2 (H.U.02) y 3 (H.U.03).

2.1.3. Product Backlog

Una vez que las historias de usuario han sido definidas, el Product Owner establece prioridades para cada una de ellas, dándoles un orden para posteriormente proceder al desarrollo del producto de software. Para el caso particular del presente proyecto, el Product Backlog es el que se observa en la Tabla 11.

Tabla 11. Product Backlog

Nro.	Código	Prioridad	Descripción	Estimación (Horas)
1	H.U.01	Alta	Gestión del token para acceder a los recursos	5
2	H.U.02	Alta	Gestión del recurso usuarios (users)	5
3	H.U.03	Alta	Gestión del recurso repositorios (repositories)	6
4	H.U.04	Alta	Gestión del recurso ramas (branches)	7
5	H.U.05	Media	Gestión de menús de selección	5
6	H.U.06	Alta	Gestión del nivel 1	10
7	H.U.07	Alta	Gestión del nivel 2	15
8	H.U.08	Alta	Gestión del nivel 3	20

2.2. Diseño del Sprint 0

La planificación del proceso de desarrollo de software para este proyecto se la llevó a cabo durante una iteración llamada Sprint 0, en la cual se procedió a definir la arquitectura tecnológica y también se elaboró el diagrama de procesos del cliente web.

2.2.1. Resumen del Sprint 0

En la reunión de planificación que tuvo lugar el día jueves 27/11/2023 con la asistencia del Scrum Master, Product Owner y Equipo de Desarrollo, se procedió a definir las tareas que se debe realizar, tal como lo indica la Tabla 12.

Tabla 12. Tareas – Sprint 0

Tarea	Horas
Definir la arquitectura tecnológica del software	5
Realizar un diagrama de procesos	7
Definir los niveles del software	10
Realizar un Scrum meeting (reunión de revisión sprint 0)	3
Realizar un Scrum meeting (reunión de retrospectiva sprint 0)	3
Realizar un Scrum meeting (reunión de planificación sprint 1)	3

Las reuniones de revisión y retrospectiva se las realizó el día miércoles 13/12/2023, con la asistencia del Scrum Master, Product Owner y Equipo de Desarrollo. Dentro de los resultados de estas reuniones están la aprobación de la arquitectura tecnológica del producto de software que se va a desarrollar, una versión mejorada del diagrama de procesos y la aprobación de los niveles de consumo que serán implementados en el software.

2.2.2. Arquitectura Tecnológica

En el presente proyecto surge la necesidad de desarrollar una aplicación web cliente, front-end, que consuma las APIs públicas de GITHUB, la versión 3 desarrollada con tecnología REST y la versión 4 desarrollada con tecnología GraphQL.

Por lo tanto, en la parte del cliente se utilizó las siguientes herramientas: Apollo Client, Angular y Bootstrap. La parte del servidor la provee el sitio web https://www.github.com, es decir, no se realizó ningún desarrollo de back-end en el presente proyecto. En la Figura 7, se muestra la arquitectura definida.

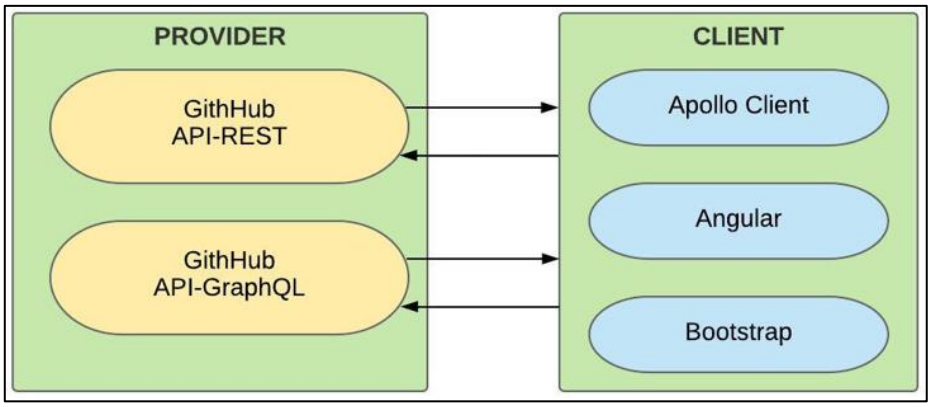


Figura 7. Arquitectura Tecnológica

2.2.3. Diagrama de Procesos

En la Figura 8, se expone una representación gráfica de los principales procesos que intervienen en el cliente web.

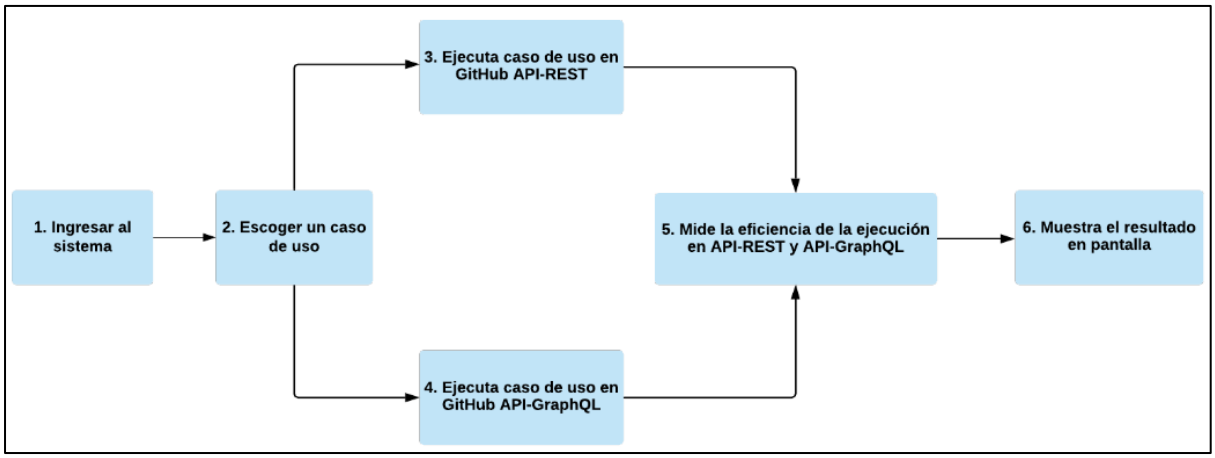


Figura 8. Diagrama de Procesos Cliente Web

La Tabla 13, muestra con mayor detalle cada etapa del diagrama de procesos.

Tabla 13. Descripción del Diagrama de Procesos del Cliente Web

Nro.	Actividad	Descripción	Responsable
1	Ingresar a la aplicación web	Se muestra la pantalla inicial	Usuario Final
2	Escoger un caso de uso	Se muestra un menú de opciones, cada opción corresponde a un caso de uso específico	Usuario Final
3	Ejecuta caso de uso en GitHub API-REST	Se realiza la ejecución automática del caso de uso elegido, a través del consumo de la API REST de GitHub	Usuario Final
4	Ejecuta caso de uso en GitHub API-GraphQL	Se realiza la ejecución automática del caso de uso elegido, a través del consumo de la API GraphQL de GitHub	Usuario Final
5	Mide la eficiencia de la ejecución en API-REST y API-GraphQL	Se captura los tiempos de respuesta de la API REST de GitHub y de la API GraphQL de GitHub	Usuario Final
6	Muestra el resultado en pantalla	Se renderiza en pantalla los datos solicitados en el caso de uso y los tiempos de respuesta previamente obtenidos. Son dos grupos de datos: los provenientes de la API REST de GitHub y de la API GraphQL de GitHub	Usuario Final

2.2.4. Casos de uso de consultas al API-GitHub

Cada nivel de consumo o caso de uso corresponde a una funcionalidad diferente.

- **Nivel 1**
  En este nivel, la funcionalidad es *usuarios*, misma que obtiene una lista de usuarios que pertenecen a la organización GitHub.

- **Nivel 2**
  La funcionalidad para este nivel es *repositorios* y se encarga de obtener los repositorios que son propiedad de un usuario específico. Previamente, se hace uso del nivel 1 para extraer el identificador de cada usuario, en este caso, el atributo login.

- **Nivel 3**
  A este nivel le corresponde la funcionalidad *ramas* y se encarga de obtener las ramas que existen en un repositorio específico. Previamente, se emplea el nivel 2 para extraer los identificadores de los usuarios y sus repositorios, en este caso, los atributos owner.login y name, respectivamente.

2.3. Implementación del Cliente Web

El Cliente Web fue desarrollado o implementado desde un enfoque iterativo-incremental. Una iteración es un Sprint, en Scrum. Un Sprint contiene las siguientes actividades: reunión de planificación, reuniones diarias, reunión de revisión y reunión de retrospectiva.

NOTA: Cabe indicar que en el presente proyecto las reuniones diarias no aplican porque el equipo de desarrollo está conformado por una única persona.

La Tabla 14, presenta un reporte de los Sprints que forman parte de este proyecto.

Tabla 14. Reporte Sprints

Nombre del Sprint	Fecha Comienzo	Fecha Finalización	Tiempo (Horas)
Sprint 0	27/11/2023	13/12/2023	25
Sprint 1	14/12/2023	04/01/2024	40
Sprint 2	05/01/2024	19/01/2024	30
Sprint 3	20/01/2024	03/02/2024	30
Sprint 4	04/02/2024	18/02/2024	30

2.3.1. Sprint 1

Planificación del Sprint 1

En este primer sprint se revisará la documentación de las APIs de GitHub, esto es, la API REST (versión 3) y API GraphQL (versión 4), con el fin de tener un mayor contexto sobre cómo trabajar con cada una de ellas, de manera especial sobre cómo obtener un token de autenticación para tener acceso a los recursos que proveen.

- **Reunión de planificación**

  La reunión de planificación se realizó de acuerdo a la siguiente información:

  - **Fecha:** 14/12/2023
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** planificar y definir el Sprint Backlog del Sprint 1

- **Sprint Backlog del Sprint 1**

  La Tabla 15, presenta el Sprint Backlog correspondiente al Sprint 1, en donde se registran las historias de usuario que se desarrollarán durante este Sprint. También se listan las tareas asignadas al Equipo de Desarrollo.

Tabla 15. Sprint Backlog del Sprint 1

Historia de Usuario	Denominación	Tareas	Tiempo (Horas)
H.U.01	Gestión del token para acceder a los recursos	Investigar el uso de la API REST de GitHub	10
		Investigar el uso de la API GraphQL de GitHub	10
		Generar o configurar el token de acceso a los recursos de la API REST y API GraphQL de GitHub	5
H.U.02	Gestión del recurso usuarios (users)	Implementar las dos soluciones (REST y GraphQL) para el nivel 1 de consumo, es decir, escribir el código necesario para obtener la lista de usuarios de GitHub desde la API REST y desde la API GraphQL	20

Continúa …

H.U.03	Gestión del recurso repositorios (repositories)	Implementación/Codificación de las dos soluciones (REST y GraphQL) para el nivel 2 de consumo	15
H.U.04	Gestión del recurso ramas (branches)	Implementación/Codificación de las dos soluciones (REST y GraphQL) para el nivel 3 de consumo	20
Reuniones Scrum	Planificación		1
	Revisión		1
	Retrospectiva		1
		TOTAL HORAS	95

Revisión del Sprint 1

Como resultado del primer sprint, se realizó la revisión de la documentación de las APIs de GitHub, esto es, la API REST (versión 3) y API GraphQL (versión 4), para conocer cómo trabajar con cada una de ellas y se procedió a obtener un token de autenticación para acceder a sus recursos.

- **Reunión de Revisión**

  En este punto se procede a verificar el cumplimiento de la totalidad de las tareas definidas en el Product Backlog correspondiente al Sprint 1, en las fechas acordadas.

  - **Fecha:** 04/01/2023
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** revisar el crecimiento del producto de software

- **Incremento del producto potencialmente entregable**

  Una vez que se ha realizado las tareas del Sprint 1, se da a conocer los avances más relevantes correspondientes a las Historias de Usuario H.U.01, H.U.02, H.U.03 y H.U.04.

  Obtener el token de acceso

  Según la documentación oficial de la API REST de GitHub (2022) y la API GraphQL de GitHub (n.d.), es posible obtener un token de acceso personal y un token generado por una app. Para el presente proyecto se utilizará un token de acceso personal.

Obtener una lista de usuarios

Para obtener los usuarios, GitHub (2022) indica que es necesario incluir el identificador de la organización de la cual se va a consultar sus usuarios en la URL, en este caso, la organización elegida es *github*, pero puede ser cualquier otra; opcionalmente se puede incluir también el parámetro *per_page* para controlar la cantidad de registros que se desea recuperar. En la cabecera de la petición se debe agregar el token previamente obtenido, como lo indica la Figura 9.

