UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Software

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

Trabajo de grado previo a la obtención del título de Ingeniero de Software presentado ante la ilustre Universidad Técnica del Norte.

Autor:

Carlos Arturo López Sánchez

Director:

PhD. José Antonio Quiña Mera

Ibarra – Ecuador

2023

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 IDENTIDAD:	1758553729		
APELLIDOS Y NOMBRES:	López Sánchez Carlos Arturo		
DIRECCIÓN:	IBARRA, Antonio José de Sucre 28-55 y Río Morona		
EMAIL:	calopezs@utn.edu.ec		
TELÉFONO FIJO:	N/A	**TELÉFONO MÓVIL:**	0969598740

DATOS DE LA OBRA	
TÍTULO:	DESARROLLO DE UN ENVOLTORIO DEL API-REST DEL SERVICIO DE MULTIMEDIA SPOTIFY UTILIZANDO EL LENGUAJE DE CONSULTAS GRAPHQL, PARA MEJORAR LA EFICIENCA DEL CONSUMO DE LOS DATOS BASADO EN LA CARACTERÍSTICA DE EFICIENCIA DE LA NORMA ISO/IEC 25023
AUTOR (ES):	CARLOS ARTURO LÓPEZ SÁNCHEZ
FECHA:	07/09/2023
PROGRAMA:	PREGRADO
TITULO POR EL QUE OPTA:	INGENIERO DE SOFTWARE
DIRECTOR:	PhD. Jose Antonio Quiña Mera
ASESOR 1:	Msc. Xavier Mauricio Rea Peñafiel

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 07 días del mes de septiembre de 2023

EL AUTOR:

ESTUDIANTE

Carlos Arturo López Sánchez

C.I: 1758553729

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN DIRECTOR

Por medio del presente yo PhD. Antonio Quiña, certifico que el Sr. **CARLOS ARTURO LÓPEZ SÁNCHEZ** portador de la cédula de ciudadanía Nro. 1758553729, ha trabajado en el desarrollo del proyecto de grado: **"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",** previo a la obtención del título de Ingeniero en Software 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
Nombre de reconocimiento (DN): 1.3.6.1.4.1.37442.10.4=1002322384, o=QUIÑA MERA JOSE ANTONIO, ou=Certificado Persona Natural RUC EC (FIRMA), givenName=JOSE ANTONIO, 2.5.4.97=1002322384001, c=EC, serialNumber=1002322384, sn=QUIÑA MERA, cn=1002322384 JOSE ANTONIO QUIÑA MERA, email=antonio_quinia@hotmail.com, title=REPRESENTANTE LEGAL, 2.5.4.13=Reg. Pub.:A Inscripción: A, Tomo A, Libro 1, Folio A, Hoja A Cargo:REPRESENTANTE LEGAL Notario:A Núm. Protocolo:A, st=IMBABURA, l=LA DOLOROSA DEL PRIORATO

PhD. Antonio Quiña

DIRECTOR DE TRABAJO DE GRADO

DEDICATORIA

Dedico este trabajo a mi familia, quienes han sido una fuente inagotable de apoyo y enseñanzas desde mi infancia. Agradezco a mi papá por inculcarme la valentía y el espíritu de colaboración, y a mi mamá por su paciencia y bondad, que siempre han sido un ejemplo a seguir en mi vida. Su amor y guía han sido fundamentales en mi crecimiento personal y en la realización de este proyecto.

Asimismo, quiero dedicar este trabajo a mi hermana, quien ha sido una fuente constante de inspiración para mí. Su inteligencia y habilidades siempre me han dejado asombrado, y estoy convencido de que logrará cosas aún más extraordinarias en el futuro. Esta dedicatoria es un recordatorio de que cuando nos apasionamos por algo, podemos alcanzar metas increíbles. Creo en ti y sé que seguirás cosechando éxitos en tu camino.

Por último, quiero dedicar este trabajo a mi familia en Colombia, especialmente a mis queridos tíos, primos y abuelos. A pesar de la distancia, siempre estuvieron presentes en mi vida, brindándome su apoyo incondicional, cariño y aliento en mi crecimiento personal y profesional. Su amor y compañía han sido fundamentales para alcanzar mis metas y enfrentar los desafíos.

Carlos Arturo López Sánchez

AGRADECIMIENTOS

Quiero expresar mi profundo agradecimiento a mis padres por estar conmigo en cada paso de este proceso. Su apoyo incondicional y guía han sido fundamentales para alcanzar este logro. Sin ellos, no habría llegado tan lejos, y estoy infinitamente agradecido por su paciencia, dedicación y entrega hacia mí. Este logro también es suyo, y estoy lleno de gratitud por todo lo que han hecho por mí.

Asimismo, quiero extender mi agradecimiento a la Universidad Técnica del Norte y a sus excelentes docentes. Gracias a esta institución y a la enseñanza impartida en el campo de la Ingeniería de Software, he adquirido conocimientos valiosos no solo para mi carrera profesional, sino también para la vida. En especial, quiero agradecer a mi tutor de tesis, el PhD. Antonio Quiña, por su dedicación y acompañamiento durante este trabajo y otros proyectos en los que hemos tenido éxito.

También quiero expresar mi gratitud a mis tutores de vida, los filósofos estoicos Séneca, Marco Aurelio y Epicteto. Sus enseñanzas han sido una fuente de inspiración y fortaleza para mejorar mi carácter y encontrar paz en mi mente. Desde que me sumergí en la filosofía estoica, mi vida ha experimentado una transformación positiva y significativa.

Por último, no puedo dejar de agradecer a mis amigos. Su compañerismo, risas y apoyo han hecho que mi experiencia universitaria sea más enriquecedora y divertida. Formar parte de un grupo unido y solidario ha sido un regalo inestimable, y estoy agradecido por el hermoso vínculo que hemos forjado juntos. Gracias a todos ellos, esta etapa de mi vida ha sido aún más significativa y memorable.

Carlos Arturo López Sánchez

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Spotify, el popular servicio de transmisión de música, podcasts y vídeos digitales, se ha convertido en una plataforma líder que ofrece acceso instantáneo a una extensa biblioteca de millones de canciones y contenido creado por artistas y creadores de todo el mundo. Su API-REST permite a los desarrolladores de aplicaciones de terceros acceder y utilizar las funcionalidades y datos proporcionados por Spotify. Sin embargo, la arquitectura REST presenta desafíos en cuanto a la personalización y eficiencia en el consumo de datos, lo que puede resultar en un rendimiento deficiente de las aplicaciones y una experiencia subóptima para los usuarios. Para abordar esta problemática, esta tesis se enfoca en desarrollar un envoltorio del API-REST de Spotify utilizando el lenguaje de consultas GraphQL. El estudio incluye una revisión sistemática de la literatura para establecer un marco teórico sobre los envoltorios GraphQL en el contexto de la API de Spotify. Además, se empleará la metodología ágil Scrum para el desarrollo del envoltorio y se llevará a cabo un experimento basado en la guía de Wohlin y la norma ISO/IEC 25023 para evaluar su eficiencia. Los resultados obtenidos demostraron que el envoltorio implementado es altamente eficiente en términos de tiempo de respuesta, con mejoras del 99,91% con caché y 56,64% sin caché, validando así la utilización de una arquitectura híbrida REST/GraphQL como una mejora significativa en la calidad del software.

ABSTRACT

Spotify, the popular music streaming, podcast, and digital video service, has become a leading platform that offers instant access to an extensive library of millions of songs and content created by artists and creators from around the world. Its API-REST allows third-party application developers to access and utilize Spotify's functionalities and data. However, the REST architecture presents challenges regarding personalization and data consumption efficiency, which can lead to poor application performance and a suboptimal user experience. To address this issue, this thesis focuses on developing a wrapper for Spotify's API-REST using the GraphQL query language. The study includes a systematic literature review to establish a theoretical framework for GraphQL wrappers in the context of Spotify's API. Additionally, the agile Scrum methodology will be employed for the wrapper's development, and an experiment based on Wohlin's guide and ISO/IEC 25023 standard will be conducted to evaluate its efficiency. The results showed that the implemented wrapper is highly efficient in terms of response time, with 99.91% improvement with cache and 56.64% improvement without cache, thus validating the use of a REST/GraphQL hybrid architecture as a significant improvement in software quality.

INTRODUCCIÓN

Tema

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.

Planteamiento del problema

La historia de origen de GraphQL se remonta al cambio de la industria a dispositivos móviles. Esto fue en un momento en que la estrategia móvil de Facebook no estaba funcionando debido a problemas relacionados con el alto uso de la red. La API existente no se diseñó para permitir que los desarrolladores expusieran una experiencia rica similar a la de un News Feed de noticias en dispositivos móviles. En 2012, Facebook decidió que necesitaba crear una nueva API de News Feed para crear la aplicación móvil de Facebook. (*What Is GraphQL: History, Components, and Ecosystem* | *by Brenda Clark* | *Level Up Coding*, n.d.).

Spotify es un servicio de música, podcasts y vídeos digitales que da accesos a millones de canciones y a otro contenido de creadores de todo el mundo(*¿Qué Es Spotify? - Spotify*, n.d.) Por medio de un servicio REST, Spotify permite utilizar sus funcionalidades para aplicaciones de terceros; el consumo de la API presenta los problemas y limitaciones de la arquitectura REST, lo que genera que los desarrolladores tengan inconvenientes con las peticiones, ya que no son personalizadas y no devuelven los datos exactos que se requiera (Vogel et al., 2018).

El servicio multimedia Spotify, posee un API-REST que puede ser consumida por la comunidad de desarrolladores interesados en este servicio. No obstante, Spotify para desarrolladores presenta varios inconvenientes reportados que afectan a la eficiencia del consumo de la API-REST, tales como: mal interpretación de los parametros de algunos endpoints, no recibir resultados precisos, y límites de velocidad en el endpoint de vista previa de mp3 (*Web API* | *Spotify for Developers*, n.d.)

Objetivos

Objetivo General

Desarrollar 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.

Objetivos Específicos

- Investigar sobre las tecnologías GraphQL y API-REST.
- Desarrollar un envoltorio GraphQL de la API-REST de Spotify que permita el consumo de las funcionalidades de los contenidos multimedia.
- Evaluar el envoltorio a través de un experimento basado en la guía de Wohlin, que permita comparar la característica de eficiencia del envoltorio GraphQL desarrollado y la API-REST de Spotify, bajo la norma ISO/IEC 25023.
- Analizar los resultados obtenidos del proyecto.

Investigar sobre las tecnologías GraphQL y API-REST.

Desarrollar un envoltorio GraphQL de la API-REST de Spotify que permita el consumo de las funcionalidades de los contenidos multimedia.

Evaluar el envoltorio a través de un experimento basado en la guía de Wohlin, que permita comparar la característica de eficiencia del envoltorio GraphQL desarrollado y la API-REST de Spotify, bajo la norma ISO/IEC 25023.

Analizar los resultados obtenidos del proyecto.

Alcance

El actual trabajo busca desarrollar un envoltorio del API-REST de Spotify con el lenguaje de consultas GraphQL, donde además se plantea comparar la característica de eficiencia frente a la API-REST original, considerando la norma ISO/IEC 25023.