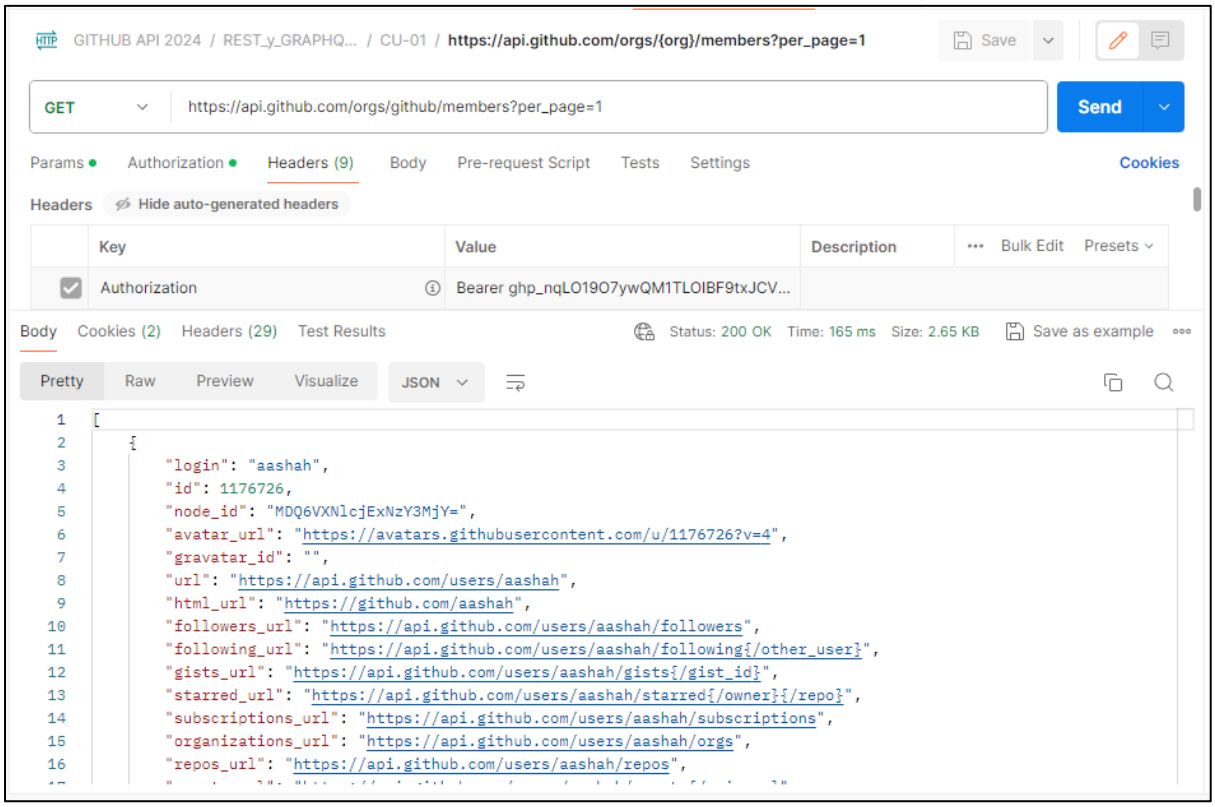


Figura 9. Petición REST para consultar los usuarios de la organización github

Una vez que se ha comprobado que la petición REST para obtener la lista de usuarios de una organización específica funciona correctamente, se procede con la petición GraphQL para obtener de igual manera la lista de usuarios de una organización, como se puede ver en la Figura 10.

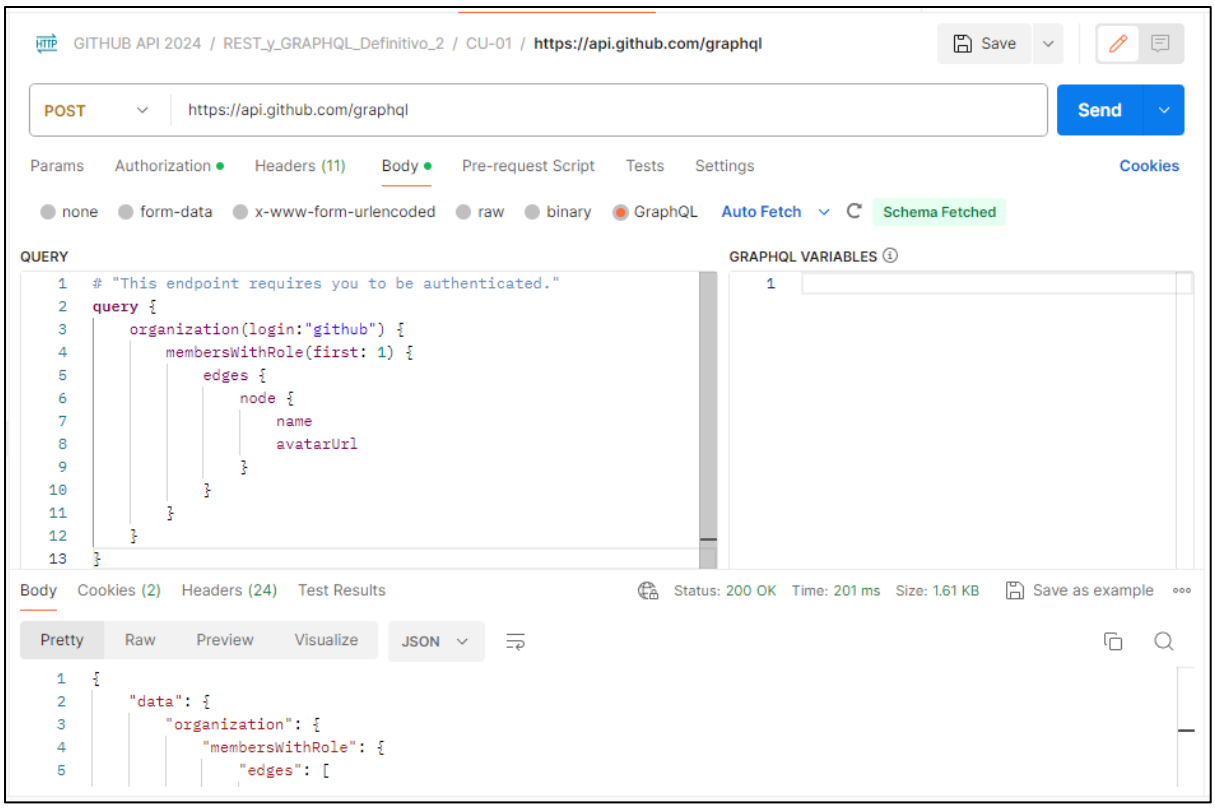


Figura 10. Petición GraphQL para consultar los usuarios de la organización github

Obtener una lista de repositorios

Para obtener los repositorios de un usuario determinado, GitHub (2022) establece que es necesario incluir el identificador del usuario que se va a consultar sus repositorios en la URL; opcionalmente se puede incluir también el parámetro *per_page* para controlar la cantidad de registros que se desea recuperar. En la cabecera de la petición se debe agregar el token previamente obtenido, como lo indica la Figura 11.

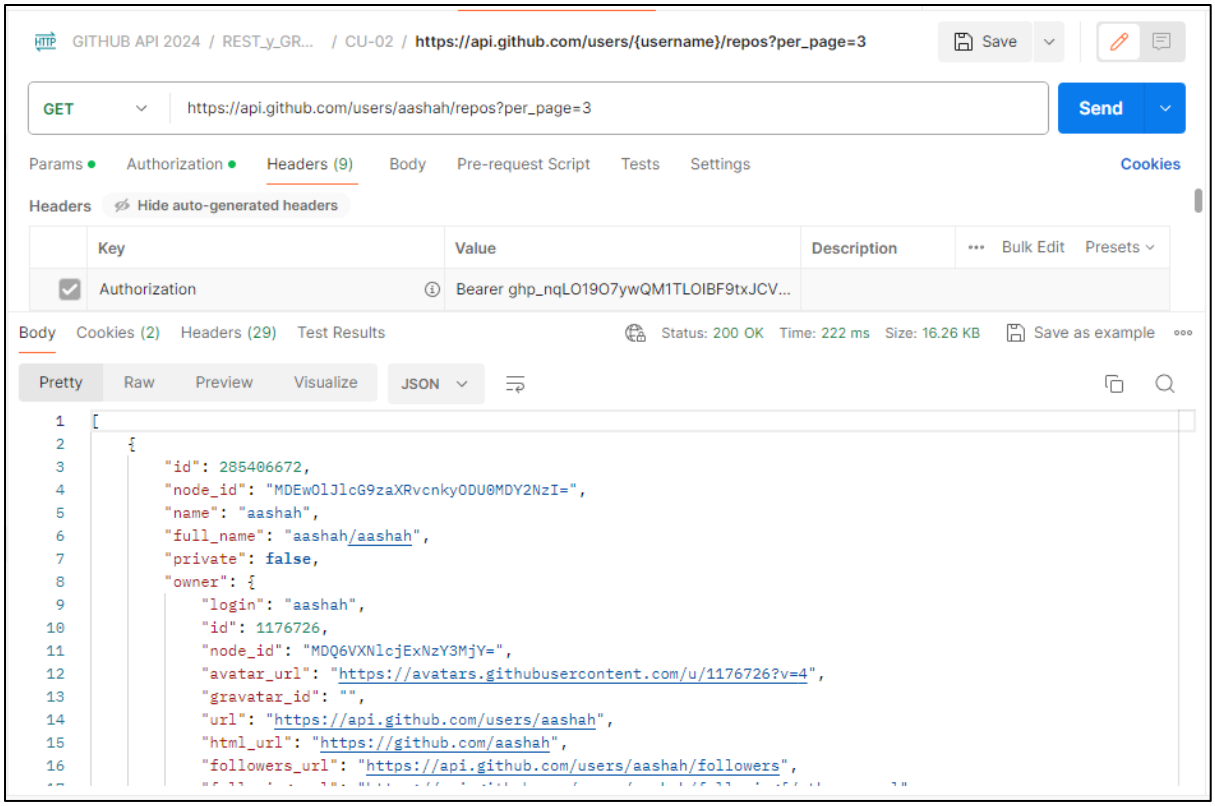


Figura 11. Petición REST para consultar los repositorios de un usuario

Una vez que se ha comprobado que la petición REST para obtener la lista de repositorios de un usuario específico funciona correctamente, se procede ahora con la petición GraphQL homóloga, como se puede ver en la Figura 12.

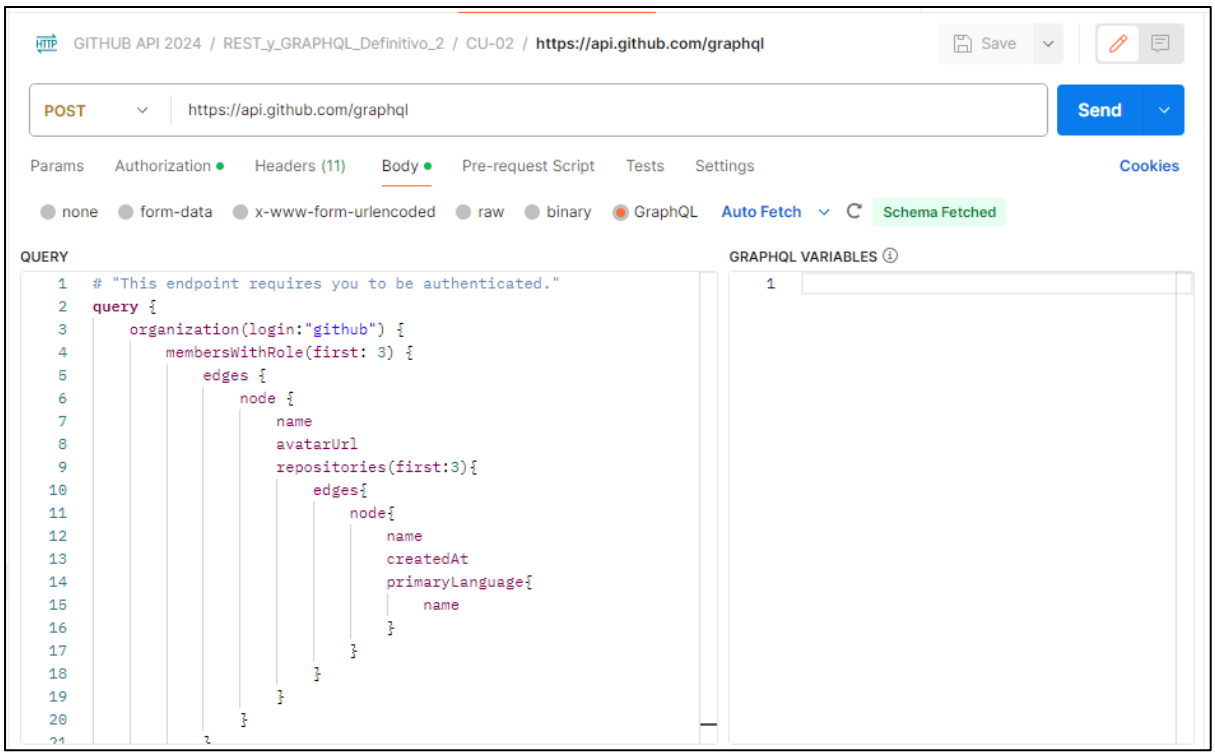


Figura 12. Petición GraphQL para consultar los repositorios de un usuario

Obtener una lista de ramas

Para obtener las ramas de un repositorio específico, GitHub (2022) establece que es necesario incluir en la URL el identificador del usuario propietario del repositorio y el identificador del repositorio que se va a consultar sus ramas; opcionalmente se puede incluir también el parámetro *per_page* para controlar la cantidad de registros que se desea recuperar. En la cabecera de la petición se debe agregar el token previamente obtenido, como lo indica la Figura 13.