El envoltorio va dirigido a distintas comunidades de desarrollo e investigación, como por ejemplo la comunidad de desarrolladores de Spotify, comunidades de desarrollo de APIs, la

comunidad académica y de investigación de la Universidad Técnica del Norte; y por último, a los desarrolladores que utilizan la API-REST de Spotify y buscan una mejor eficiencia en el consumo.

Servicios de Spotify que se implementarán al proyecto.

Conexión de cuenta de usuario de Spotify.

- Obtener listas de reproducción por usuario.
- Obtener canciones de una lista de reproducción.
- Obtener canciones por álbum.
- Obtener canción por el identificador.
- Obtener el artista por el identificador.

Cada uno de los servicios de Spotify fue seleccionado pensando en las grandes cantidades de datos que se pueda obtener. Dado que, para la realización del experimento, es necesario observar variaciones significativas con respecto al tiempo de respuesta que se demora en obtener los datos del envoltorio y de la API-REST original.

En la etapa de creación del marco teórico se realizará una investigación que permitirá conocer conceptos importantes sobre las arquitecturas orientadas a microservicios donde, además, se podrá comprender sus principales diferencias y el beneficio del uso del lenguaje de consultas GraphQL como una alternativa de las API-REST.

Después de la creación del marco teórico, se procederá a desarrollar un prototipo del envoltorio GraphQL con algunas de las funcionalidades del API-REST de Spotify. Para la comparación se hará un experimento tomando en cuenta la guía de Wohlin, donde se contrastará y medirá el tiempo medio de respuesta de ida y vuelta de los datos en la API-REST original y el envoltorio API GraphQL.

A continuación, en la Figura 1 se presenta la arquitectura y en la Figura 2 el proceso experimental con sus fases a utilizar.

Figura 1

Arquitectura del proyecto

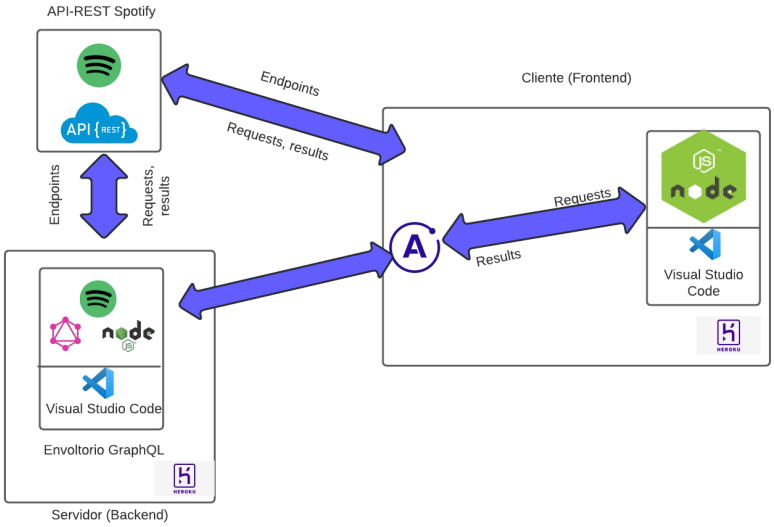
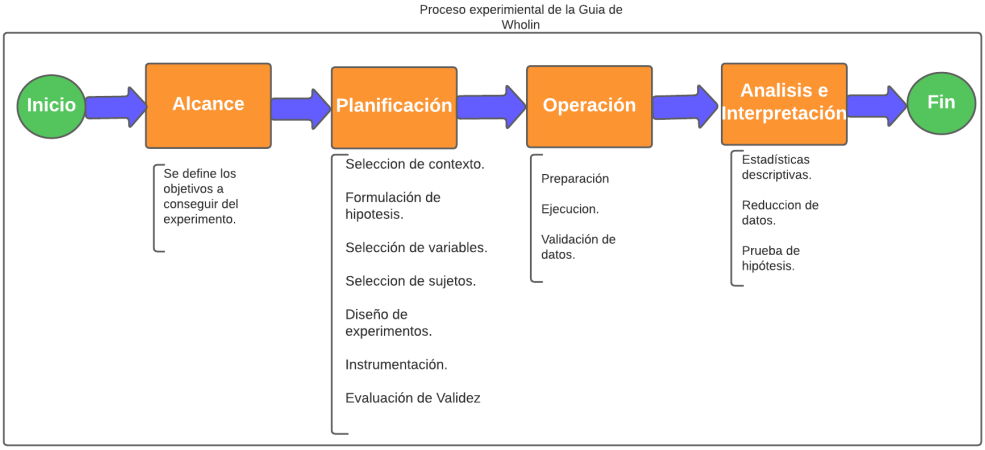


Figura 2

Proceso experimental



Metodología

El proceso metodológico por seguir en el presente proyecto ayudará a cumplir con los objetivos establecidos previamente. Para el primer objetivo se dispone de la creación de un marco teórico que se realizara por medio de una revisión sistemática de la literatura. Las revisiones sistemáticas de literatura se identifican como un procedimiento riguroso de relevamiento conceptual que permiten identificar publicaciones notables y detectar brechas de conocimiento, tensiones entre la teoría y la práctica o controversias de resultados empíricos

dentro de la academia (Manterola et al., 2013). Esta revisión de la literatura ayudará a entender sobre la creación de envoltorios GraphQL sobre otras APIs; además, conocer las principales ventajas y diferencias con las API-REST tradicionales.

Para cumplir con el objetivo número dos, se tendrá en cuenta la metodología de desarrollo ágil Scrum. Las metodologías ágiles se centran en las mejores prácticas de programación y su integración en el proceso de desarrollo. Son enfoques que definen la gestión disciplinada del desarrollo de software: la agilidad recomienda el uso de un método iterativo e incremental para desarrollar sistemas de software que realmente satisfagan las necesidades del cliente (Chantit & Essebaa, 2021).

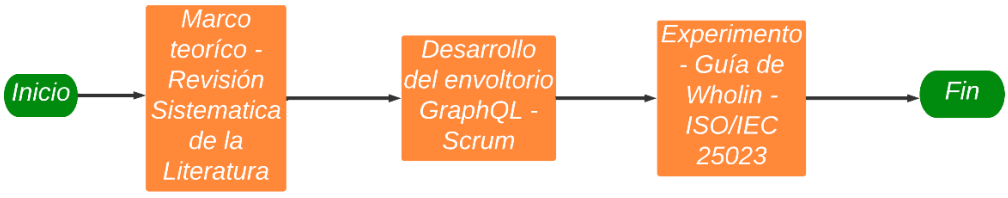
En el objetivo número tres, se evaluará por medio de un experimento la característica de eficiencia del envoltorio, donde, se tomará en cuenta la guía de Wholin sobre la experimentación en la ingeniería de software. Wholin (2012), plantea: La experimentación es fundamental para cualquier esfuerzo científico y de ingeniería. Comprender una disciplina implica construir modelos de los diversos elementos de la disciplina, por ejemplo, los objetos en el dominio, los procesos utilizados para manipular esos objetos, la relación y los objetos.

Junto con la guía de Wholin también se considerará la norma ISO/IEC 25023.Esta norma define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software(*ISO/IEC 2502n – División de Medición de Calidad*, n.d.).

A continuación, en la Figura 3 se presenta el proceso metodológico a seguir.

Figura 3

Proceso metodológico



Justificación

Este trabajo está enfocado en los Objetivos de Desarrollo Sostenible (ODS), en el cual se toman en consideración los siguientes:

Educación de calidad (Objetivo 4): Busca garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje durante toda la vida para todos (*Objetivo 4: Educación de Calidad – ODS Territorio Ecuador*, n.d.).

Industria, innovación e infraestructura (Objetivo 9): Los avances tecnológicos son esenciales para encontrar soluciones permanentes a los desafíos económicos y ambientales, al igual que la oferta de nuevos empleos y la promoción de eficiencia energética (*Objetivo 9: Industria, Innovación e Infraestructura | El PNUD En Ecuador*, n.d.).

En el ámbito del Plan Nacional de Desarrollo Toda una Vida (PNTV), el presente trabajo apoya a "A6. Crear programas de formación técnica y tecnológica pertinentes al territorio, con un enfoque de igualdad de oportunidades." Según el Plan de Creación de Oportunidades 2021-2025 (*Plan de Creación de Oportunidades 2021-2025 – Secretaría Nacional de Planificación*, n.d.).

Justificación Institucional.- El presente trabajo busca apoyar a la Universidad Técnica del norte en la visión de constituirse como un instituto superior de investigación que sea líder en la generación de ciencia y tecnología e innovación con indicadores de calidad nacional e internacional (*DIRECCIÓN DE INVESTIGACIÓN – Universidad Técnica Del Norte*, n.d.).

Justificación Tecnológica. - Desarrollar un envoltorio GraphQL de la API-REST del servicio de multimedia Spotify que permita a los desarolladores de Spotify y de APIs en general, obtener los datos de las consultas de forma personalizada, exacta y eficiente.

Justificación Académica. - La comunidad académica y de investigación de la Universidad Técnica del Norte, obtendrá un estudio de comparación evaluado con métricas sobre el envoltorio GraphQL desarrollada y la API-REST de Spotify

CAPÍTULO 1: Marco Teórico

1.1 Fundamentos de arquitecturas orientada a microservicios.

La arquitectura de software se refiere a la estructura y organización de un sistema de software, incluyendo sus componentes, módulos, interfaces y sus relaciones. La arquitectura de software es importante porque permite una gestión más efectiva de la complejidad del software, lo que facilita el desarrollo, mantenimiento y evolución del sistema (Pressman Roger, 2002; Schmidt, 2000).

Existen diferentes enfoques y estilos de arquitectura de software, algunos de los cuales son ampliamente utilizados en la industria. A continuación, se describe el enfoque orientado a microservicios, que servirá como base para el caso de estudio de la presente tesis.

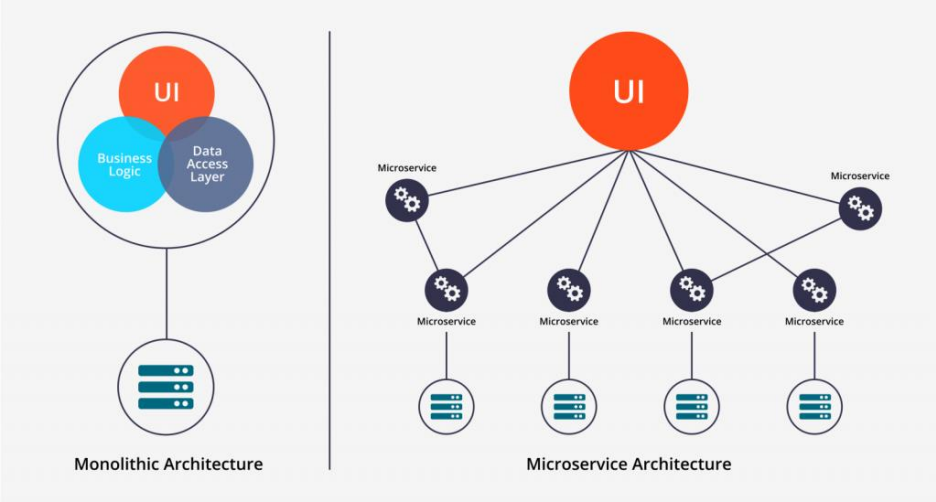
1.1.1. Definición.