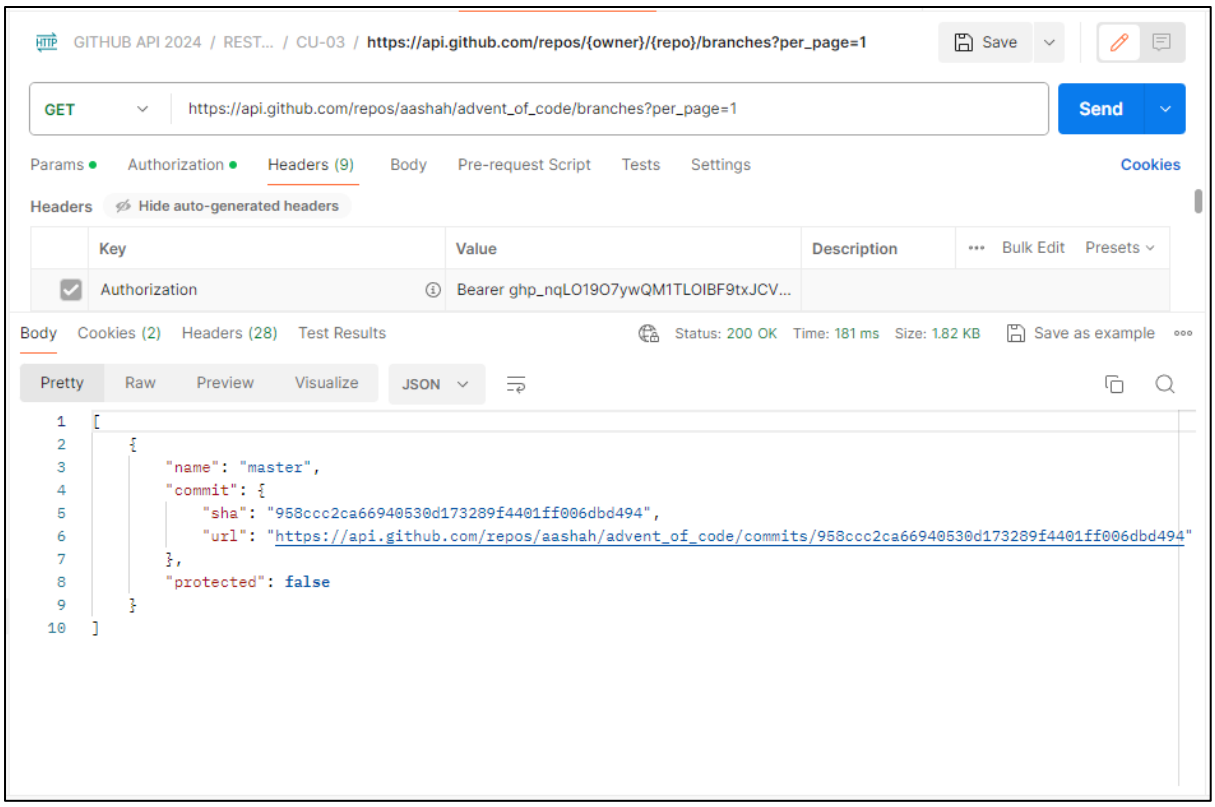


Figura 13. Petición REST para consultar las ramas de un repositorio

Una vez que se ha comprobado que la petición REST para obtener la lista de ramas de un repositorio específico funciona correctamente, se procede ahora con la petición GraphQL homóloga, como lo indica la Figura 14.

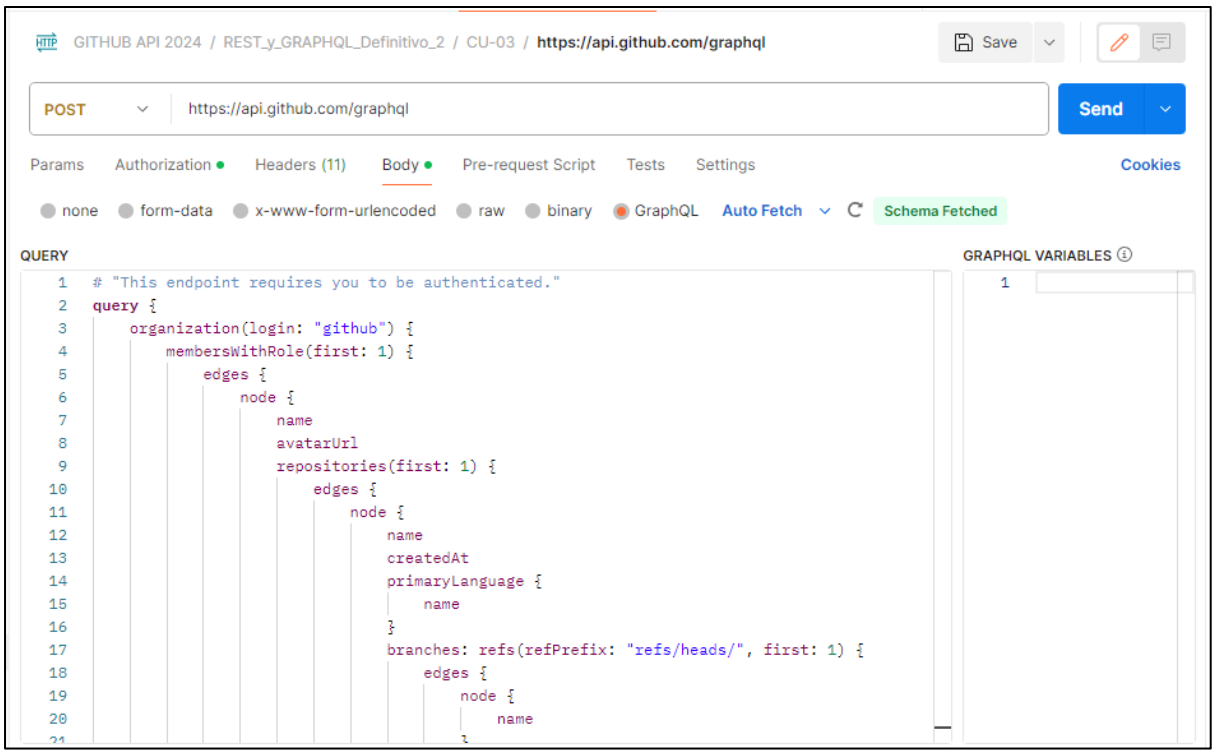


Figura 14. Petición GraphQL para consultar las ramas de un repositorio

Retrospectiva del Sprint 1

- **Reunión de Retrospectiva**

  La reunión de retrospectiva se la realizó según la planificación, respetando las fechas acordadas.

  - **Fecha:** 04/01/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** Analizar las tareas desarrolladas y proponer un plan para mejorar

- **Plan de mejora**

  El plan de mejora consiste de tres aspectos: aciertos (pregunta: ¿Qué salió bien en el Sprint?), errores (pregunta: ¿Qué no salió bien en el Sprint?), mejoras (pregunta: ¿Qué mejoras se procederá a implementar?).

  La Tabla 16, da a conocer los resultados obtenidos en la reunión de retrospectiva y las mejoras que se implementarán en el siguiente Sprint.

Tabla 16. Plan de mejora del Sprint 1

Aciertos	Errores/Problemas	Mejoras
Las peticiones y consultas GraphQL han sido planteadas correctamente, de manera que permiten obtener datos similares a las peticiones REST.	Falta de conocimientos técnicos sobre GraphQL.	En las consultas GraphQL, se debe recuperar la mayor cantidad de atributos posible, de manera que, si las peticiones REST devuelven, por decir, quince atributos, esos mismos quince atributos se debe recuperar en la consulta GraphQL. Esto es con la finalidad de comparar el rendimiento de las dos APIs bajo las mismas condiciones.

2.3.2. Sprint 2

Planificación del Sprint 2

En este sprint se desarrollará la interfaz de usuario del cliente web, misma que hará posible la ejecución de los diferentes niveles de consumo, empezando por la selección de un nivel específico y posteriormente seleccionando la cantidad de registros que se desea recuperar en las consultas.

- **Reunión de Planificación**

  La reunión de planificación se realizó de acuerdo a la siguiente información:

  - **Fecha:** 05/01/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** planificar y definir el Sprint Backlog del Sprint 2

- **Sprint Backlog del Sprint 2**

  La Tabla 17, presenta el Sprint Backlog correspondiente al Sprint 2, en donde se registran las historias de usuario que se desarrollarán durante este Sprint. También se listan las tareas asignadas al Equipo de Desarrollo.

Tabla 17. Sprint Backlog del Sprint 2

Historia de Usuario	Denominación	Tareas	Tiempo (Horas)
H.U.05	Gestión de menús de selección	Configurar el Ambiente de Desarrollo	5
		Crear el proyecto web Angular, importar las librerías necesarias y establecer la estructura de carpetas del proyecto	2
		Configurar GraphQL en el lado del cliente para consumir la API GraphQL de GitHub	10
		Diseñar y desarrollar las pantallas para: - Selección del nivel de consumo - Selección de la cantidad de registros para el nivel de consumo previamente seleccionado	10
H.U.06	Gestión del nivel 1	Diseñar y desarrollar la lógica de programación de la pantalla para ejecutar el nivel 1 de consumo y visualizar los tiempos de respuesta, en milisegundos, de las dos APIs de GitHub (REST y GraphQL)	20
Reuniones Scrum	Planificación		2
	Revisión		2
	Retrospectiva		2
		TOTAL HORAS	36

Revisión del Sprint 2

Como resultado del segundo sprint, se desarrolló la interfaz de usuario del cliente web, la cual consta de tres pantallas principales: selección del nivel de consumo, selección de la cantidad de registros y ejecución del nivel de consumo seleccionado.

- **Reunión de Revisión**

  En este punto se procede a verificar el cumplimiento de la totalidad de las tareas definidas en el Product Backlog correspondiente al Sprint 2, en las fechas acordadas.

  - **Fecha:** 19/01/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** revisar el crecimiento del producto de software

- **Incremento del producto potencialmente entregable**

  Una vez que se ha realizado las tareas del Sprint 2, se da a conocer los avances más relevantes correspondientes a las Historias de Usuario H.U.07 y H.U.08.

  Pantalla para seleccionar el nivel de consumo

  En este punto, se muestran las pantallas de la interfaz de usuario del cliente web. La Figura 15, muestra la pantalla con la que inicia el proceso de ejecución de los diferentes niveles de consumo.

Figura 15. Pantalla para seleccionar un nivel de consumo específico

  Pantalla para seleccionar la cantidad de registros

  La Figura 16, muestra la pantalla que permite elegir la cantidad de registros con la que se desea trabajar durante la ejecución del nivel de consumo previamente seleccionado.

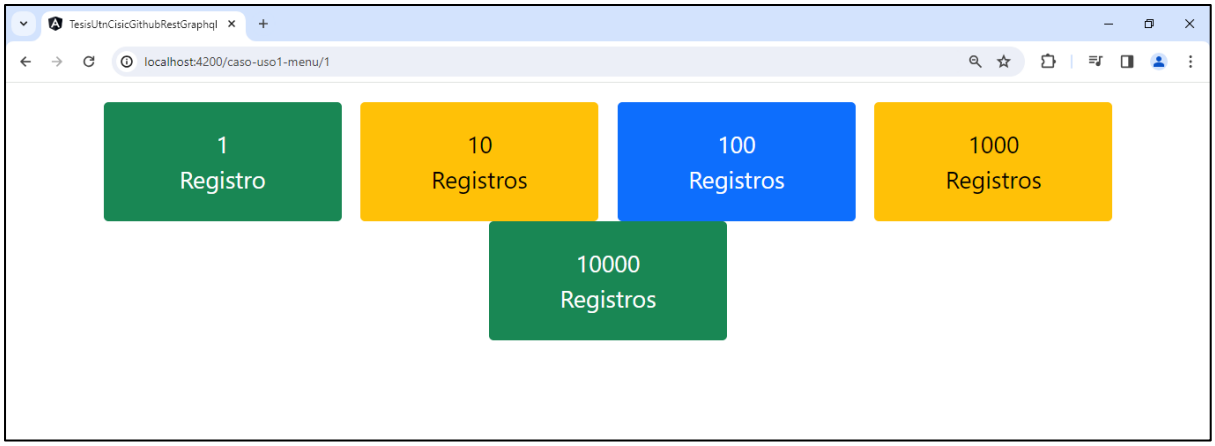


Figura 16. Pantalla para seleccionar una cantidad de registros específica

Pantalla de ejecución del nivel 1 de consumo

La Figura 17, muestra la pantalla que permite ejecutar el nivel 1 de consumo y la visualización de los tiempos de respuesta de la API REST y API GraphQL de GitHub. La ejecución se repite tres veces para posteriormente calcular el tiempo promedio de respuesta.

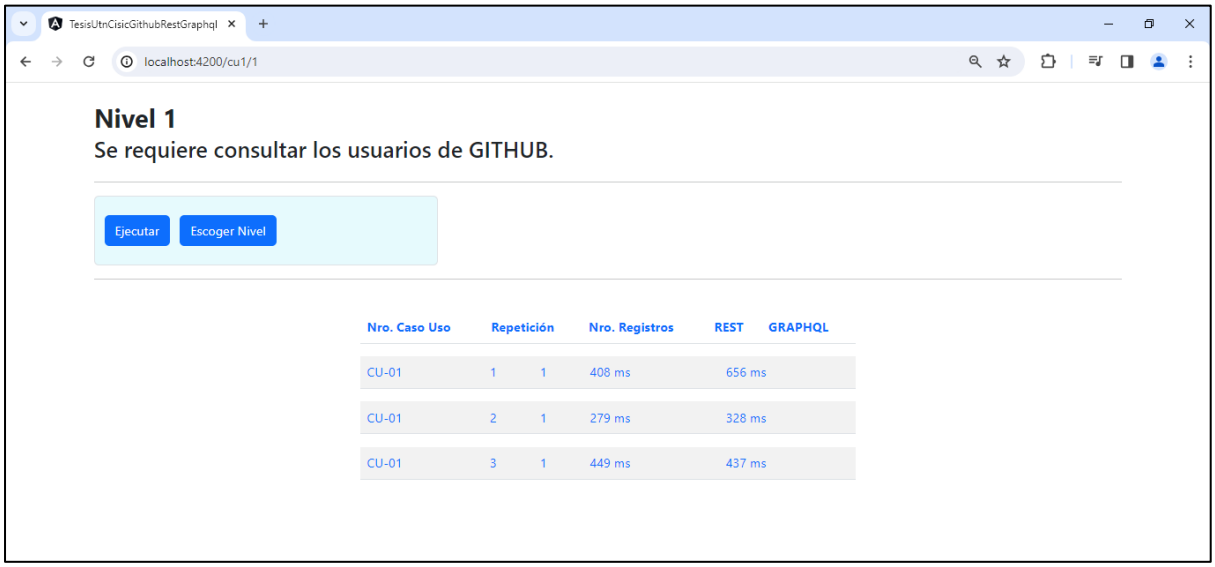


Figura 17. Pantalla de ejecución del nivel 1 de consumo

Retrospectiva del Sprint 2

- **Reunión de Retrospectiva**

  La reunión de retrospectiva se la realizó según la planificación, respetando las fechas acordadas.

  - **Fecha:** 19/01/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo

- **Objetivos:** Analizar las tareas desarrolladas y proponer un plan para mejorar

- **Plan de mejora**

  La Tabla 18, da a conocer los resultados obtenidos en la reunión de retrospectiva y las mejoras que se implementarán en el siguiente Sprint.

Tabla 18. Plan de mejora del Sprint 2

Aciertos	Errores/Problemas	Mejoras
Ambiente de Desarrollo configurado correctamente. Soporte para GraphQL en el lado del cliente configurado correctamente. Manera correcta de obtener el tiempo de respuesta de las APIs REST y GraphQL. Implementación/Codificación correcta de las dos soluciones (REST y GraphQL) para el nivel 1 de consumo. Pantallas que permiten visualizar los tiempos de respuesta del nivel 1 han sido desarrolladas correctamente.	Falta de información sobre GraphQL. Falta de conocimientos técnicos para configurar el soporte para GraphQL en el front-end. Poco conocimiento sobre el Framework Angular. Dificultad en la paginación de las respuestas de las peticiones, tanto en la API REST como en la API GraphQL. GitHub establece varios niveles de restricciones en cuanto a los límites de peticiones o llamadas que se puede realizar a sus APIs REST y GraphQL, lo cual dificulta las peticiones con las cantidades de registros más grandes.	Diseño de las pantallas (Interfaz de Usuario): - Selección del nivel de consumo. - Selección de la cantidad de registros. - Ejecución del nivel 1 de consumo.

2.3.3. Sprint 3

Planificación del Sprint 3

En este sprint se desarrollará una nueva pantalla que forma parte de la interfaz de usuario del cliente web, esta pantalla hará posible la ejecución del nivel 2 de consumo.

- **Reunión de Planificación**

  La reunión de planificación se realizó de acuerdo a la siguiente información:

  - **Fecha:** 20/01/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** planificar y definir el Sprint Backlog del Sprint 3

- **Sprint Backlog del Sprint 3**

  La Tabla 19, presenta el Sprint Backlog correspondiente al Sprint 3, en donde se registran las historias de usuario que se desarrollarán durante este Sprint. También se listan las tareas asignadas al Equipo de Desarrollo.

Tabla 19. Sprint Backlog del Sprint 3

Historia de Usuario	Denominación	Tareas	Tiempo (Horas)
H.U.07	Gestión del nivel 2	Diseñar y desarrollar la lógica de programación de la pantalla para ejecutar el nivel 2 de consumo y visualizar los tiempos de respuesta, en milisegundos, de las dos APIs de GitHub (REST y GraphQL)	25
Reuniones Scrum	Planificación		2
	Revisión		2
	Retrospectiva		2
		TOTAL HORAS	31

Revisión del Sprint 3

Como resultado del tercer sprint, se diseñó y desarrolló la pantalla que permite ejecutar el nivel 2 de consumo.

- **Reunión de Revisión**

  En este punto se procede a verificar el cumplimiento de la totalidad de las tareas definidas en el Product Backlog correspondiente al Sprint 3, en las fechas acordadas.

  - **Fecha:** 03/02/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** revisar el crecimiento del producto de software

- **Incremento del producto potencialmente entregable**

  Una vez que se ha realizado las tareas del Sprint 3, se da a conocer los avances más relevantes correspondientes a la Historia de Usuario H.U.09.

  Pantalla de ejecución del nivel 2 de consumo

  La Figura 18, muestra la pantalla que permite ejecutar el nivel 2 de consumo y la visualización de los tiempos de respuesta de la API REST y API GraphQL de GitHub. La ejecución se repite tres veces para posteriormente calcular el tiempo promedio de respuesta.

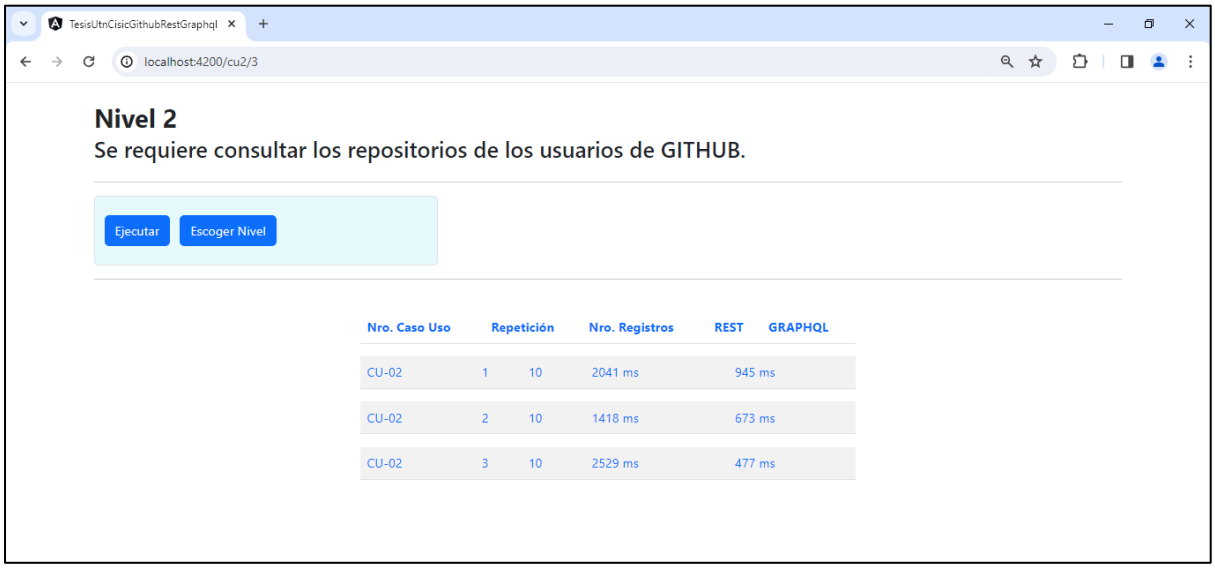


Figura 18. Pantalla de ejecución del nivel 2 de consumo

Retrospectiva del Sprint 3

- **Reunión de Retrospectiva**

  La reunión de retrospectiva se la realizó según la planificación, respetando las fechas acordadas.

  - **Fecha:** 03/02/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Resultado:** Analizar las tareas desarrolladas y proponer un plan para mejorar

- **Plan de mejora**

  La Tabla 20, da a conocer los resultados obtenidos en la reunión de retrospectiva y las mejoras que se implementarán en el siguiente Sprint.

Tabla 20. Plan de mejora del Sprint 3

Aciertos	Errores/Problemas	Mejoras
Manera correcta de obtener el tiempo de respuesta de las APIs REST y GraphQL. Implementación/Codificación correcta de las dos soluciones (REST y GraphQL) para el nivel 2 de consumo. Pantallas que permiten visualizar los tiempos de respuesta del nivel 2 han sido desarrolladas correctamente.	Dificultad para realizar la iteración de los endpoints REST. GitHub establece varios niveles de restricciones en cuanto a los límites de peticiones o llamadas que se puede realizar a sus APIs REST y GraphQL en un lapso de tiempo, lo cual dificulta el proceso.	Diseño de la pantalla de ejecución del nivel 2 de consumo.

2.3.4. Sprint 4

Planificación del Sprint 4

En este sprint se desarrollará una pantalla adicional que forma parte de la interfaz de usuario del cliente web, esta pantalla hará posible la ejecución del nivel 3 de consumo.

- **Reunión de Planificación**
  - **Fecha:** 04/02/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** planificar y definir el Sprint Backlog del Sprint 4

- **Sprint Backlog del Sprint 4**

  La Tabla 21, presenta el Sprint Backlog correspondiente al Sprint 4, en donde se registran las historias de usuario que se desarrollarán durante este Sprint. También se listan las tareas asignadas al Equipo de Desarrollo.

Tabla 21. Sprint Backlog del Sprint 4

Historia de Usuario	Denominación	Tareas	Tiempo (Horas)
H.U.08	Gestión del nivel 3	Diseñar y desarrollar la lógica de programación de la pantalla para ejecutar el nivel 3 de consumo y visualizar los	30

Continúa …

		tiempos de respuesta, en milisegundos, de las dos APIs de GitHub (REST y GraphQL)	
Reuniones Scrum	Planificación		2
	Revisión		2
	Retrospectiva		2
		TOTAL HORAS	36

Revisión del Sprint 4

Como resultado del cuarto sprint, se diseñó y desarrolló la pantalla que permite ejecutar el nivel 3 de consumo.

- **Reunión de Revisión**

  En este punto se procede a verificar el cumplimiento de la totalidad de las tareas definidas en el Product Backlog correspondiente al Sprint 4, en las fechas acordadas.

  - **Fecha:** 18/02/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** revisar el crecimiento del producto de software

- **Incremento del producto potencialmente entregable**

  Una vez que se ha realizado las tareas del Sprint 4, se da a conocer los avances más relevantes correspondientes a la Historia de Usuario H.U.10.

  Pantalla de ejecución del nivel 3 de consumo

  La Figura 19, muestra la pantalla que permite ejecutar el nivel 3 de consumo y la visualización de los tiempos de respuesta de la API REST y API GraphQL de GitHub. La ejecución se repite tres veces para posteriormente calcular el tiempo promedio de respuesta.

Figura 19. Pantalla de ejecución del nivel 3 de consumo

Retrospectiva del Sprint 4

- **Reunión de Retrospectiva**

  La reunión de retrospectiva se la realizó según la planificación, respetando las fechas acordadas.

  - **Fecha:** 18/02/2024
  - **Participantes:** Scrum Master, Product Owner y Equipo de Desarrollo
  - **Objetivo:** Analizar las tareas desarrolladas y proponer un plan para mejorar

- **Plan de mejora**

  La Tabla 22, da a conocer los resultados obtenidos en la reunión de retrospectiva y se dará por concluido el proyecto.

Tabla 22. Plan de mejora del Sprint 4

Aciertos	Errores/Problemas	Mejoras
Manera correcta de obtener el tiempo de respuesta de las APIs REST y GraphQL.	Dificultad para realizar la iteración de los endpoints REST.	Diseño de la pantalla de ejecución del nivel 3 de consumo.
Implementación/Codificación correcta de las dos soluciones (REST y GraphQL) para el nivel 3 de consumo. Continúa …	GitHub establece varios niveles de restricciones en cuanto a los límites de peticiones o llamadas que se puede realizar a sus APIs REST y GraphQL en un	

Pantallas que permiten visualizar los tiempos de respuesta del nivel 3 han sido desarrolladas correctamente.	lapso de tiempo, lo cual dificulta el proceso.	

2.4. Pruebas de Conformidad

Luego de haber culminado la implementación de todas las historias de usuario que constan en el Product Backlog de cada Sprint, el resultado es un producto de software terminado, en este caso la prueba de concepto de la funcionalidad y consumo de las APIs REST y GraphQL públicas de GitHub.

Se realizaron las pruebas de aceptación o de conformidad del software desarrollado, conjuntamente con el Product Owner del proyecto mediante la verificación de los requisitos y su funcionamiento. La Tabla 23, muestra las funcionalidades y su estado de aceptación por parte del Product Owner.

Tabla 23. Pruebas de conformidad

Código Historia de Usuario	Denominación	Funcionalidad	Aceptación	
			Sí	No
H.U.01	Gestión del token para acceder a los recursos	Obtención de un token que permita tener acceso a los recursos de la API REST y API GraphQL	X	
H.U.02	Gestión del recurso usuarios (users)	Obtención de la lista de usuarios de GitHub, tanto desde la API REST como desde la API GraphQL	X	
H.U.03	Gestión del recurso repositorios (repositories)	Obtención de la lista de repositorios de cada usuario de GitHub, tanto desde la API REST como desde la API GraphQL	X	
H.U.04	Gestión del recurso ramas (branches)	Obtención de la lista de ramas de cada uno de los repositorios de los usuarios de GitHub, tanto desde la API REST como desde la API GraphQL	X	
H.U.05	Gestión de menús de selección	Pantalla para la selección de un nivel de consumo específico	X	

Continúa …

		Pantalla para la selección de la cantidad de registros para el nivel de consumo previamente seleccionado	X
H.U.06	Gestión del nivel 1 de consumo	Pantalla que permite la ejecución del nivel 1 de consumo y también la visualización de los tiempos de respuesta, en milisegundos, de las APIs REST y GraphQL	X
H.U.07	Gestión del nivel 2 de consumo	Pantalla que permite la ejecución del nivel 2 de consumo y también la visualización de los tiempos de respuesta, en milisegundos, de las APIs REST y GraphQL	X
H.U.08	Gestión del nivel 3 de consumo	Pantalla que permite la ejecución del nivel 3 de consumo y también la visualización de los tiempos de respuesta, en milisegundos, de las APIs REST y GraphQL	X

CAPÍTULO 3

Resultados

En el desarrollo de este capítulo se emplea la guía de Wohlin denominada *Experimentación en la Ingeniería de Software,* para diseñar y presentar un experimento controlado, con la intención de abordar la pregunta de investigación (PI): ¿Cuál de las dos tecnologías sería más eficiente para el desarrollo de aplicaciones informáticas?

En este contexto, el experimento se configura para comparar las dos tecnologías en términos de eficiencia en el desarrollo de aplicaciones informáticas, centrándose netamente en su capacidad de consulta de datos.

El propósito es proporcionar una base empírica sólida para determinar cuál de las dos tecnologías, ya sea REST o GraphQL, ofrece ventajas significativas en términos de eficiencia, permitiendo tomar decisiones informadas sobre su uso en el desarrollo de aplicaciones informáticas.

Para evaluar el tiempo de respuesta se utilizó la norma ISO/IEC 25023, específicamente la característica denominada eficiencia del desempeño o rendimiento, todo esto dentro del contexto de la calidad del software.

3.1. Contexto del experimento

3.1.1. Objetivo del experimento

Comparar la eficiencia del rendimiento de la tecnología REST y la tecnología GraphQL a través del consumo de las APIs versión 3 (REST) y versión 4 (GraphQL) de GitHub en un entorno JavaScript local (localhost), en relación con la calidad del producto de software.