La arquitectura orientada a microservicios se ha convertido en una tendencia popular en el desarrollo de software debido a su capacidad para mejorar el rendimiento y la escalabilidad de las aplicaciones (López, 2017). Su enfoque se basa en la creación de servicios independientes, altamente cohesivos y débilmente acoplados, lo que permite la fácil integración de nuestros servicios y la actualización de los existentes sin afectar a otros componentes del sistema (González & Rafael, 2021). Esta arquitectura cambia el enfoque tradicional de comunicación entre servicios, lo que permite una mayor flexibilidad y eficiencia en la gestión de los servicios (Nebel Andrés, 2018).

Además, la implementación de microservicios permite a los desarrolladores trabajar de forma más autónoma y en paralelo, lo que acelera el proceso de desarrollo y mejora la calidad del software en general. En la Figura 4 se puede apreciar cómo trabaja la arquitectura orientada a microservicios frente a la arquitectura monolítica.

Figura 4

Arquitectura monolítica frente a una arquitectura orientada a microservicios



Nota. Adaptado de una arquitectura tradicional a una arquitectura microservicios, por Hiberus, 2021, HiberusBlog (https://www.hiberus.com).

1.1.2. Características

Alfonso y Contreras (2018) proponen las siguientes características de una arquitectura orientada a microservicios:

- **Pequeño y enfocado en hacer una sola cosa**
  Los microservicios intentan aislar la especialidad de una lógica de negocio específica en un solo lugar, lo que permite disminuir la tentación de hacer crecer mucho el componente.

- **Organizado en torno a la lógica de negocio**
  El enfoque a microservicios divide a las organizaciones alrededor de la lógica de negocio. Esta organización es interdisciplinar pues mezcla la UI, lógica de negocio y base de datos, lo que conlleva a que el equipo tenga diferentes habilidades.

- **Autonomía**
  Los microservicios son una entidad separada. Deben ser desplegados como un servicio aislado en infraestructuras que se enfocan en ofrecer plataformas como servicio (PAAS, por sus siglas en inglés) o deben estar en su propio sistema. Todas las comunicaciones entre servicios son llamados de red para reforzar la separación y evitar los peligros de estrecho acoplamiento.

- **Diversidad de tecnología**

  Dado que cada microservicio es una unidad de despliegue independiente, se cuenta con una considerable libertad en las elecciones de tecnología que se pueden tomar. Los microservicios pueden usar diferentes lenguajes, librerías o almacenamiento de datos, con lo que la elección se basa en cuál es el problema por resolver, y con base en ello escoger la mejor herramienta. No se debe sobrepasar el conocimiento que puede manejar la organización, pues puede ocurrir que el que exista demasiada diversidad tecnológica se convierta en un problema más que en una solución.

1.1.3. Beneficios

La escalabilidad es uno de los principales beneficios de la arquitectura de microservicios. Según el artículo de Dragoni et al. (2016), esta arquitectura permite la escalabilidad horizontal, lo que significa que se pueden agregar más instancias de un servicio para manejar un mayor volumen de tráfico. Además, los servicios se pueden agregar o eliminar según sea necesario sin afectar la disponibilidad de otros servicios en la aplicación.

Otro beneficio de la arquitectura de microservicios es la flexibilidad. Como señala el artículo de Dragoni et al. (2016), cada servicio de puede desarrollar, implementar y escalar de forma independiente, lo que permite a los equipos de desarrollo trabajar de forma independiente en diferentes servicios sin afectar al resto de la aplicación. De esta manera, las empresas pueden iterar y evolucionar cada servicio por separado para responder rápidamente a las necesidades del mercado y de los clientes.

Finalmente, la arquitectura de microservicios también puede mejorar la eficiencia de las aplicaciones. Según el libro de Newman (2015), los servicios se pueden escalar independientemente y se pueden implementar en diferentes plataformas y tecnologías según sea necesario. Esto permite a las empresas aprovechar las tecnologías más adecuadas para cada servicio, lo que puede mejorar el rendimiento y reducir los costos de la infraestructura.

1.1.4. Problemas de la arquitectura orientada a microservicios

Aunque la arquitectura orientada a microservicios puede ofrecer una serie de servicios, también presenta desafíos significativos. Tal como indica Nebel Andrés (2018), los desafíos que presenta esta arquitectura son los siguientes:

- **Complejidad operacional**

Esto se debe a la necesidad de gestionar y mantener una cantidad considerable de microservicios, lo cual motiva a las empresas a la aplicación de DevOps. También, impulsa al uso de estrategias relacionadas como despliegue y entrega continuos. Dichos enfoques poseen una curva pronunciada de aprendizaje, debido al cambio que implica organizativo y también al uso de tecnología nueva que apoya a los procesos.

- **Consistencia Eventual**
  La arquitectura de microservicios también posee problemas relacionados a la consistencia eventual, ya que usa este enfoque para la persistencia descentralizada. Así, es necesario mitigar los problemas de datos desactualizados y controlar posibles decisiones que tome el código en base a ellas.

De la misma manera el artículo de Nebel Andrés (2018) comenta que la arquitectura orientada a microservicios hereda los problemas relacionados a los sistemas distribuidos:

- Latencia relacionada a comunicación por red en vez de invocaciones en memoria.
- Problema de arrastre de fallas (fallas en cascada).
- Complejidad de supervisión del sistema (monitoreo).
- Complejidad en la identificación de las fallas y problemas.

1.2 Arquitectura de servicios REST.

En el año 2000, Roy Fielding propuso la transferencia de estado representacional (REST) como un enfoque arquitectónico para diseñar servicios web. REST es un estilo arquitectónico para construir sistemas distribuidos basados en hipermedia. REST es independiente de cualquier protocolo subyacente y no está necesariamente vinculado a HTTP. Sin embargo, las implementaciones de API REST más comunes usan HTTP como protocolo de aplicación (Microsoft, 2023).

Según la documentación de Oracle (2021), la arquitectura REST se fundamenta en el protocolo HTTP, y se emplea en la creación de servicios web. Dado que el protocolo HTTP es el que subyace en la World Wide Web (WWW), REST se considera una tecnología web. En la actualidad, la mayoría de las aplicaciones web modernas utilizan la arquitectura REST para exponer sus funcionalidades en forma de servicios web, los cuales pueden ser accedidos por medio de HTTP.

1.2.1 Principios de la arquitectura REST.

La arquitectura REST se basa en un conjunto de principios que permiten la creación de servicios web simples, escalables y eficientes (Mascheroni & Irrazábal, 2016). Estos principios incluyen el uso de recursos identificados por URIs, la manipulación de recursos a través de representaciones, la utilización de métodos HTTP para indicar la acción a realizar sobre el recurso, interfaz uniforme, la utilización de un sistema de capas, entre otros (Mascheroni & Irrazábal, 2016). En la Tabla 1 se muestra cada uno de los principios y su descripción.

Tabla 1

Principios de la arquitectura REST

Principio	Descripción
Identificación de recursos	"El principio establece que cada recurso debe tener una identificación única y globalmente reconocida, como una URL (Uniform Resource Locator) en la web. Esta identificación permite que el recursos sea manipulado por el cliente y el servidor utilizando el protocolo http."[b]
Manipulación de recursos a través de representaciones	"El principio de manipulación de recursos en la arquitectura REST implica que los recursos del sistema deben ser manipulados a través de un conjunto uniforme de operaciones definidas por el protocolo HTTTP. Estas operaciones incluyen GET, POST, PUT Y DELETE, y se utilizan para leer, crear, actualizar y eliminar recursos respectivamente."[a]
Autodescripción de mensajes	"Los mensajes en REST deben ser autodescriptivos, lo que significa que deben contener suficiente información para describir como procesar el mensaje."[a]
HATEOAS	"El principio HATEOAS significa que el cliente puede interactuar con el sistema a través de

	enlaces hipertextuales que se proporcionan en las respuestas del servidor."[b]
Sistema en capas	"La arquitectura REST permite que los componentes se implementen en capas, lo que significa que los componentes no necesitan conocer la identidad de los componentes con los que interactúan directamente."[b]
Interfaz uniforme	"La interfaz uniforme es el principal factor que distingue a REST de otros estilos arquitectónicos. Define la interfaz de un recurso como una colección de operaciones bien definidas, que son comunes a todos los recursos."[b]

Nota. [a](Richardson & Ruby, 2007)[b](Fielding, 2000)

1.2.2 Ventajas y desventajas de la arquitectura REST.

Como cualquier arquitectura, REST tiene sus ventajas y desventajas que deben ser consideradas antes de poder adoptarla en un proyecto. De la misma manera, su uso debe ser cuidadosamente considerando en función de los requisitos y objetivos del proyecto.

La arquitectura REST presenta varias ventajas que la hacen una opción atractiva para el desarrollo de sistemas web y servicios web. A continuación, se presentan algunas ventajas según Marii & Zholubak (2022) :

- REST API es fácil de entender y aprender debido a su sencillez.
- Se puede organizar complejas aplicaciones y facilitar el uso de recursos.
- La carga alta se puede administrar con servidores proxy HTTP y cache.
- Las API REST son fáciles de explorar y descubrir.
- Brinda flexibilidad de formato al serializar datos en formato XML o JSON.
- Permite utilizar seguridad estándar mediante OAuth protocolos para validar solicitudes REST.
- Los usuarios pueden acceder a los mismos objetos y datos estándar modelo en comparación con los servicios web basados en SOAP.

A pesar de las ventajas que ofrece la arquitectura REST, también existen algunas desventajas que deben ser tomadas en cuenta por los desarrolladores y diseñadores de sistemas web y servicios web. Las desventajas de acuerdo con Halili & Ramadani (2018) son:

- Comparado con SOAP no cubre todas las variades de estándares de servicios web como: Seguridad, Transacciones, etc.
- Las solicitudes REST (especialmente GET) no son adecuadas para una gran cantidad de datos.
- Latencia en los tiempos de procesamiento de solicitudes y uso de ancho de banda.
- Las API REST terminan dependiendo de los encabezados para el estado (como para enrutar las solicitudes posteriores al mismo servidor de back-end que manejó la actualización anterior o para la autenticación). El uso de encabezados es torpe y vincula a la API a http como un transporte.

1.2.3 API-REST de Spotify.

Spotify es un servicio de música, pódcasts y vídeos digitales que te da acceso a millones de canciones y a otro contenido de creadores de todo el mundo (Spotify, n.d.).

Spotify Web API permite la creación de aplicaciones que pueden interactuar con el servicio de transmisión de Spotify, como recuperar metadatos de contenido, obtener recomendaciones, crear y administrar listas de reproducción o controlar la reproducción (Spotify, n.d.).