3.1.2. Elementos y tratamiento

El elemento que será objeto de investigación es la arquitectura de software, específicamente los servicios de las APIs. Los tratamientos para aplicar al elemento mencionado son:

- Arquitectura REST para desarrollar APIs.
- Arquitectura GraphQL para desarrollar APIs.

Como modelo de arquitectura REST se utilizará de manera directa la API-REST pública de GitHub y como modelo de arquitectura GraphQL, de igual manera, se utilizará la API-GraphQL pública de GitHub.

3.1.3. Variables

Dentro de este estudio se definen las variables independiente y dependiente. La variable independiente es la arquitectura mientras que la variable dependiente viene a ser la eficiencia del rendimiento. Para este caso particular, la arquitectura es la elección de la arquitectura REST o la arquitectura GraphQL puras.

La intención es evaluar y comparar las dos arquitecturas anteriormente mencionadas en el contexto de la eficiencia del rendimiento. La variable independiente se medirá y evaluará haciendo uso de las métricas definidas por la norma ISO/IEC 25023 de acuerdo a la característica de eficiencia del desempeño o rendimiento. La Tabla 24, muestra las variables definidas para este experimento.

Tabla 24. Variables definidas para el experimento

Variable independiente	Arquitectura REST para desarrollar APIs
	Arquitectura GraphQL para desarrollar APIs
Variable dependiente	Eficiencia del rendimiento

Para evaluar adecuadamente el rendimiento de un producto de software, es fundamental seguir estándares reconocidos que aseguren la validez y consistencia de las mediciones. Por lo tanto, se da a conocer la característica de *Eficiencia del Desempeño o Rendimiento* y, específicamente, la métrica de *Tiempo medio de respuesta,* definidas en la norma ISO/IEC 25023.

Tiempo medio de respuesta

Es el tiempo que toma completar un proceso, en este caso, es el tiempo promedio en el que se completa un proceso asíncrono. Dentro de este experimento se hizo uso de la Ecuación 1, con el fin de calcular el tiempo de respuesta de las peticiones enviadas tanto a la arquitectura REST, así como a la arquitectura GraphQL.

3.1.4. Hipótesis

En este apartado se establece la interrogante de investigación que se deriva de la pregunta de investigación inicial **PI.**

- **PI_1:** ¿Qué diferencia se aprecia en la eficiencia del rendimiento del software empleando una API desarrollada con arquitectura REST y empleando una API desarrollada con arquitectura GraphQL?

Ahora, con base en esta interrogante de investigación, lo siguiente es determinar las hipótesis correspondientes a este experimento.

- **H_0 (Hipótesis nula):** no existe una diferencia significativa en la eficiencia del desempeño o rendimiento entre la arquitectura REST y la arquitectura GraphQL. Es decir, no hay afectación al rendimiento del producto de software en términos de tiempo de respuesta.
- **H_1 (Hipótesis alternativa 1):** la arquitectura REST es mejor en lo que respecta a eficiencia del desempeño o rendimiento que la arquitectura GraphQL, en consecuencia, se espera que REST aporte una mayor calidad al producto de software que GraphQL.
- **H_2 (Hipótesis alternativa 2):** los servicios de las APIs desarrolladas con la arquitectura GraphQL muestran una mayor eficiencia en lo que respecta a rendimiento comparado con los servicios REST. Esto se traduce en que el producto de software final tendrá una mejor calidad utilizando APIs GraphQL.

3.1.5. Diseño

El presente diseño experimental está orientado a definir el ambiente adecuado para realizar la comparación de la eficiencia de la arquitectura REST y la arquitectura GraphQL. Dicho esto, se procederá a elaborar un laboratorio computacional que hará posible realizar un análisis riguroso del rendimiento de las dos arquitecturas en distintos escenarios.

Por lo tanto, se considerarán dos tareas experimentales donde las condiciones para consultar datos son similares. La tarea uno consiste en el consumo de datos de la API-REST de GitHub sin hacer uso de la memoria caché y la tarea dos consiste en el consumo de datos de la API-GraphQL de GitHub sin utilizar la memoria caché.

Se plantea tres casos de uso con niveles de complejidad diferentes. Cada uno de estos niveles es un escenario, en cada escenario se realiza tres repeticiones del consumo con cantidades de datos diferentes para comprobar la eficiencia de las APIs. Para las cantidades de datos se emplea una distribución adecuada debido a la no disponibilidad de data de las APIs públicas de GitHub.

Los casos de uso están vinculados a los niveles de consumo que se explicó en el apartado 2.2.4. Esto quiere decir que el caso de uso número uno emplea solamente un endpoint de la API-REST de GitHub, el caso de uso número dos emplea dos endpoints y finalmente el tercer caso de uso emplea tres endpoints.

En la Tabla 25, se detallan las distribuciones de datos de acuerdo a cada caso de uso.

Tabla 25. Distribución de datos según el caso de uso y la cantidad de endpoints

Caso de uso	Nro. de endpoints	Cantidad total de datos a consultar	Distribución de datos por endpoint
CU-01	1	1	1
		10	10
		100	100
		1000	10 iteraciones de 100
		10000	100 iteraciones de 100
CU-02	2	1	1 x 1
		10	3 x 3
		100	10 x 10
		1000	32 x 31
		10000	100 x 100
CU-03	3	1	1 x 1 x 1
		10	2 x 2 x 2
		100	5 x 5 x 4
		1000	10 x 10 x 10
		10000	22 x 21 x 21

Se emplearán las métricas establecidas en la norma ISO/IEC 25023 para capturar el tiempo de respuesta de cada caso de uso. En la Tabla 26, se da a conocer el diseño del experimento.

Tabla 26. Diseño para el experimento

Nombre del Caso de uso	Número de Repeticiones	Arquitectura REST	Arquitectura GraphQL
Caso Uso 1	3	**Registros:** 1, 10, 100, 1000, 10000	**Registros:** 1, 10, 100, 1000, 10000
Caso Uso 2	3	**Registros:** 1, 10, 100, 1000, 10000	**Registros:** 1, 10, 100, 1000, 10000
Caso Uso 3	3	**Registros:** 1, 10, 100, 1000, 10000	**Registros:** 1, 10, 100, 1000, 10000

3.1.6. Tareas para experimentación

En este estudio, como lo indica la Figura 20, se desarrolló un laboratorio computacional con las siguientes dos tareas de experimentación:

- Tarea de experimentación 1: consultar datos a la API-REST de GitHub.
- Tarea de experimentación 2: consultar datos a la API-GraphQL de GitHub.

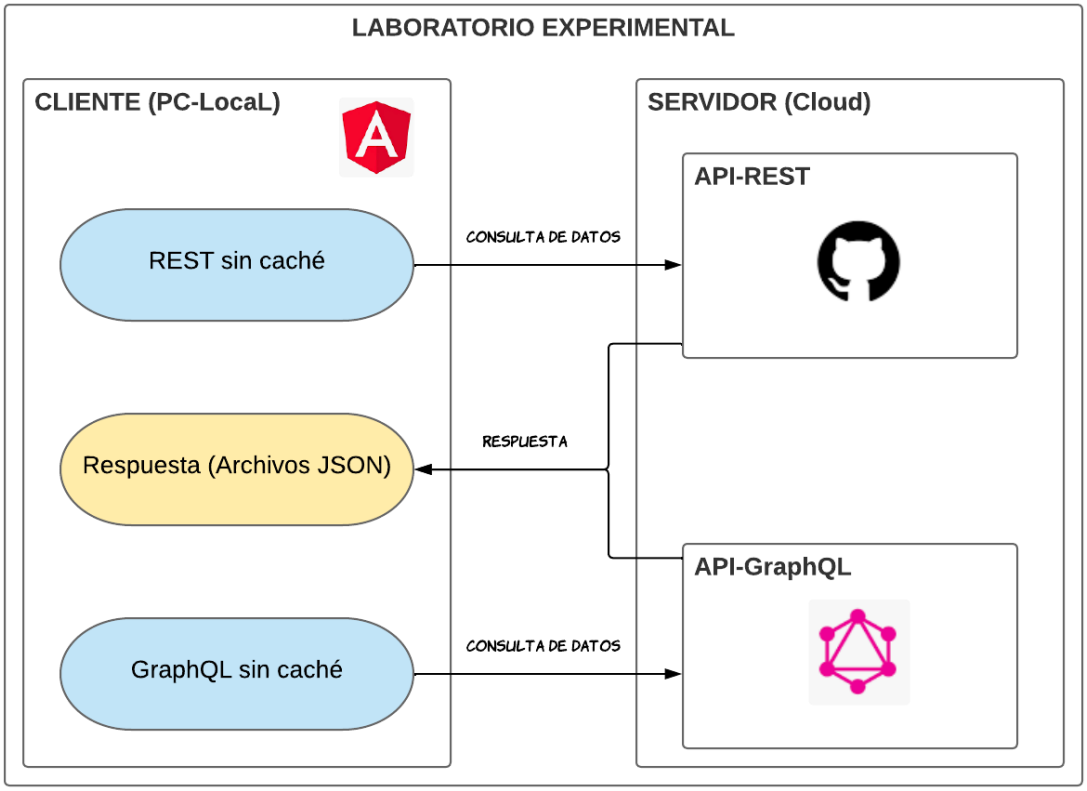


Figura 20. Arquitectura del laboratorio de experimentación

3.1.7. Herramientas

En este apartado se da a conocer los elementos del laboratorio computacional, frameworks y tecnologías que hacen parte de este entorno.

Características de la computadora local (PC-local):

- Sistema Operativo Windows 10 Pro 64-bit
- Procesador Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz
- Memoria RAM 8 GB (7,84 disponible)

Características del enlace a Internet:

- Velocidad carga: 110 Mbps

- Velocidad descarga: 120 Mbps

Herramientas de desarrollo:

- IDE (Entorno de Desarrollo Integrado): Visual Studio Code v1.87.1
- Lenguaje de programación: Angular (JavaScript, TypeScript)
- Manejador de paquetes: npm v8.19.3
- Librerías npm para el cliente Angular: @apollo/client v3.0.0, apollo-angular v5.0.2, graphql v16.0.0, graphql-tag v2.12.6, graphql-ws v5.5.5

Obtención y análisis de datos

- Microsoft Excel 2019

3.1.8. Obtención de datos

La Tabla 27, muestra el formato de datos del archivo de Microsoft Excel mismo que se utilizará para el registro de los resultados los cuales se obtendrán al realizar la ejecución del experimento expuesto en la Tabla 26.

Tabla 27. Formato para recolección de datos

Elemento	Descripción
Nro.	Identificador de la muestra
Caso de uso	Nombre del caso de uso en ejecución
Nro. Registros	Número de registros consultados en el caso de uso
Repetición	Identificador de la ejecución (puede ser 1, 2 o 3)
Arquitectura	REST o GraphQL
Tiempo	Tiempo en que la ejecución de cada caso de uso tarda en recibir una respuesta, se mide en milisegundos

3.2. Puesta en marcha del experimento

La ejecución de este experimento se la realizó de acuerdo a los pasos establecidos en el diseño experimental de la sección 3.1 para efectuar la comparación de la eficiencia de la arquitectura REST con la eficiencia de la arquitectura GraphQL.

3.2.1. Explicación de la ejecución

Al momento de ejecutar el experimento, existió la posibilidad de elegir un determinado caso de uso, para cada caso de uso fue posible elegir también una

cantidad de registros específica, donde se aplicó la distribución correspondiente de acuerdo con la Tabla 25. La ejecución de cada caso de uso por cada cantidad de registros se la realizó tres veces, es decir, hubo tres repeticiones, tanto para la arquitectura REST como para la arquitectura GraphQL. Este enfoque hizo posible la obtención de mediciones mucho más exactas y significativas de los tiempos de respuesta de las APIs.

Cabe mencionar que en cada una de las repeticiones se ejecutó la consulta correspondiente de acuerdo al número de registros seleccionado y se registró el tiempo de respuesta obtenido. Estas condiciones experimentales se las mantuvo en cada una de las repeticiones para de esta manera asegurar que los resultados sean consistentes.

Para una mejor compresión, se procede a detallar un ejemplo del proceso de consumo de datos de las APIs. Entonces, se comienza por seleccionar un caso de uso, para efectos del ejemplo será el caso de uso 1, mismo que involucra la obtención de los usuarios de GitHub. Ahora, se selecciona la cantidad de registros (usuarios de GitHub) que se desea obtener, puede ser 1, 10, 100, 1000 o 10000 registros.

Una vez que comienza el proceso, las tres repeticiones se realizan automáticamente, una después de que finaliza la anterior. En cada repetición se ejecuta la consulta que obtiene los usuarios de GitHub con la cantidad de registros especificada. Finalmente, se presenta en pantalla los tiempos de respuesta obtenidos en cada una de las tres repeticiones.

De esta manera, se obtiene tres veces el tiempo de respuesta para el caso de uso 1 por cada cantidad de registros especificada. Ahora, estos tres tiempos de respuesta permiten calcular promedios, dando paso así al tiempo medio de respuesta.

3.2.2. Recolección de datos

Una vez levantado el cliente web de manera local, es decir, en un entorno localhost y, además, habiéndose iniciado la ejecución del experimento de acuerdo al caso de uso y cantidad de registros seleccionados, se procedió a tomar nota de cada tiempo de respuesta, en milisegundos, en una hoja de cálculo de Microsoft Excel 2019 para realizar la tabulación y posteriormente el respectivo análisis.

En la Tabla 28, se da a conocer los datos que se recopilaron durante la ejecución del experimento, específicamente el caso de uso 1 para 1 registro en las arquitecturas REST y GraphQL.

Tabla 28. Tabulación tiempos de respuesta ejecución CU-01 para 1 registro

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-01	1	1	REST	715
2	CU-01	1	2	REST	311
3	CU-01	1	3	REST	422
4	CU-01	1	1	GraphQL	755
5	CU-01	1	2	GraphQL	315
6	CU-01	1	3	GraphQL	406

En la Tabla 29, se da a conocer los datos que se recopilaron durante la ejecución del experimento, específicamente el caso de uso 1 para 10 registros en las arquitecturas REST y GraphQL.