Según Spotify (2023), para comenzar a usar la API-REST se necesita realizar los siguientes pasos:

- Inicie sesión en el tablero con su cuenta de Spotify.

- Crea una aplicación. Una vez que haya creado su aplicación, tendrá accesos a las credenciales de la aplicación. Estos serán necesarios para la autorización de la API para obtener un token de acceso. Tal como se muestra en la Figura 5.

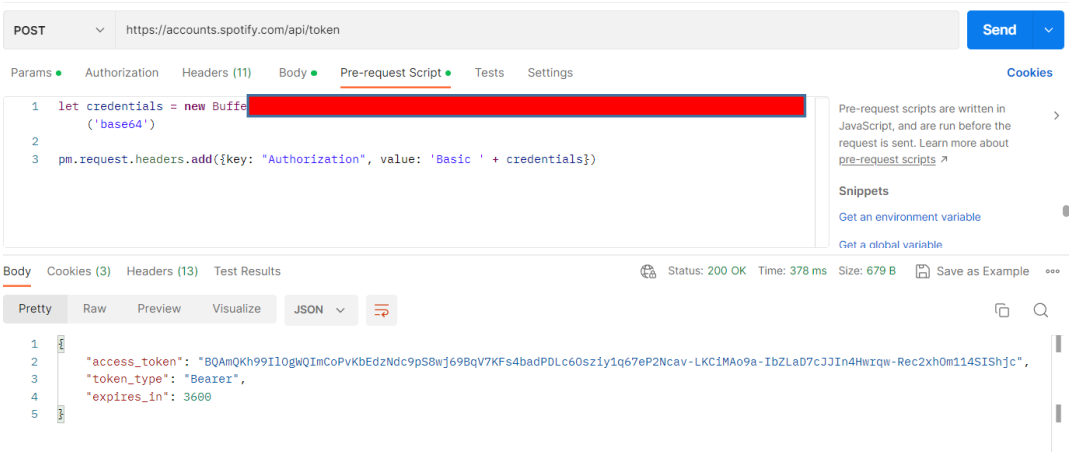
Figura 5

Obtención de las credenciales de la aplicación.

- Una vez obtenido las credenciales de la aplicación, se pueden utilizar para solicitar un token tal y como se indica en la Figura 6, en dónde se usa la herramienta tecnológica Postman.

Figura 6

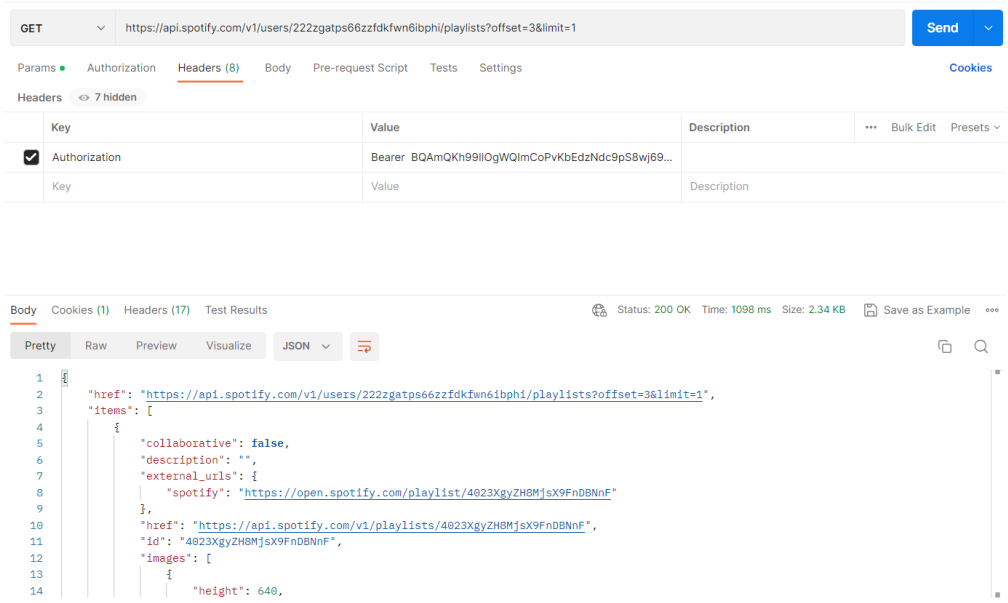
Obtención del token usando las credenciales de la aplicación.



- Por último, es posible utilizar el token generado para la solicitud del recurso que queramos de Spotify, en la Figura 7 se muestra cómo se usa el token para obtener las listas de reproducción de un usuario.

Figura 7

Obtener listas de reproducción de un usuario en específico.



1.3 Lenguaje de consultas GraphQL.

1.3.1 Definición.

GraphQL es un lenguaje de consulta y un servidor de tiempo de ejecución que se utiliza en las API para proporcionar a los clientes los datos que solicitan de manera precisa. Con GraphQL, los desarrolladores pueden crear consultas que extraen datos de múltiples fuentes en una sola llamada a la API, lo que hace que las API sean rápidas, flexibles y fáciles de usar (RedHat, 2019).

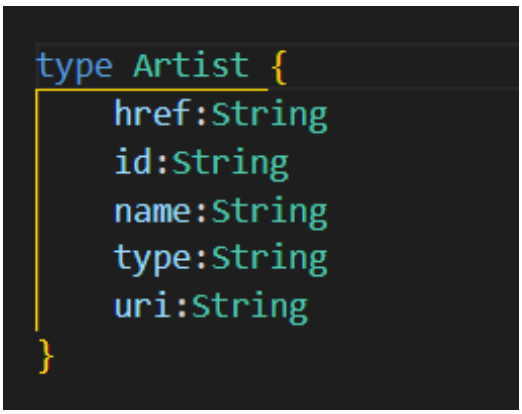
Además, GraphQL permite a los clientes solicitar solo los datos que se necesitan y nada más, lo que puede mejorar el rendimiento y la eficiencia de las aplicaciones. A diferencia de REST, que utiliza múltiples endpoints para diferentes tipos de datos, GraphQL utiliza un solo endpoint para todas las solicitudes de datos (Andrés & Solano, 2019).

1.3.2 Esquemas.

De acuerdo con Moreno (2019),las APIs GraphQL se pueden escribir en cualquier lenguaje de programación. GraphQL tiene su propio sistema para definir el esquema de una API. La sintaxis para escribir esquemas se denomina lenguaje de definición de esquema (SDL por sus siglas en inglés) En la Figura 8 se puede apreciar cómo se define un esquema.

Figura 8

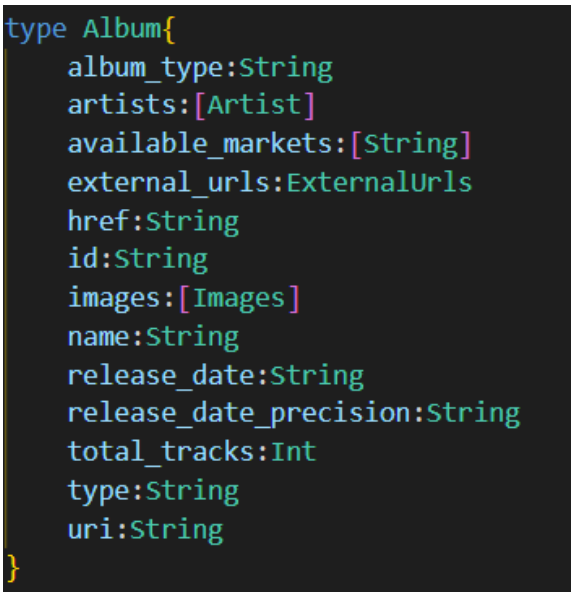
Esquema representando a un artista.



También es posible expresar relaciones entre tipos, teniendo en cuenta que las relaciones entre tipos cuando son entre varios se expresan a modo de corchete. En el ejemplo anterior un artista o varios pueden estar asociados a un álbum, tal y como se muestra en la Figura 9.

Figura 9

Esquema representando a un álbum.



1.3.3 Consultas.

En GraphQL, una query es una solicitud de datos que se envía al servidor. Las queries en GraphQL son declarativas, lo que significa que el cliente especifica exactamente qué datos necesita y cómo se deben estructurar. Esto permite que el servidor envié solo los datos

necesarios, lo que puede mejorar el rendimiento y la eficiencia de la aplicación. Además, GraphQL permite que varias consultas se combinen en una sola solicitud, lo que reduce la cantidad de solicitudes que deben enviar al servidor (Angele et al., 2022). En la Figura 10 se muestra cómo se define una query, en este caso se desea obtener un artista, por lo que va a devolver la respuesta de tipo “ResponseArtistById” y como parámetro se le tiene que pasar el “ID” del artista.

Figura 10

Definición de un Query en GraphQL.

```
type Query {
    obtenerArtistById(id:ID!):ResponseArtistById
}
```

Una vez creado el query y su resolver correspondiente, en la Figura 11 se puede observar cómo realizar una solicitud para obtener en este caso a un artista por su identificador y en la Figura 12 se muestra la respuesta.

Figura 11

Cuerpo de la petición para obtener el artista por ID en GraphQL.

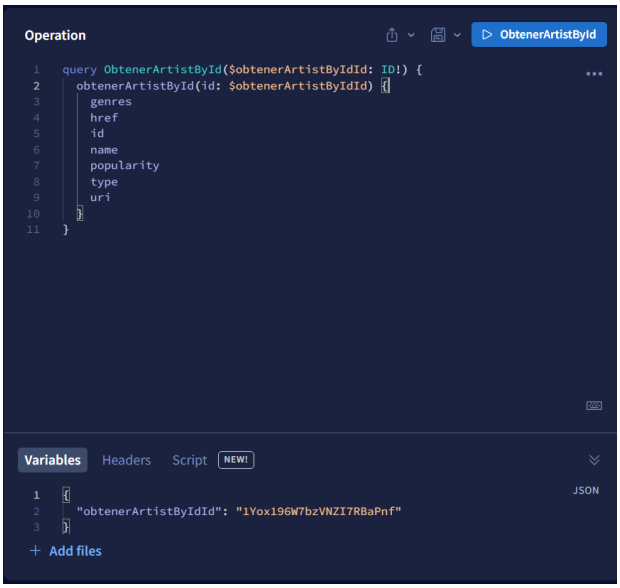
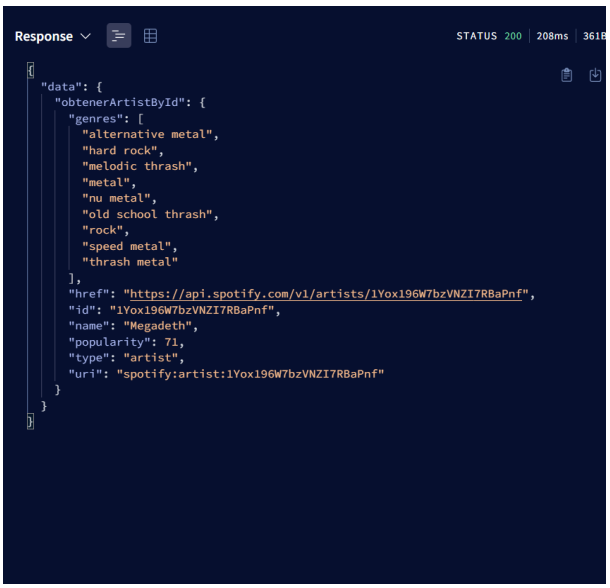


Figura 12

Cuerpo de la petición para obtener el artista por ID en GraphQL.



1.3.4 Mutaciones.

En REST, cualquier solicitud puede terminar causando algunos efectos secundarios en el servidor, pero por convención se sugiere que no se use el verbo GET para modificar datos. GraphQL es similar: técnicamente, cualquier consulta podría implementarse para provocar una escritura de datos. Sin embargo, es útil establecer una convención de que cualquier operación que provoque escrituras debe enviarse explícitamente a través de una mutación (GraphQL, n.d.).

De acuerdo con Moreno (2019), las mutaciones siguen la misma estructura sintáctica de las querys, pero siempre deben empezar con la palabra clave mutation. Un ejemplo para crear un nuevo usuario se puede demostrar en la Figura 13, donde se crea un usuario con nombre y edad, para luego solicitar su identificador.

Figura 13

Cuerpo de la petición para crear un nuevo usuario.

```
mutation {
  createUsuario(nombre: "Freddy", edad: 42) {
    _id
  }
}
```

Nota. Adaptado de Generación automática de APIs GraphQL a partir de mappings RML con MongoDB (p.10), por Moreno D, 2019, Escuela Técnica Superior de Ingenieros Informáticos.

1.4 Metodologías y herramientas de desarrollo de software.

1.4.1 Metodologías de software

La metodología de software es un conjunto de métodos, herramientas y técnicas utilizados para gestionar la calidad en el desarrollo de un producto de software(Carrizo & Alfaro, 2018). Las organizaciones buscan constantemente nuevas metodologías y herramientas que apoyen los procesos de desarrollo de productos de calidad, y surgen metodologías ágiles como scrum y XP que buscan ahorrar tiempos y costos en el desarrollo de software (Jiménez Builes et al., 2019).

En la actualidad, existen dos tipos de metodologías que se utilizan en la construcción de software:

- **Metodologías tradicionales**

  Las metodologías tradicionales de desarrollo de software se centran en el control del proceso y en un riguroso seguimiento de las actividades involucradas en ellas. Algunas de las metodologías tradicionales más utilizadas incluyen la modelo cascada y el modelo espiral (Tinoco et al., 2010).

- **Metodologías agiles**

  Las metodologías ágiles de desarrollo de software son un conjunto de prácticas y principios que se utilizan para mejorar la eficiencia y la calidad del producto final. Estas metodologías se centran en la entrega rápida y la adaptabilidad a los cambios en los requisitos del proyecto. Algunas de las metodologías ágiles más populares

incluyen Scrum, Extreme Programming (XP), Kanban y Scrumban (Molina et al., 2018).

1.4.2 Scrum como marco de trabajo

Scrum es una metodología de gestión de proyectos de software que se centra en la entrega rápida y la adaptabilidad a los cambios en los requisitos del proyecto. Scrum se basa en un enfoque iterativo e incremental, en el que el equipo de desarrollo trabaja en sprints, que son períodos de tiempo fijos y cortos, generalmente de dos a cuatro semanas, durante las cuales se desarrolla un conjunto de funcionalidades (Tinoco et al., 2010). En la Figura 14 se muestra la metodología de scrum de forma gráfica.

- **Roles en Scrum**

  En Scrum, hay tres roles principales: el Scrum Master, el Product Owner y el equipo de Desarrollo. El Scrum Master es responsable de garantizar que el equipo de desarrollo siga las prácticas de Scrum y de eliminar cualquier obstáculo que pueda impedir el progreso del equipo. El producto Owner es responsable de definir y priorizar el Product Backlog, que es una lista de funcionalidades que se deben desarrollar. El equipo de Desarrollo es responsable de desarrollar el incremento del Producto durante cada Sprint (Baumgart et al., 2015).

- **Artefactos en Scrum**

  Tal como menciona Nazareno et al (2015), los artefactos en Scrum son el Producto Backlog, el Sprint Backlog y el incremento. El Product Backlog es una lista ordenada de las funcionalidades que se deben desarrollar en el proyecto, mientras que el Sprint Backlog es una lista de las funcionalidades que se deben desarrollar durante el sprint actual y debe ser una versión potencial del producto.

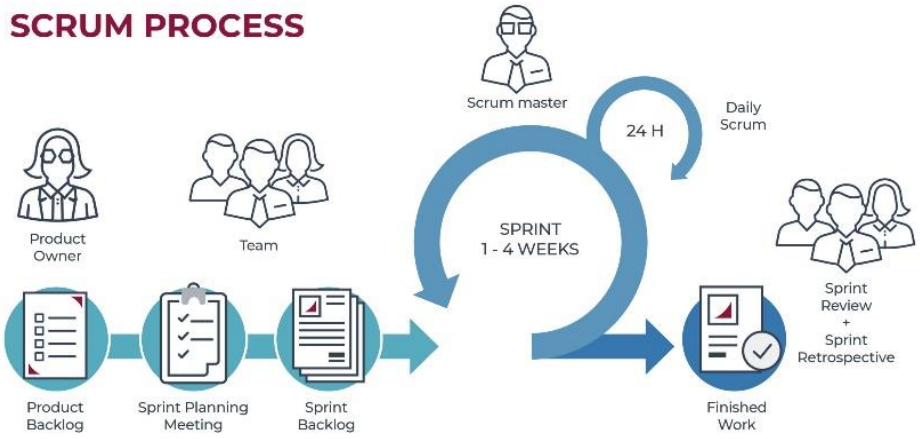
- **Eventos en Scrum**

  Según Ballesteros (2021), se realizan cuatro eventos por sprint con tiempo limitado: la planificación del sprint, la reunión diaria (daily scrum), la revisión del sprint y la retrospectiva del sprint. La planificación del sprint se lleva a cabo al comienzo de cada sprint y se utiliza para definir los objetivos y el alcance del sprint. La reunión diaria es una reunión de 15 minutos en la que el equipo se pone al día sobre el progreso del sprint. La revisión del sprint se lleva a cabo al final del sprint y se utiliza

para revisar el trabajo completado y demostrar el producto al cliente. La retrospectiva del sprint se lleva a cabo después de la revisión del sprint y se utiliza para reflexionar sobre el sprint y mejorar el proceso de desarrollo. En la Figura 14 se puede observar un resumen de todo el proceso.

Figura 14

Proceso de Scrum



Nota. Adaptado de ¿QUÉ ES SCRUM? CONOCE EL FRAMEWORK QUE AGILIZA EL TRABAJO EN EQUIPO, por Toro, 2022, Escuela de negocios y dirección (https://www.escueladenegociosydireccion.com).

1.4.3 Herramientas tecnológicas usadas en el desarrollo web

- **Node.js**

  Node.js es un entorno de tiempo de ejecución de JavaScript que se utiliza para crear aplicaciones del lado del servidor y de red a través de servidores privados virtuales. Ofrece operaciones de entrada/salida (E/S) no bloqueantes y está construido según una arquitectura asincrónica basa en eventos para ayudar a los desarrolladores a crear diversos proyectos de forma eficiente y sencilla (Hostinger, n.d.). Además, Node.js se utiliza de forma frecuente para la creación de proveedores de servicios web RESTful (Huerta Guijarro, 2015).

- **Express.js**

Express.js es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles. Con Express se pueden utilizar miles de métodos de programa y de utilidad HTTP y middleware a su disposición, la creación de una API sólida es rápida y sencilla (Express, n.d.). Según Barsoti & Gibertoni (2020), Express.js actúa como una capa en la parte superior de Node.js, lo que el desarrollo de APIs en Node.js sea más práctico y fácil de organizar.

- **Apollo**

Apollo GraphQL es la plataforma de desarrollo para crear un supergrafo: una red unificada de los datos y servicios de su organización, todo compuesto en una sola API GraphQL distribuida (Apollo, n.d.). Apollo GraphQL cuenta con una amplia comunidad de desarrolladores que comparten conocimientos y herramientas, lo que hace que sea más fácil para los desarrolladores resolver problemas y aprender nuevas habilidades (Apollo Community, n.d.).

1.5 Experimentación en la ingeniería de software.

1.5.1 Introducción

La aplicación de experimentos en ingeniería de software es de suma importancia y puede ser bien aprovechadas desde un punto de vista tanto docente como estudiantil. El diseño de experimentos permite a los estudiantes comprender un sistema y las variables externas que pueden afectarlo, logrando minimizar su efecto al obtener información necesaria que conlleva a la toma de decisiones y a la aplicación de conceptos propios de la ingeniería (Gómez, 2010).

1.5.2 Beneficios

El diseño de experimentos en la ingeniería de software tiene varios beneficios. En primer lugar, permite la validación empírica de propuestas teóricas y la evolución del conocimiento. Además, hace del desarrollo de software una actividad predecible gracias al conocimiento de las relaciones entre los procesos de producción y los productos que se obtienen (Gómez, 2010). También permite la combinación de resultados de varios estudios experimentales para generar nuevas piezas de conocimientos más generales y fiables (Fernández et al., 2010).

1.5.3 Importancia

Wohlin et al. (2012) argumenta que la experimentación es esencial para mejorar la calidad y la eficiencia del software. Además, la experimentación es una herramienta valiosa para evaluar nuevas tecnologías, herramientas y metodologías de desarrollo de software.

Wohlin et al. (2012) también sostiene que la experimentación puede ayudar a los desarrolladores de software a tomar decisiones informadas sobre cuestiones importantes del proceso de desarrollo, como la selección de métodos de prueba, la estimación de costos y la evaluación de la satisfacción del cliente. Además, la experimentación permite la evaluación objetiva y cuantitativa de la eficacia de diferentes soluciones a problemas de software.

1.5.4 Proceso de experimentación

De acuerdo con Wohlin et al. (2012) el proceso de experimentación en la ingeniería de software consta de los siguientes pasos:

- **Identificación del problema**: En este paso se identifica el problema o la cuestión que se desea resolver o investigar mediante la experimentación.
- **Formulación de la hipótesis**: Se formulan hipótesis que se van a comprobar durante la experimentación.
- **Diseño experimental**: Se diseña el experimento en sí mismo, definiendo las variables y los grupos de prueba.
- **Selección de muestras**: Se selecciona la muestra de participantes para la experimentación.
- **Recopilación de datos**: Se recopilan los datos necesarios para llevar a cabo el experimento.
- **Análisis de datos:** Se analizan los datos recopilados y se comprueban las hipótesis formuladas.

- **Interpretación de resultados:** Se interpretan los resultados obtenidos a partir del análisis de datos y se extraen conclusiones.

- **Comunicación de resultados:** Se comunican los resultados de la experimentación y las conclusiones a través de publicaciones, informes y presentaciones.

- **Verificación y validación:** Se verifica y valida la experimentación para asegurar su validez y confiabilidad.