Tabla 29. Tabulación tiempos de respuesta ejecución CU-01 para 10 registros

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-01	10	1	REST	700
2	CU-01	10	2	REST	304
3	CU-01	10	3	REST	412
4	CU-01	10	1	GraphQL	756
5	CU-01	10	2	GraphQL	355
6	CU-01	10	3	GraphQL	443

En la Tabla 30, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 1 para 100 registros en las arquitecturas REST y GraphQL.

Tabla 30. Tabulación tiempos de respuesta ejecución CU-01 para 100 registros

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-01	100	1	REST	700
2	CU-01	100	2	REST	349
3	CU-01	100	3	REST	448
4	CU-01	100	1	GraphQL	1109
5	CU-01	100	2	GraphQL	101
6	CU-01	100	3	GraphQL	866

En la Tabla 31, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 1 para 1000 registros en las arquitecturas REST y GraphQL.

Tabla 31. Tabulación tiempos de respuesta ejecución CU-01 para 1000 registros

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-01	1000	1	REST	1398
2	CU-01	1000	2	REST	947
3	CU-01	1000	3	REST	1083
4	CU-01	1000	1	GraphQL	2107
5	CU-01	1000	2	GraphQL	2158
6	CU-01	1000	3	GraphQL	2475

En este punto es importante dar a conocer que no fue posible recopilar los 10000 registros para el caso de uso 1, ya que las APIs de GitHub no disponen de tal cantidad de datos.

En la Tabla 32, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 2 para 1 registro en las arquitecturas REST y GraphQL.

Tabla 32. Tabulación tiempos de respuesta ejecución CU-02 para 1 registro

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-02	1	1	REST	1072
2	CU-02	1	2	REST	738
3	CU-02	1	3	REST	818
4	CU-02	1	1	GraphQL	849
5	CU-02	1	2	GraphQL	416
6	CU-02	1	3	GraphQL	423

En la Tabla 33, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 2 para 10 registros en las arquitecturas REST y GraphQL.

Tabla 33. Tabulación tiempos de respuesta ejecución CU-02 para 10 registros

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-02	10	1	REST	1919
2	CU-02	10	2	REST	1475
3	CU-02	10	3	REST	1396
4	CU-02	10	1	GraphQL	2059
5	CU-02	10	2	GraphQL	299
6	CU-02	10	3	GraphQL	693

En la Tabla 34, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 2 para 100 registros en las arquitecturas REST y GraphQL.

Tabla 34. Tabulación tiempos de respuesta ejecución CU-02 para 100 registros

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-02	100	1	REST	6026
2	CU-02	100	2	REST	5027
3	CU-02	100	3	REST	4910
4	CU-02	100	1	GraphQL	4950
5	CU-02	100	2	GraphQL	1450
6	CU-02	100	3	GraphQL	2574

En este punto es importante dar a conocer que no fue posible recopilar los 1000 y 10000 registros para el caso de uso 2 ya que la API-GraphQL de GitHub superó los límites de acceso permitidos. En la API-REST fue posible obtener los 1000 registros, sin embargo, se decidió no incluir el tiempo registrado con la intención de mantener las mismas condiciones en las dos APIs.

En la Tabla 35, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 3 para 1 registro en las arquitecturas REST y GraphQL.

Tabla 35. Tabulación tiempos de respuesta ejecución CU-03 para 1 registro

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-03	1	1	REST	1454
2	CU-03	1	2	REST	1139
3	CU-03	1	3	REST	1143
4	CU-03	1	1	GraphQL	795
5	CU-03	1	2	GraphQL	472
6	CU-03	1	3	GraphQL	437

En la Tabla 36, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 3 para 10 registros en las arquitecturas REST y GraphQL.

Tabla 36. Tabulación tiempos de respuesta ejecución CU-03 para 10 registros

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-03	10	1	REST	2689
2	CU-03	10	2	REST	2457
3	CU-03	10	3	REST	2603
4	CU-03	10	1	GraphQL	880
5	CU-03	10	2	GraphQL	520
6	CU-03	10	3	GraphQL	564

En la Tabla 37, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 3 para 100 registros en las arquitecturas REST y GraphQL.

Tabla 37. Tabulación tiempos de respuesta ejecución CU-03 para 100 registros

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-03	100	1	REST	12306
2	CU-03	100	2	REST	12959
3	CU-03	100	3	REST	11310
4	CU-03	100	1	GraphQL	1881
5	CU-03	100	2	GraphQL	1070
6	CU-03	100	3	GraphQL	1394

En la Tabla 38, se da a conocer los datos que se recopilaron durante la ejecución del caso de uso 3 para 1000 registros en las arquitecturas REST y GraphQL.

Tabla 38. Tabulación tiempos de respuesta ejecución CU-03 para 1000 registros

Nro.	Caso de uso	Nro. Registros	Repetición	Arquitectura	Tiempo (ms)
1	CU-03	1000	1	REST	50645
2	CU-03	1000	2	REST	44087
3	CU-03	1000	3	REST	52959
4	CU-03	1000	1	GraphQL	5998
5	CU-03	1000	2	GraphQL	4642
6	CU-03	1000	3	GraphQL	3583

En este punto es importante dar a conocer que no fue posible recopilar los 10000 registros para el caso de uso 3 ya que la API-GraphQL de GitHub superó los límites de acceso permitidos. En la API-REST sí fue posible obtener los 10000 registros, sin embargo, se decidió no incluir el tiempo registrado con la intención de mantener las mismas condiciones en las dos APIs.

3.3. Análisis de resultados

3.3.1. Presentación de resultados

En esta sección se da a conocer los tiempos de respuesta finales obtenidos por cada caso de uso, es decir, el promedio de las tres repeticiones para cada número de registros, tanto de la arquitectura REST como de la arquitectura GraphQL.

En la Tabla 39, se presenta los tiempos de respuesta promedio, en base a la cantidad de registros, del caso de uso 1 en las dos arquitecturas.

Tabla 39. Tiempos de respuesta promedio CU-01 según el número de registros

Nro.	Caso de uso	Nro. Registros	Nro. Repeticiones	Tiempo (ms)	
				Arquitectura REST	Arquitectura GraphQL
1	CU-01	1	3	482,67	492
2	CU-01	10	3	472	518
3	CU-01	100	3	499	692
4	CU-01	1000	3	1142,67	2246,67

En la Tabla 40, se presenta los tiempos de respuesta promedio, en base a la cantidad de registros, del caso de uso 2 en las dos arquitecturas.

Tabla 40. Tiempos de respuesta promedio CU-02 según el número de registros

Nro.	Caso de uso	Nro. Registros	Nro. Repeticiones	Tiempo (ms)	
				Arquitectura REST	Arquitectura GraphQL
1	CU-02	1	3	876	562,67
2	CU-02	10	3	1596,67	1017
3	CU-02	100	3	5321	2991,33

En la Tabla 41, se presenta los tiempos de respuesta promedio, en base a la cantidad de registros, del caso de uso 3 en las dos arquitecturas.

Tabla 41. Tiempos de respuesta promedio CU-03 según el número de registros

Nro.	Caso de uso	Nro. Registros	Nro. Repeticiones	Tiempo (ms)	
				Arquitectura REST	Arquitectura GraphQL
1	CU-03	1	3	1245,33	568
2	CU-03	10	3	2583	654,67
3	CU-03	100	3	12191,67	1448,33
4	CU-03	1000	3	49230,33	4741

3.3.2. Análisis de eficiencia por cada caso de uso

Ahora, se procede a realizar un análisis de la eficiencia del rendimiento de las arquitecturas REST y GraphQL tomando como base los casos de uso que han sido planteados. Para llevar a cabo este análisis, se procedió a convertir la unidad de

tiempo de milisegundos a segundos con el fin de evidenciar de mejor manera las diferencias.

Caso de uso 1

Este primer caso de uso se encarga de consultar los usuarios de GitHub (Nivel 1). Se calculó el tiempo de respuesta promedio de cada arquitectura, obteniendo los siguientes valores:

- Arquitectura REST: 0,65 segundos
- Arquitectura GraphQL: 0,99 segundos

En la Figura 21, se muestra los datos de forma gráfica.

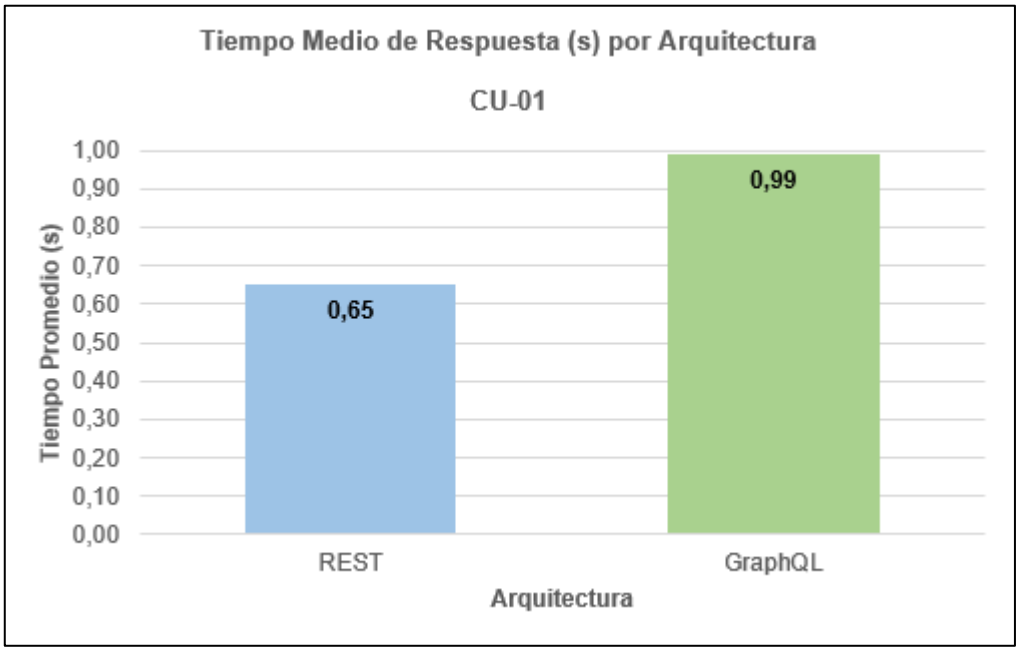


Figura 21. Tiempos de respuesta promedio CU-01 por arquitectura

Como lo da a conocer la Figura 21, el tiempo de respuesta promedio que corresponde a la arquitectura REST es menor que el tiempo de respuesta promedio de la arquitectura GraphQL, en consecuencia, REST es 1,52 veces más eficiente, así lo indica la Tabla 42.

Tabla 42. Eficiencia del caso de uso 1

Arquitectura	**Tiempo Promedio (s)**
REST	0,65
GraphQL	0,99
REST es 1,52 veces más eficiente que GraphQL	

Caso de uso 2

El segundo caso de uso se encarga de consultar los repositorios de los usuarios de GitHub (Nivel 2). Se calculó el tiempo de respuesta promedio de cada arquitectura, obteniendo los siguientes valores:

- Arquitectura REST: 2,60 segundos
- Arquitectura GraphQL: 1,52 segundos

En la Figura 22, se muestra los datos de forma gráfica.

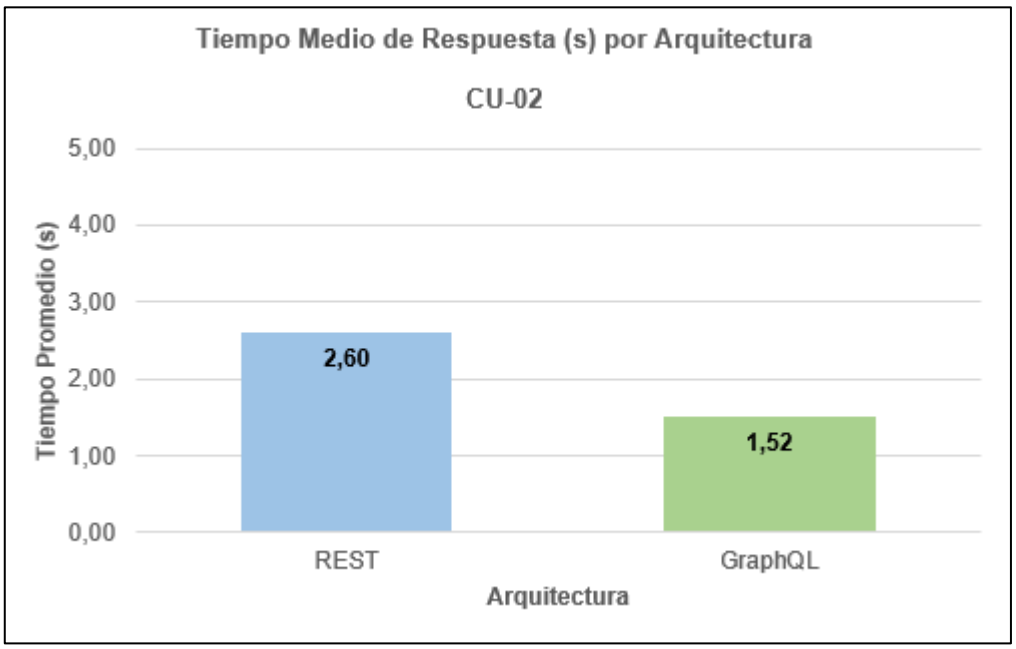


Figura 22. Tiempos de respuesta promedio CU-02 por arquitectura

Como lo da a conocer la Figura 22, el tiempo de respuesta promedio que corresponde a la arquitectura REST es mayor que el tiempo de respuesta promedio de la arquitectura GraphQL, en consecuencia, GraphQL es 1,71 veces más eficiente, así lo indica la Tabla 43.

Tabla 43. Eficiencia del caso de uso 2

Arquitectura	**Tiempo Promedio (s)**
REST	2,60
GraphQL	1,52
GraphQL es 1,71 veces más eficiente que REST	

Caso de uso 3

El tercer caso de uso se encarga de consultar las ramas de los repositorios de los usuarios de GitHub (Nivel 3). Se calculó el tiempo de respuesta promedio de cada arquitectura, obteniendo los siguientes valores:

- Arquitectura REST: 16,31 segundos
- Arquitectura GraphQL: 1,85 segundos

En la Figura 23, se muestra los datos de forma gráfica.

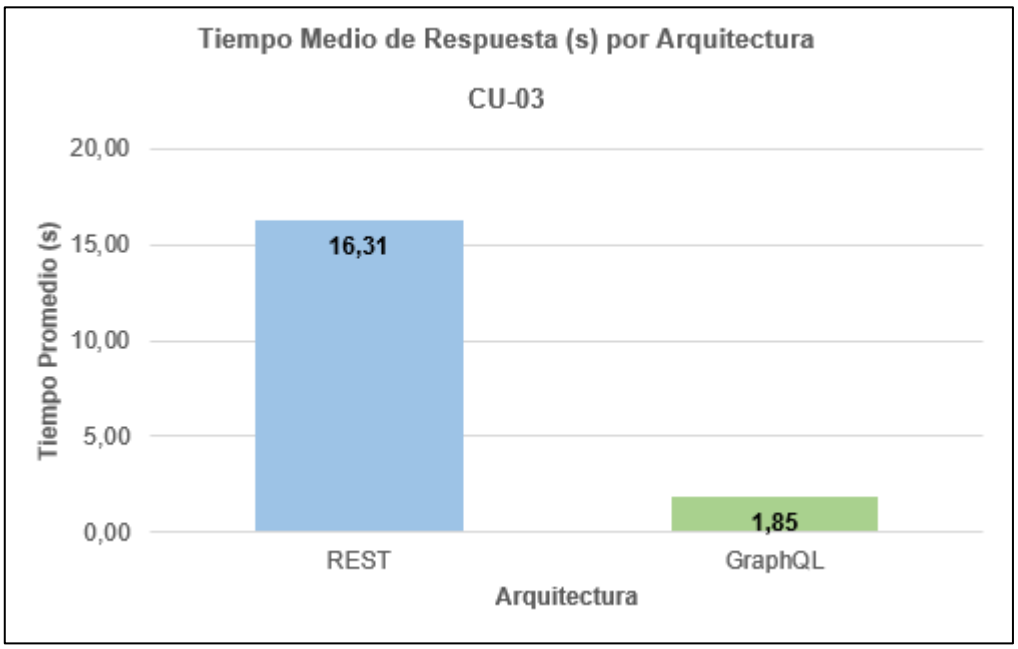


Figura 23. Tiempos de respuesta promedio CU-03 por arquitectura

Como lo da a conocer la Figura 23, el tiempo de respuesta promedio que corresponde a la arquitectura REST es mucho mayor que el tiempo de respuesta promedio de la arquitectura GraphQL, en consecuencia, GraphQL es 8,82 veces más eficiente, así lo indica la Tabla 44.

Tabla 44. Eficiencia del caso de uso 3

Arquitectura	Tiempo Promedio (s)
REST	16,31
GraphQL	1,85
GraphQL es 8,82 veces más eficiente que REST	

3.3.3. Resumen y promedios totales por arquitectura

Una vez que se han dado a conocer los resultados que se han obtenido de cada uno de los casos de uso, la Figura 24, expone un resumen grupal de los promedios obtenidos de los casos de uso por arquitectura.

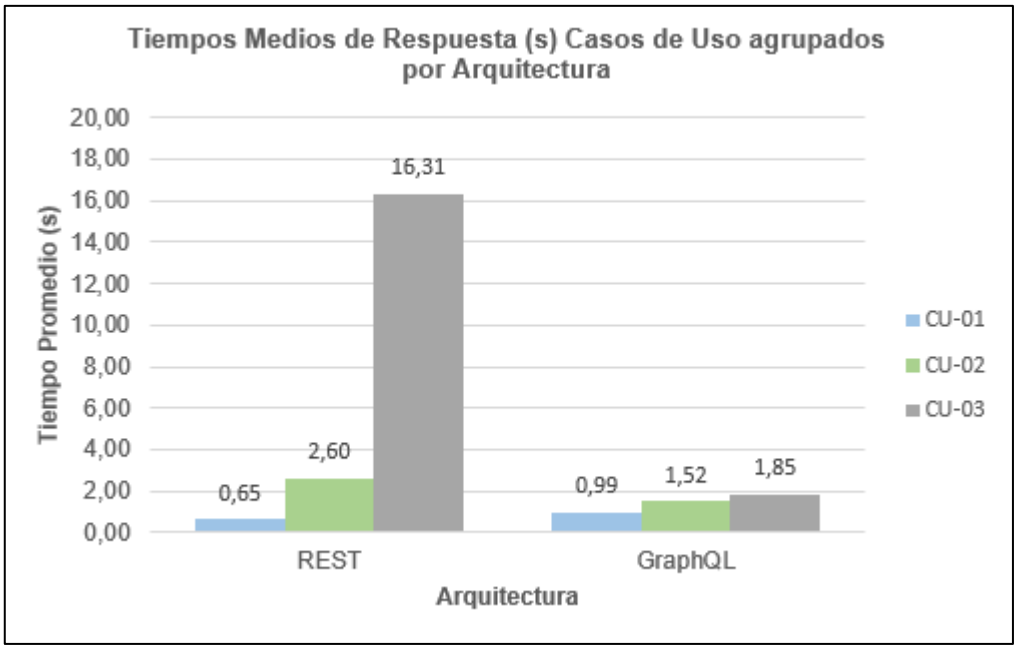


Figura 24. Resumen tiempos promedio todos los casos de uso por arquitectura

En la Tabla 45, se da a conocer los resultados globales de los tiempos de respuesta por cada arquitectura, donde se pone en evidencia que la media de REST es de 6,52 segundos y en el caso de GraphQL la media es de 1,45 segundos.

Tabla 45. Tiempos de respuesta promedio por arquitectura

Arquitectura	Tiempo Promedio (s)
REST	6,52
GraphQL	1,45

3.3.4. Análisis de impactos

Luego del análisis realizado, se estableció que el tiempo medio de respuesta correspondiente a la arquitectura GraphQL es 4,49 veces más eficiente que la arquitectura REST. En la Tabla 46, se da a conocer los promedios globales de cada arquitectura y la eficiencia.

Tabla 46. Eficiencia final

Arquitectura	Caso Uso 1	Caso Uso 2	Caso Uso 3	Tiempo Promedio (s)
REST	0,65	2,60	16,31	6,52
GraphQL	0,99	1,52	1,85	1,45
GraphQL es 4,49 veces más eficiente que REST				

CONCLUSIONES

A través de una revisión en diferentes fuentes bibliográficas, se elaboró un marco teórico sólido que hace posible un entendimiento pleno de los temas que intervienen en la realización del experimento. En este sentido, esta base teórica aporta el contexto necesario sobre las diferentes arquitecturas que existen para desarrollar APIs, las alternativas para consumir los servicios que proveen, la guía para realizar el estudio experimental y el mecanismo para llevar a cabo el posterior análisis de los resultados obtenidos.

Se implementó un cliente web (prueba de concepto) que realiza el consumo de las dos APIs públicas de GitHub, siendo estas la versión 3 y la versión 4, cuyas arquitecturas son REST y GraphQL, respectivamente. Para esto, se hizo uso del framework para front-end Angular y diversas librerías que permiten consumir las dos APIs. El proceso de desarrollo de software se enmarcó en la metodología ágil SCRUM, la cual, aporto con buenas prácticas para la construcción del cliente web dentro de este estudio experimental.

La evaluación de la eficiencia y rendimiento de las arquitecturas REST y GraphQL se la realizó mediante un experimento computacional basado en la metodología de Wohlin, centrándose netamente en el indicador conocido como tiempo de respuesta. En este contexto, se planteó la interrogante de investigación (PI) de la siguiente manera: ¿Cuál de las dos tecnologías sería más eficiente para el desarrollo de aplicaciones informáticas? Para dar la respectiva respuesta se diseñaron e implementaron tres casos de uso de diferente nivel de complejidad, los mismos para las dos arquitecturas. Estos tres casos de uso permitieron realizar la comparativa de la eficiencia de las arquitecturas REST y GraphQL, empleando la métrica “tiempo medio de respuesta” establecida por la norma ISO/IEC 25023. Finalmente, luego de ejecutar el experimento computacional se realizó un análisis de los resultados obtenidos, donde se evidencia que la API-GraphQL es 4.49 veces más eficiente que la API-REST. En consecuencia, se admite la segunda hipótesis alternativa de este estudio donde se manifiesta que “los servicios de las APIs desarrolladas con la arquitectura GraphQL muestran una mayor eficiencia en términos de rendimiento en comparación con los servicios REST”.

RECOMENDACIONES

En el desarrollo de estudios experimentales como el que aquí se presenta, se recomienda hacer una investigación teórica profunda en fuentes de información confiables de manera que haya una comprensión clara de todos los temas y procedimientos que se vayan abordando progresivamente en la investigación.

Se recomienda realizar el mismo experimento, pero con el diseño e implementación de un cliente móvil. De esta manera se tendría una perspectiva más amplia sobre el rendimiento de las dos arquitecturas en cuestión y se podría analizar su comportamiento en diferentes entornos tecnológicos.

Se sugiere contemplar la posibilidad de implementar este experimento, con la variante de acceder a campos específicos en lugar de recibir toda la data como respuesta desde las APIs públicas. Es decir, la idea sería consultar y mostrar en pantalla solamente ciertos campos que sean de interés en lugar de recibir toda la data desde las APIs. Esto permitiría analizar el comportamiento de las dos arquitecturas en un contexto de especificidad y optimización.

GLOSARIO DE TÉRMINOS

ABREVIATURAS

- **REST**
  Representational State Transfer
- **API**
  Application Programming Interface
- **ISO/IEC**
  International Organization for Standardization / International Electrotechnical Commission
- **SOAP**
  Simple Object Access Protocol
- **URI**
  Uniform Resource Identifier
- **XML**
  eXtensible Markup Language
- **SOA**
  Service-oriented architecture
- **HTTP**
  Hypertext Transfer Protocol
- **JSON**
  JavaScript Object Notation

DEFINICIONES

- **GITHUB**
  Es una plataforma para desarrollo colaborativo para alojar proyectos de software haciendo uso del sistema de control de versiones Git.

- **SCRUM**

  Es un marco de trabajo para desarrollo ágil de software. Se trata de aplicar un conjunto de buenas prácticas para trabajar de manera colaborativa, en equipo y así obtener el mejor resultado posible en los proyectos.

- **Scopus**

  Es una herramienta para evaluar el rendimiento de una revista científica.

- **IEEE Explore**

  Es una base de datos para investigación que permite descubrir y acceder a artículos de revistas, actas de congresos, normas técnicas y otros materiales relacionados con la informática, la ingeniería eléctrica y electrónica, y áreas relacionadas.

- **DBLP**

  Es un sitio web que cuenta con un extenso repositorio de artículos relacionados con las ciencias de la computación.

- **Google Scholar**

  Es un motor de búsqueda diseñado para localizar documentos de tipo académico como tesis, artículos, documentos de congresos, patentes, libros y resúmenes.

- **Framework**

  Es un marco de trabajo o entorno de trabajo que consiste en un conjunto estandarizado de criterios, prácticas y conceptos para abordar un tipo de problemática particular. En el contexto del desarrollo de software, permite definir la estructura de los proyectos y proporciona las herramientas necesarias que se usarán como bloques de construcción.

- **PHP**

  Es un lenguaje de programación interpretado que se ejecuta en el lado del servidor y es de propósito general, especialmente adecuado para el desarrollo web.

- **Backend**

  Dentro del desarrollo de software, el backend se encarga de funciones como procesar formularios, almacenar y recuperar datos de una base de datos, gestionar la seguridad del sitio y autenticar usuarios.

- **Angular**

  Es un framework de código abierto para aplicaciones web, empleado para crear y mantener aplicaciones web de una sola página.

- **NoSQL**

  Significa “no solo SQL”, hace referencia a las bases de datos no relacionales que usan un formato no tabular para almacenar datos, en lugar de hacerlo en tablas relacionales basadas en reglas, como las bases de datos relacionales.

- **Endpoint**

  En el contexto de las APIs, un endpoint es una dirección de una API, o bien un backend que se encarga de dar respuesta a una petición. Se basa en HTTP y sirve para obtener y enviar datos e información.

- **Stakeholders**

  En el ámbito del desarrollo de software, un stakeholder es cualquier individuo, entidad o grupo que tiene un interés o participa en un proyecto de software. Esto abarca a desarrolladores, diseñadores, clientes, patrocinadores, usuarios finales, gerentes de proyecto, equipos de control de calidad y otros actores relevantes.

- **Tester**

  Los “probadores de software” diseñan y ejecutan pruebas para verificar el correcto funcionamiento del software.

- **Sprint**

  Es un intervalo de tiempo corto y definido durante el cual un equipo de scrum trabaja para cumplir con una cantidad específica de trabajo.

- **Release**

  En el contexto del desarrollo de software, hace referencia a un nuevo lanzamiento o una nueva publicación, es decir, luego de implementar una

nueva funcionalidad o corregir un error en un producto de software, se publica una nueva versión para probarla o para desplegarla en producción.

- **Caché**
  Es una memoria que está entre la unidad central de procesamiento (CPU) y la memoria de acceso aleatorio (RAM) que permite acelerar el intercambio de datos.