1.6 Estándar ISO/IEC 25023.

1.6.1 ISO/IEC 25000

ISO/IEC 25000, conocida como SQuaRE (System and Software Quality Requirements and Evaluation), es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software (ISO25000, n.d.).

La familia ISO/IEC 25000 es el resultado de la evolución de otras normas anteriores, especialmente de las normas ISO/IEC 9126, que describe las particularidades de un modelo de calidad del producto software, e ISO/IEC 14598, que abordaba el proceso de evaluación de productos software. Esta familia de normas ISO/IEC 25000 se encuentra compuesta por cinco divisiones (ISO25000, n.d.). Tal y como se muestra en la Figura 15.

Figura 15

División para gestión de la calidad

Nota. Adaptado de División para gestión de la calidad, por ISO 2500, ISO2500 (https://iso25000.com/index.php/normas-iso-25000).

1.6.2 ISO/IEC 2502n

Estas normas incluyen un modelo de referencia de la medición de la calidad del producto, definiciones de medidas de calidad (interna, externa y en uso) y guías prácticas para su aplicación (ISO25000, n.d.).

Según la ISO25000 (n.d.), estas son las diferentes normas que existen en la familia 2502n:

- **ISO/IEC 25020 - *Measurement reference model and guide*:** presenta una explicación introductoria y un modelo de referencia común a los elementos de medición de la calidad. También proporciona una guía para que los usuarios seleccionen o desarrollen y apliquen medidas propuestas por normas ISO.

- **ISO/IEC 25021 - *Quality measure elements*:** define y especifica un conjunto recomendado de métricas base y derivadas que puedan ser usadas a lo largo de todo el ciclo de vida del desarrollo software.

- **ISO/IEC 25022 - *Measurement of quality in use*:** define específicamente las métricas para realizar la medición de la calidad en uso del producto.

- **ISO/IEC 25023 - *Measurement of system and software product quality*:** define específicamente las métricas para realizar la medición de la calidad de productos y sistemas software.

- **ISO/IEC 25024 - *Measurement of data quality*:** define específicamente las métricas para realizar la medición de la calidad de datos.

1.6.3 ISO/IEC 25023

El propósito de ISO/IEC 25023 es definir medidas de calidad para evaluar cuantitativamente la calidad del sistema y del producto de software en términos de características y subcaracterísticas definidas en ISO/IEC 25010.

ISO/IEC 25023 proporciona un conjunto básico de medidas de calidad para cada característica y subcaracterísticas. El estándar está diseñado para usarse junto con ISO/IEC 25010 y puede usarse junto con los estándares ISO/IEC 2503n e ISO/IEC 2504n, de manera más general, para satisfacer las necesidades de los usuarios con respecto a la calidad del sistema o del producto de software. Los principales usuarios de ISO/IEC 25023 son personas que realizan actividades de especificación y evaluación de requisitos de calidad como parte del proceso de desarrollo (ISO25000, n.d.)

CAPÍTULO 2: Desarrollo

En este capítulo se explica a detalle la creación de dos productos de software. Por un lado, se tiene la creación del envoltorio GraphQL de la API-REST de Spotify y por otro la creación de un cliente con Node.js, ambos productos desarrollados bajo la metodología Scrum. El objetivo es utilizar el cliente para comparar el tiempo de respuesta que se demora en consumir el envoltorio GraphQL y la API-REST de Spotify junto con un experimento basado en la guía de Wohlin et al. (2012) titulada "*Experimentación en la ingeniería de software"* que se desarrollará en el capítulo tres.

2.1 Análisis del envoltorio GraphQL a desarrollar.

2.1.1 Roles del proyecto

En el desarrollo del proyecto utilizando la metodología Scrum, se definieron tres roles clave: el Scrum Master, el Product Owner y el Equipo de Desarrollo. En la Tabla 2 se explican los roles y los miembros del equipo.

Tabla 2

Miembros y roles en el desarrollo del proyecto

Miembro	Descripción	Rol
PhD. Antonio Quiña	Director del presente trabajo de grado y docente de la Universidad Técnica del Norte.	Dueño del producto (Product Owner)

Sr. Carlos López	Estudiante de la carrera de ingeniería de software de la Universidad Técnica del Norte.	Líder del proyecto (Scrum Master)
Sr. Carlos López	Estudiante de la carrera de ingeniería de software de la Universidad Técnica del Norte	Equipo de desarrollo (Development Team)

2.1.2 Levantamiento de requisitos

Para el levantamiento de requisitos en el desarrollo del proyecto, se adoptó un enfoque colaborativo junto al dueño del producto. Se utilizaron las historias de usuario como técnica principal para capturar y documentar los requisitos funcionales y no funcionales. Este enfoque garantizó una comprensión compartida entre el equipo de desarrollo y el dueño del producto.

Tabla 3

Historia de Usuario Nro. 1

Nombre: Obtener Token	**Identificador:** HU-01
Usuario: Investigador	
Prioridad en el negocio: Alta	**Estimación:** 4
Descripción: Como investigador quiero obtener el token para usar los endpoints del API-REST de Spotify y el envoltorio GraphQL	
Criterios de aceptación: • Tener una opción que me permita generar el token. • Mostrar el token que se generó pese a que no se va a usar explícitamente. • Mostrar el tiempo de duración del token.	
Dependencias: No tiene	

Tabla 4

Historia de Usuario Nro. 2

Nombre: Obtener listas de reproducción de un usuario en específico.	**Identificador:** HU-02
Usuario: Investigador	
Prioridad en el negocio: Media	**Estimación:** 11
Descripción: Cómo investigador quiero obtener las listas de reproducción de un usuario en específico consumiendo la API-REST de Spotify y el envoltorio GraphQL de manera individual para poder usar en los casos de prueba.	
Criterios de aceptación: • Se debe hacer la petición teniendo en cuenta la opción de usar caché o no. • La API Individual no se podrá usar directamente, solo será un requisito para los casos de prueba.	
Dependencias: HU-01	

Tabla 5

Historia de Usuario Nro. 3

Nombre: Obtener canciones de la lista de reproducción	**Identificador:** HU-03
Usuario: Investigador	
Prioridad en el negocio: Media	**Estimación:** 11
Descripción: Cómo investigador quiero obtener la lista de reproducción específica consumiendo la API-REST de Spotify y el envoltorio GraphQL de manera individual para poder usar en los casos de prueba.	
Criterios de aceptación: • Se debe hacer la petición teniendo en cuenta la opción de usar caché o no. • La API Individual no se podrá usar directamente, solo será un requisito para los casos de prueba.	

Dependencias: HU-01

Tabla 6

Historia de Usuario Nro. 4

Nombre: Obtener canciones por álbum	**Identificador:** HU-04
Usuario: Investigador	
Prioridad en el negocio: Media	**Estimación:** 11
Descripción: Cómo investigador quiero obtener todas las canciones de un álbum en específico consumiendo la API-REST de Spotify y el envoltorio GraphQL de manera individual para poder usar en los casos de prueba.	
Criterios de aceptación: • Se debe hacer la petición teniendo en cuenta la opción de usar caché o no. • La API Individual no se podrá usar directamente, solo será un requisito para los casos de prueba.	
Dependencias: HU-01	

Tabla 7

Historia de Usuario Nro. 5

Nombre: Obtener canción por el identificador	**Identificador:** HU-05
Usuario: Investigador	
Prioridad en el negocio: Media	**Estimación:** 11
Descripción: Cómo investigador quiero obtener los datos de una canción en específica consumiendo la API-REST de Spotify y el envoltorio GraphQL de manera individual para poder usar en los casos de prueba.	
Criterios de aceptación: • Se debe hacer la petición teniendo en cuenta la opción de usar caché o no.	

• La API Individual no se podrá usar directamente, solo será un requisito para los casos de prueba.
Dependencias: HU-01

Tabla 8

Historia de Usuario Nro. 6

Nombre: Obtener artista por el identificador.	**Identificador:** HU-06
Usuario: Investigador	
Prioridad en el negocio: Media	**Estimación:** 11
Descripción: Cómo investigador quiero obtener los datos de un artista en específico consumiendo la API-REST de Spotify y del envoltorio GraphQL de manera individual para poder usar en los casos de prueba.	
Criterios de aceptación: • Se debe hacer la petición teniendo en cuenta la opción de usar caché o no. • La API Individual no se podrá usar directamente, solo será un requisito para los casos de prueba.	
Dependencias: HU-01	

Tabla 9

Historia de Usuario Nro. 7

Nombre: Especificar la cantidad de registros por cada nivel.	**Identificador:** HU-07
Usuario: Investigador	
Prioridad en el negocio: Alta	**Estimación:** 27
Descripción: Cómo investigador quiero especificar el número de registros que deseo obtener por nivel tanto en el envoltorio GraphQL como en la API Rest de Spotify, dependiendo del caso de prueba especificado en el experimento.	
Criterios de aceptación:	

- El número de registros que se necesita en cada nivel no se solicitarán en la interfaz, serán quemados y se darán por hecho para el beneficio del experimento.
- El número máximo de registros que se podrá probar en cada caso es:100.000.
- La cantidad de registros serán por endpoint, dependiendo cuantos se utilicen en el nivel (uno, dos, tres…, endpoints) y el caso de prueba especificado en el experimento.

Dependencias: HU-01

Tabla 10

Historia de Usuario Nro. 8

Nombre: Escoger el número del nivel.	**Identificador:** HU-08
Usuario: Investigador	
Prioridad en el negocio: Alta	**Estimación:** 4

Descripción: Cómo investigador quiero escoger el número de nivel deseado para realizar distintos casos de prueba.

Criterios de aceptación:

- El número máximo de niveles son 5.
- En cada nivel se debe especificar el número de endpoints a llamar.
- Antes se debe escoger si quiere consultar el envoltorio GraphQL o la API Rest de Spotify directamente.

Dependencias: HU-01

Tabla 11

Historia de Usuario Nro. 9

Nombre: Nivel Nro. 1	**Identificador:** HU-09
Usuario: Investigador	
Prioridad en el negocio: Alta	**Estimación:** 15

Descripción: Como investigador, necesito obtener las listas de reproducción de un usuario específico para comparar el rendimiento entre la API-REST de Spotify y el envoltorio GraphQL. Esto me permitirá evaluar la eficiencia de cada método y comprender sus ventajas y desventajas.

Criterios de aceptación: • Mostrar el tiempo de respuesta. • El tiempo se tiene que mostrar en milisegundos, segundos y minutos. • El **nivel 1** solo tendrá en cuenta la petición especificada en la HU-02.
Dependencias: HU-01, HU-02, HU-07

Tabla 12

Historia de Usuario Nro. 10

Nombre: Nivel Nro. 2	**Identificador:** HU-10
Usuario: Investigador	
Prioridad en el negocio: Alta	**Estimación:** 40
Descripción: Como investigador, deseo obtener información detallada sobre las canciones en las listas de reproducción de un usuario para analizar sus preferencias musicales. Compararé el rendimiento entre la API-REST y el envoltorio GraphQL para obtener los detalles de las canciones y determinar cuál método es más adecuado.	
Criterios de aceptación: • Mostrar el tiempo de respuesta. • El tiempo se tiene que mostrar en milisegundos, segundos y minutos. • El **nivel 2** tendrá en cuenta la petición especificada en la HU-02 y la HU-03	
Dependencias: HU-01, HU-02, HU-03, HU-07	

Tabla 13

Historia de Usuario Nro. 11

Nombre: Nivel Nro. 3	**Identificador:** HU-11
Usuario: Investigador	
Prioridad en el negocio: Alta	**Estimación:** 60
Descripción: Como investigador, tengo interés en obtener canciones específicas de álbumes particulares para ampliar mi análisis. Evaluaré la eficacia de la API-REST de Spotify y el envoltorio GraphQL en la recuperación de estos datos para comprender sus fortalezas y limitaciones.	

Criterios de aceptación:

- Mostrar el tiempo de respuesta.
- El tiempo se tiene que mostrar en milisegundos, segundos y minutos.
- El **nivel 3** tendrá en cuenta la petición especificada en la HU-02, HU-03 y HU-04

Dependencias: HU-01, HU-02, HU-03, HU-04, HU-07

2.1.3 Product Backlog

Tras explicar cada Historia de Usuario, se creó el Product Backlog. Es una lista priorizada de funcionalidades y requisitos tanto para el envoltorio GraphQL de la API-REST de Spotify y el cliente en Node.Js. En la Tabla 14 se muestra el Product Backlog con las historias de usuario priorizadas por el Product Owner.

Tabla 14

Product Backlog

Nro. Orden	ID HU	Nombre	Estimación
1	HU-01	Obtener token.	4
2	HU-02	Obtener listas de reproducción de un usuario en específico.	11
3	HU-03	Obtener canciones de la lista de reproducción.	11
4	HU-04	Obtener canciones por álbum.	11
5	HU-05	Obtener canción por el identificador.	11
6	HU-06	Obtener artista por el identificador.	11
7	HU-07	Especificar la cantidad de registros por cada nivel.	27
8	HU-08	Escoger el número de caso.	4
9	HU-09	Nivel Nro. 1	15
10	HU-10	Nivel Nro. 2	40
11	HU-11	Nivel Nro. 3	60

2.2 Desarrollo de la API GraphQL.

El proyecto se dividirá en un total de cinco sprints. Cada sprint tendrá una duración de dos semanas, excepto el tercer sprint, que se extenderán por tres semanas. Durante la mayoría de los Sprints, se dedicarán 30 horas a la implementación de las tareas y funcionalidades. Sin embargo, en el tercer sprint, se asignarán 35 horas para abordar de manera más exhaustiva las tareas de mayor complejidad y asegurar un progreso significativo en el proyecto. En el último Sprint solo se tendrá en cuenta 20 horas, ya que solo se utilizará para implementar lo restante de todo lo planeado. En la Tabla 15 se especifica de manera resumida los Sprints.

Tabla 15

Sprints del proyecto

Sprint	Fecha inicio	Fecha fin	Duración (Horas)
Sprint 0	29-nov 2022	05-dic 2022	15
Sprint 1	06-dic 2022	20-dic 2022	30
Sprint 2	03-ene 2023	17-ene 2023	30
Sprint 3	18-ene 2023	07-feb 2023	35
Sprint 4	08-feb 2023	21-feb-2023	30
Sprint 5	22-feb 2023	07-mar 2023	20

Además de los cinco sprints mencionados, en la Tabla 15 se incluye un “Sprint 0” en el proceso. En este sprint inicial, se enfoca en la definición de la arquitectura y la validación de los puntos finales que se utilizara para la creación del envoltorio, junto a un breve entendimiento de lo que significa “niveles”. Aunque no se especifica en las Historias de Usuario, el Sprint 0 es crucial para establecer una base sólida y asegurar que el desarrollo posterior se realice de manera eficiente y con una compresión clara de los componentes técnicos involucrados.

2.2.1 Sprint 0 - Arquitectura tecnológica

Planificación

Objetivo: Definir la arquitectura tecnológica y validar los endpoints que se utilizaran en la creación del envoltorio.

Asistentes: Product Owner, Scrum Master, Development Team

Tareas: En la Tabla 16 se muestran las tareas planificadas del Sprint 0

Tabla 16

Tareas del Sprint 0

Fase de desarrollo	Tarea	Duración (Horas)
Diseño	Revisar las herramientas tecnológicas a usar.	3
Diseño	Diseñar el diagrama tecnológico.	3
Diseño	Especificar los niveles.	3
Diseño	Verificar los puntos finales a usar para el envoltorio.	3
Planificación	Detallar las tareas del sprint actual.	1
Revisión	Analizar los resultados del sprint.	1

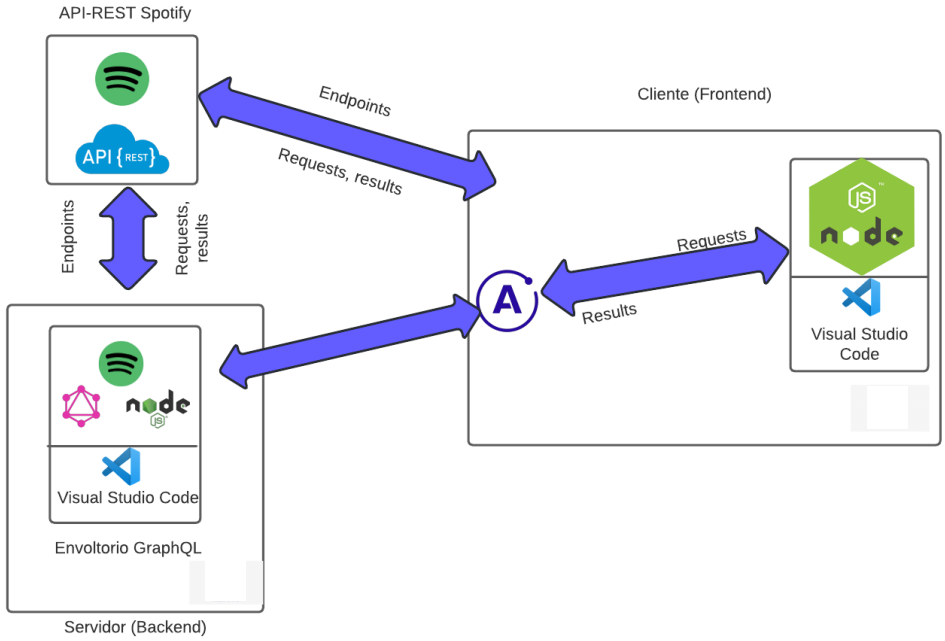
Revisión del sprint 0

Durante el Sprint 0, se llevó a cabo un análisis exhaustivo y deliberado para determinar la arquitectura tecnológica más adecuada para el proyecto. Tras un proceso de toma de decisiones, se llegó a un consenso sobre la arquitectura tecnológica que proporcionar una base sólida para el desarrollo del proyecto. En la Figura 16 se muestra de manera detallada la arquitectura y las tecnologías a usar.

Arquitectura del proyecto

Figura 16

Arquitectura tecnológica del proyecto



Puntos finales para usar en la creación del envoltorio

En el Sprint 0, se definieron los siguientes endpoints de la API-REST de Spotify a utilizar en el proyecto: "Get User's Playlists" para obtener listas de reproducción por usuario, " Get Playlist Items" para obtener canciones de una lista de reproducción y " Get Album Tracks" para obtener canciones de un álbum

Niveles

Los niveles se refieren a diferentes etapas o capas de consulta de datos en el desarrollo de la aplicación relacionada con Spotify. Estos niveles representan distintos niveles de complejidad y detalles en la obtención de información de la API-REST de Spotify y el envoltorio GraphQL.

En cada nivel, se realizan consultas adicionales para obtener datos más específicos. Por ejemplo, en el nivel 1 se obtienen las listas de reproducción por usuario. En el nivel 2, además

de las listas de reproducción, se recuperan las canciones de cada lista. En el nivel 3, se agrega la opción de obtener canciones por álbum, ampliando así la cantidad de datos disponibles.

A continuación, se explica cada uno de los tres niveles a desarrollar:

- **Nivel 1:** En este nivel, se realiza una consulta básica para obtener las listas de reproducción por usuario desde la API-REST de Spotify. Esto implica recuperar las listas de reproducción asociadas a un usuario específico y obtener los datos esenciales de cada lista. Esta consulta inicial sienta las bases para obtener información general sobre las listas de reproducción de un usuario.

- **Nivel 2:** En el nivel 2, se amplía la consulta del nivel anterior al incluir también las canciones de cada lista de reproducción. Esto implica recuperar información detallada sobre las canciones contenidas en cada lista. De esta manera, se obtiene un conjunto más completo de datos que permite analizar las canciones asociadas a las listas de reproducción de un usuario.

- **Nivel 3:** En el nivel 3, se agrega la posibilidad de obtener canciones por álbum a las consultas anteriores. Esto implica realizar consultas adicionales a la API-REST de Spotify para obtener las canciones incluidas en un álbum específico. Así, se obtiene información detallada sobre las canciones de un álbum en particular, lo que enriquece aún más los datos disponibles para el análisis.

Cada nivel sucesivo agrega nuevas capas de información y complejidad a las consultas, permitiendo una exploración más profunda de los datos de Spotify y brindando un contexto más completo para el análisis de las listas de reproducción, las canciones y los álbumes.

2.2.2 Sprint 1

Planificación del sprint 1

En el primer sprint, se revisó la documentación de la API-REST de Spotify para obtener el token necesario y se construyó un envoltorio que permite realizar peticiones relacionadas con las listas de reproducción por usuario y las canciones de una lista de reproducción en específico.

Reunión de planificación

La reunión de planificación se desarrolló con los siguientes datos generales:

- **Fecha:** 29 de noviembre de 2022
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Creación del Sprint Backlog del Sprint 2.

Sprint Backlog del Sprint 1

En la Tabla 17 se observa el Sprint Backlog del presente Sprint, donde se especifican las diferentes tareas a desarrollar con su estimación de tiempo correspondiente.

Tabla 17

Sprint 1 Backlog

Historia de usuario	Fase de desarrollo	Tarea	Estimación Tiempo
HU-01	Análisis	Revisar la documentación de cómo obtener el Token.	4
	Pruebas	Realizar pruebas con alguna herramienta de peticiones de API.	2
	Desarrollo	Crear la función en GraphQL capaz de obtener el token.	2
HU-02	Análisis	Revisar la documentación de cómo obtener las listas de un usuario en específico.	4
	Pruebas	Realizar pruebas con alguna herramienta de peticiones de API.	2
	Desarrollo	Crear la función en GraphQL capaz de obtener las listas de un usuario específico.	2
HU-03	Análisis	Revisar la documentación de cómo obtener las listas de un usuario en específico.	4
	Pruebas	Realizar pruebas con alguna herramienta de peticiones de API.	2
	Desarrollo	Crear la función en GraphQL capaz de obtener las listas de un usuario específico.	2
Eventos	Planificación	Especificar las tareas del Sprint actual	3
	Revisión	Revisar los resultados del Sprint	2

Retrospectiva	Analizar los resultados del Sprint	1
	TOTAL HORAS	30

Revisión del sprint 1

Como resultado del Sprint 1 se realizó una revisión exhaustiva de la documentación de la API-REST de Spotify con el fin de obtener el token requerido para las peticiones. Además, se construyó un envoltorio GraphQL que brinda la capacidad de realizar peticiones relacionadas con las listas de reproducción y las canciones de las listas de reproducción

Reunión de revisión

- **Fecha:** 05 de diciembre de 2022
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Revisar el incremento del producto.