BIBLIOGRAFÍA

Aqsha, Fauziyah, U., Tee, K., & Kaburuan, E. R. (2019). Planning catering business model for E-commerce SOA based: AU-OR! *International Journal of Recent Technology and Engineering*, *8*(3), 8161–8166. https://doi.org/10.35940/ijrte.C6114.098319

Cai, D., Zhao, Y., Deng, J., & Zhang, D. (2019). Organizational Level Cloud Diagnostics System of Complex Electromechanical System Based on SOA. *IOP Conference Series: Materials Science and Engineering*, *626*(1). https://doi.org/10.1088/1757-899X/626/1/012001

Fundación Sadosky. (2014). *Prueba de concepto (PoC – por sus siglas en inglés) | Área de Vinculación Tecnológica - Fundación Sadosky*. http://www.fundacionsadosky.org.ar/avt/glossary/prueba-de-concepto-poc-por-sus-siglas-en-ingles/

Ganesh, N., & Narayanan, R. C. (2019). Challenges faced in the enterprise resource planning material management section when transitioning towards agile software development. *International Journal of Engineering and Advanced Technology*, *8*(6), 3472–3475. https://doi.org/10.35940/ijeat.F9521.088619

GitHub. (n.d.). *GitHub GraphQL API documentation*. GitHub GraphQL API Documentation. Retrieved August 11, 2024, from https://docs.github.com/en/graphql

GitHub. (2022). *GitHub REST API documentation*. GitHub REST API Documentation. https://docs.github.com/en/rest?apiVersion=2022-11-28

Gómez-Sierra, C. J., & López-Bustamante, B. J. (2019). Development of a web application for consultation and online payments of technological services in higher education institutions. *IOP Conference Series: Materials Science and Engineering*, *519*(1). https://doi.org/10.1088/1757-899X/519/1/012032

Guo, Y., Deng, F., & Yang, X. (2018). Design and Implementation of Real-Time Management System Architecture based on GraphQL. *IOP Conference Series: Materials Science and Engineering*, *466*(1). https://doi.org/10.1088/1757-899X/466/1/012015

INTECO. (2020). *INTE/ISO/IEC 25023:2020. Ingeniería de sistemas de Software - Requisits y evaluación de calidad de sistemas y del software (SQuaRE) - Medición de calidad de sistemas y productos Software*.

ISO/IEC_25022. (2016). *International Standard ISO/IEC 25022*.

Kalliamvakou, E., Singer, L., Gousios, G., German, D. M., Blincoe, K., & Damian, D. (2014). The promises and perils of mining GitHub. *11th Working*

Conference on Mining Software Repositories, MSR 2014 - Proceedings, 92–101. https://doi.org/10.1145/2597073.2597074

Krohn, R., & Weninger, T. (2019). Library adoption in public software repositories. *Journal of Big Data*, *6*(1). https://doi.org/10.1186/s40537-019-0201-8

Lara, M. (2021). *Modelo de una Interfaz de Programación de Aplicaciones REST utilizando GO basado en normas y principios de seguridad de la información y aplicaciones web*. http://dspace.espoch.edu.ec/handle/123456789/14676

López, C. (2023). *Desarrollo de un envoltorio del API-REST del servicio de multimedia Spotify utilizando el lenguaje de consultas GraphQL, para mejorar la eficiencia del consumo de los datos basado en la característica de eficiencia de la norma ISO/IEC 25023*. http://repositorio.utn.edu.ec/handle/123456789/14798

Malla, J. (2014). *Servicios Web*. https://www.researchgate.net/publication/262688345_Servicios_Web

Mombach, T., & Valente, M. T. (2018). *GitHub REST API vs GHTorrent vs GitHub Archive: A Comparative Study*. https://developer.github.com/v3/

Naciones Unidas. (2019). *Objetivos y metas de desarrollo sostenible - Desarrollo Sostenible*. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

Nicula, D., & Ghimii, S. S. (2019). Command and Control vs self Management. *IOP Conference Series: Materials Science and Engineering*, *514*(1). https://doi.org/10.1088/1757-899X/514/1/012039

Nogatz, F., & Seipel, D. (2017). Implementing GraphQL as a query language for deductive databases in SWI-Prolog using DCGs, quasi quotations, and dicts. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, *234*, 42–56. https://doi.org/10.4204/EPTCS.234.4

Priyatna, F., Chaves-Fraga, D., Alobaid, A., & Corcho, O. (2019). morph-GraphQL: GraphQL Servers Generation from R2RML Mappings (S). *Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering*, *2019*, 291–296. https://doi.org/10.18293/seke2019-055

Quinaluiza, A. (2018). *Interfaz de programación de aplicaciones para la generación automática de procedimientos almacenados en MySQL*. http://repositorio.uta.edu.ec/jspui/handle/123456789/27820

Russell, P. H., Johnson, R. L., Ananthan, S., Harnke, B., & Carlson, N. E. (2018). A large-scale analysis of bioinformatics code on GitHub. *PLoS ONE*, *13*(10), 1–19. https://doi.org/10.1371/journal.pone.0205898

Scrum Manager. (2014). *Gestión de proyectos Scrum Manager*. http://www.safecreative.org/work/1404220636268

Soni, A., & Ranga, V. (2019). API Features Individualizing of Web Services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering*, *8*(9S), 664–671. https://doi.org/10.35940/ijitee.i1107.0789s19

Tarkowska, A., Carvalho-Silva, D., Cook, C. E., Turner, E., Finn, R. D., & Yates, A. D. (2018). Eleven quick tips to build a usable REST API for life sciences. *PLoS Computational Biology*, *14*(12), 1–8. https://doi.org/10.1371/journal.pcbi.1006542

Tee, K., Nurjannah, R., Lee, D., & Kaburuan, E. R. (2019). Design of waste management service system in Indonesia based on service oriented architecture (SOA): Go-waste. *International Journal of Recent Technology and Engineering*, *8*(3), 5257–5261. https://doi.org/10.35940/ijrte.C5909.098319

Ulrich, H., Kern, J., Tas, D., Kock-Schoppenhauer, A. K., Ückert, F., Ingenerf, J., & Lablans, M. (2019). QL 4 MDR: A GraphQL query language for ISO 11179-based metadata repositories. *BMC Medical Informatics and Decision Making*, *19*(1), 1–7. https://doi.org/10.1186/s12911-019-0794-z

Véliz, A. (2023). *Desarrollo de un envoltorio de la API-REST de live streaming Twitch utilizando GraphQL, para mejorar la eficiencia del consumo de datos, basado en la norma ISO/IEC 25023*. http://repositorio.utn.edu.ec/handle/123456789/14797

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. *Experimentation in Software Engineering*, *9783642290442*, 1–236. https://doi.org/10.1007/978-3-642-29044-2/COVER

ANEXOS

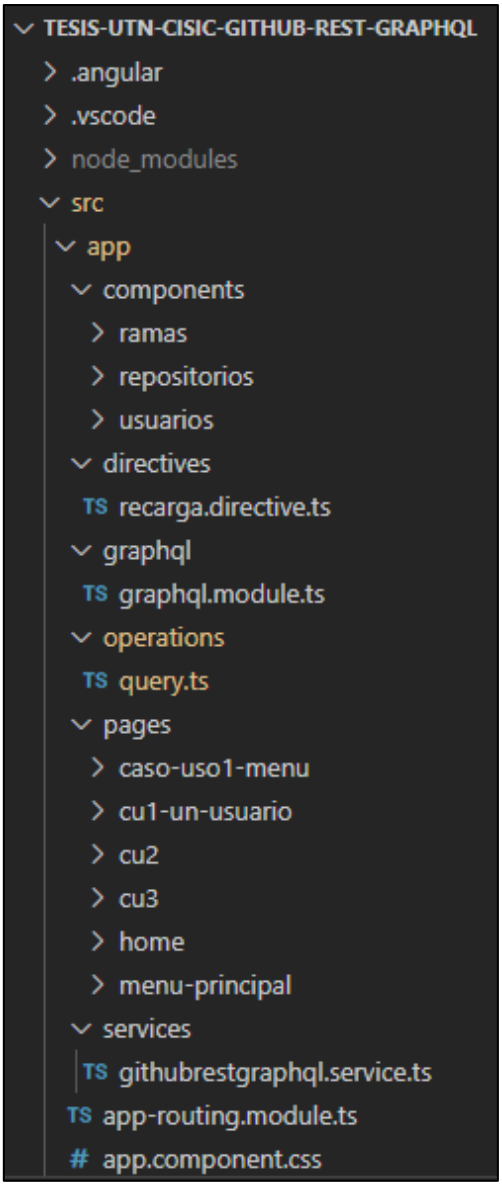
ANEXO A

Código fuente del proyecto.

Proyecto	Repositorio
Cliente web (Prueba de concepto)	https://github.com/jibarrakru/tesis-utn-cisic-github-rest-graphql.git

ANEXO B

Estructura de carpetas del cliente web Angular.

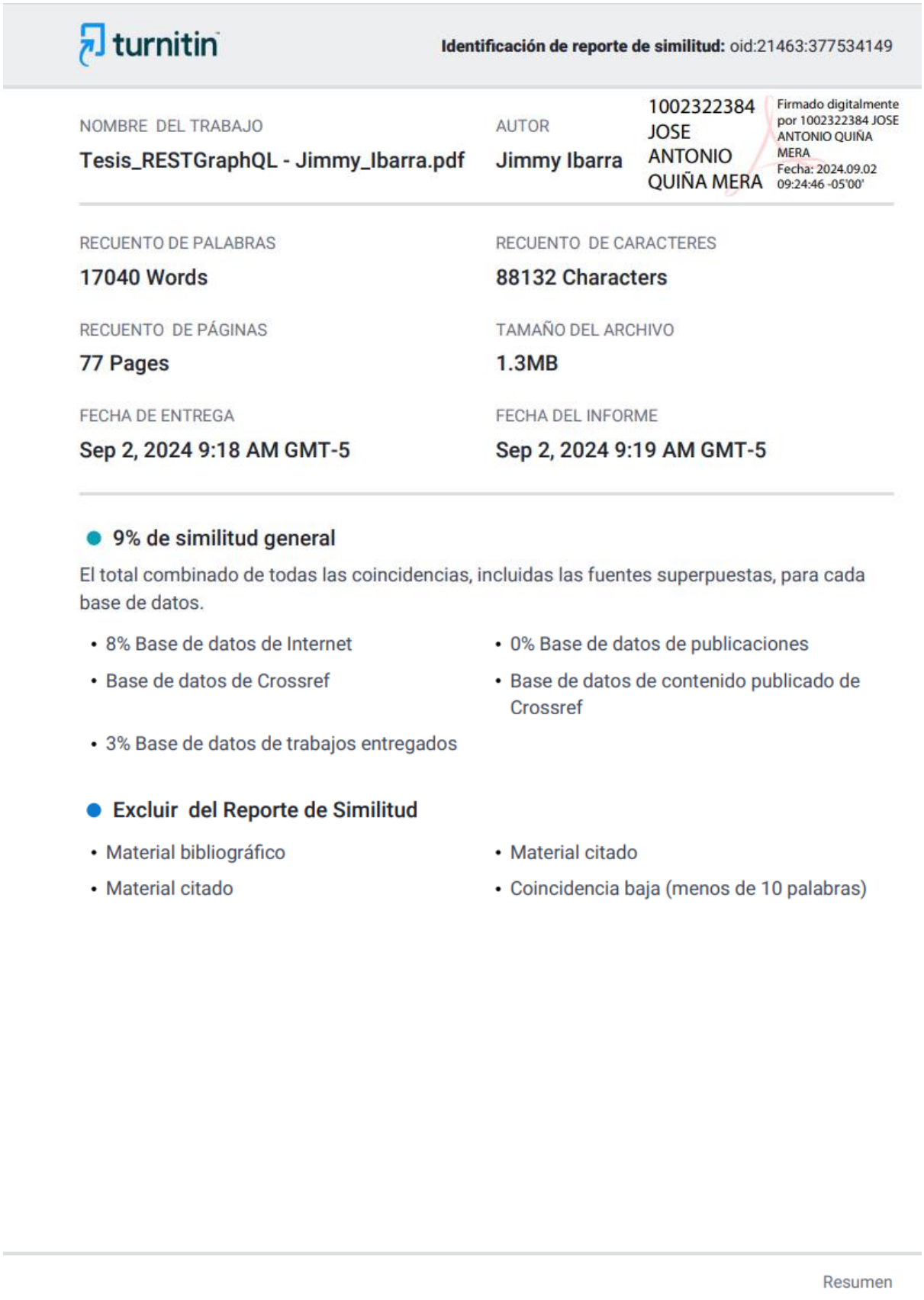

TESIS-UTN-CISIC-GITHUB-REST-GRAPHQL
.angular
.vscode
node_modules
src
app
components
ramas
repositorios
usuarios
directives
recarga.directive.ts
graphql
graphql.module.ts
operations
query.ts
pages
caso-uso1-menu
cu1-un-usuario
cu2
cu3
home
menu-principal
services
githubrestgraphql.service.ts
app-routing.module.ts
app.component.css

ANEXO C

Librerías externas utilizadas en el cliente web Angular.

ANEXO D

Reporte Turnitin.



turnitin

Identificación de reporte de similitud: oid:21463:377534149

NOMBRE DEL TRABAJO
Tesis_RESTGraphQL - Jimmy_Ibarra.pdf

AUTOR
Jimmy Ibarra

1002322384 JOSE ANTONIO QUIÑA MERA
Firmado digitalmente por 1002322384 JOSE ANTONIO QUIÑA MERA
Fecha: 2024.09.02 09:24:46 -05'00'

RECUENTO DE PALABRAS
17040 Words

RECUENTO DE CARACTERES
88132 Characters

RECUENTO DE PÁGINAS
77 Pages

TAMAÑO DEL ARCHIVO
1.3MB

FECHA DE ENTREGA
Sep 2, 2024 9:18 AM GMT-5

FECHA DEL INFORME
Sep 2, 2024 9:19 AM GMT-5

9% de similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base de datos.

- 8% Base de datos de Internet
- Base de datos de Crossref
- 3% Base de datos de trabajos entregados
- 0% Base de datos de publicaciones
- Base de datos de contenido publicado de Crossref

Excluir del Reporte de Similitud

- Material bibliográfico
- Material citado
- Material citado
- Coincidencia baja (menos de 10 palabras)

Resumen