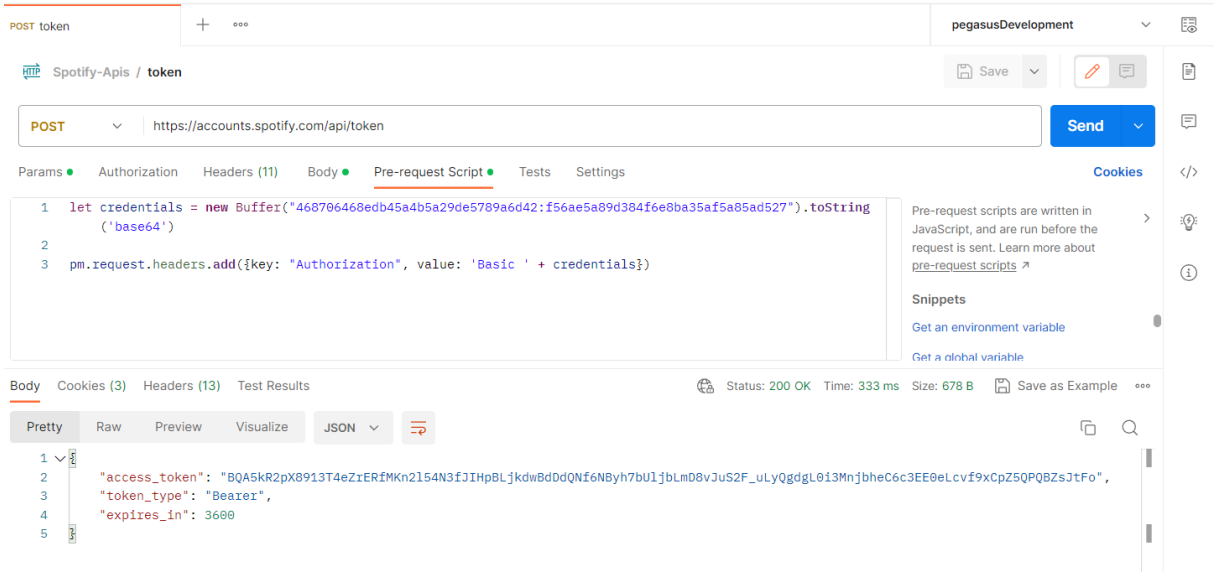
Entregable (Incremento del producto)

Obtener el Token

Según la documentación oficial de Spotify (n.d.), para obtener un token sin la necesidad de iniciar sesión como usuario es necesario utilizar el “Client Id” y el “Secret Id”, estos se le proporcionan al desarrollador cuando de crear un proyecto en la consola de desarrollador en la página oficial de Spotify para desarrolladores. En la Figura 17 se muestra una petición realizada con la herramienta Postman para obtener el token.

Figura 17

Petición para obtener el token con Postman



Luego de comprobar cómo se realiza la petición para obtener el token, se creó el envoltorio de esa misma petición obteniendo el token y guardándolo como instancia a la vez, obteniendo el resultado que se muestra en la Figura 18.

Figura 18

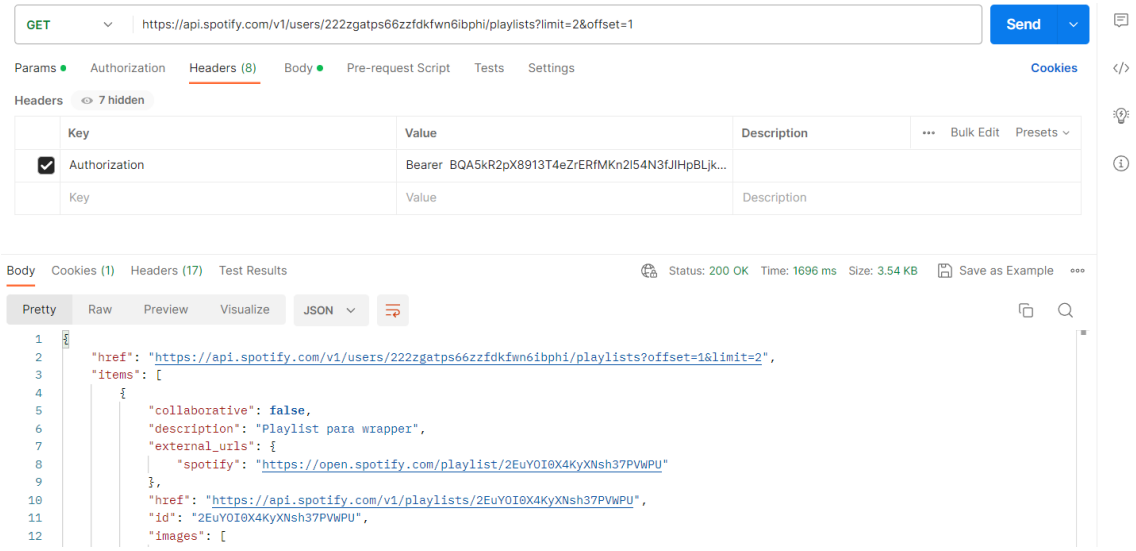
Petición para obtener el token del envoltorio GraphQL

Obtener listas de reproducción de un usuario específico

Para obtener las listas de reproducción de un usuario específico, Spotify (n.d.) específica que se necesita pasar en la URL el identificador del usuario de Spotify, en la cabecera de la petición el token obtenido y de manera opcional como parámetro se le puede pasar el “limit” para la cantidad de datos y el “offset” para el manejo de la paginación, tal y como se muestra en la Figura 19.

Figura 19

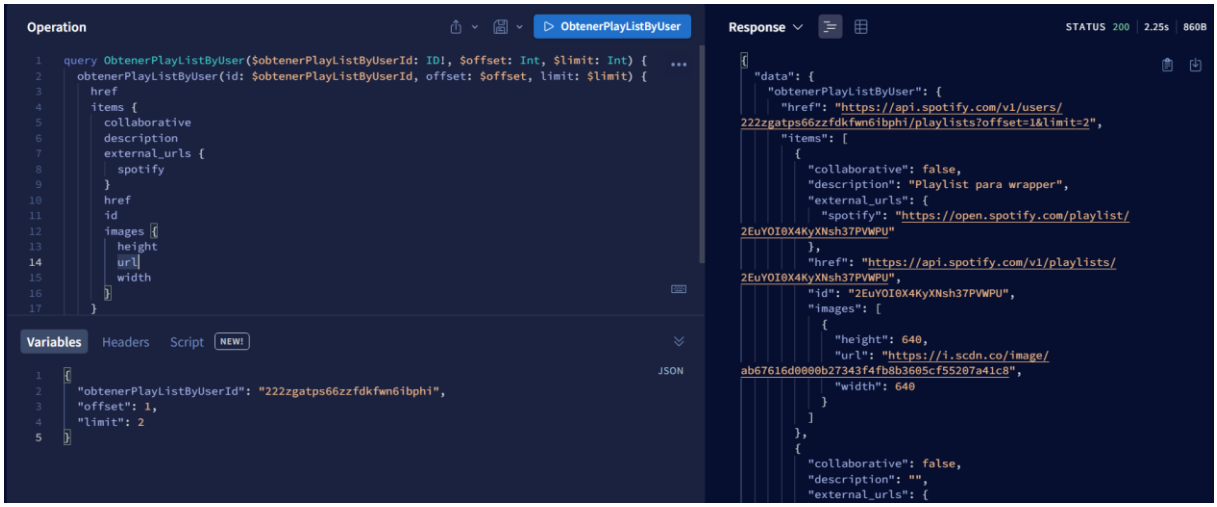
Petición para obtener las listas de reproducción de un usuario



Luego de comprobar cómo se realiza la petición, se creó el envoltorio de esa misma petición obteniendo las listas de reproducción de un usuario, Logrando el resultado que se muestra en la Figura 20.

Figura 20

Petición para obtener las listas de reproducción de un usuario del envoltorio GraphQL

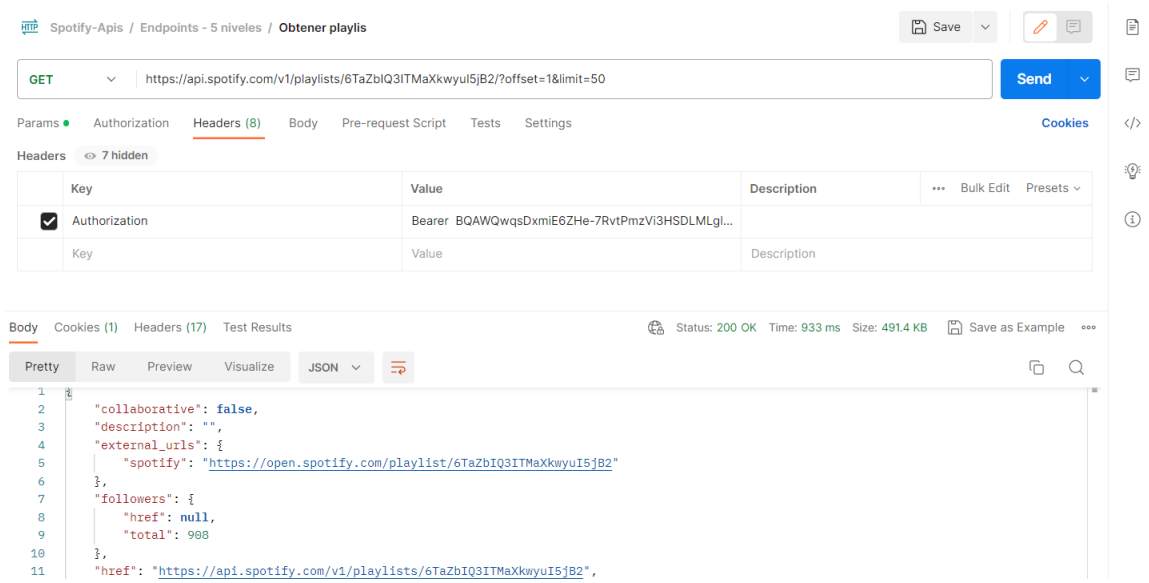


Obtener canciones de una lista de reproducción en específico.

Para obtener las canciones de una lista de reproducción, Spotify (n.d.) específica que se necesita pasar en la URL el identificador de la lista de reproducción, en la cabecera de la petición el token obtenido y de manera opcional como parámetro se le puede pasar el “limit” para la cantidad de datos y el “offset” para el manejo de la paginación, tal y como se muestra en la Figura 21.

Figura 21

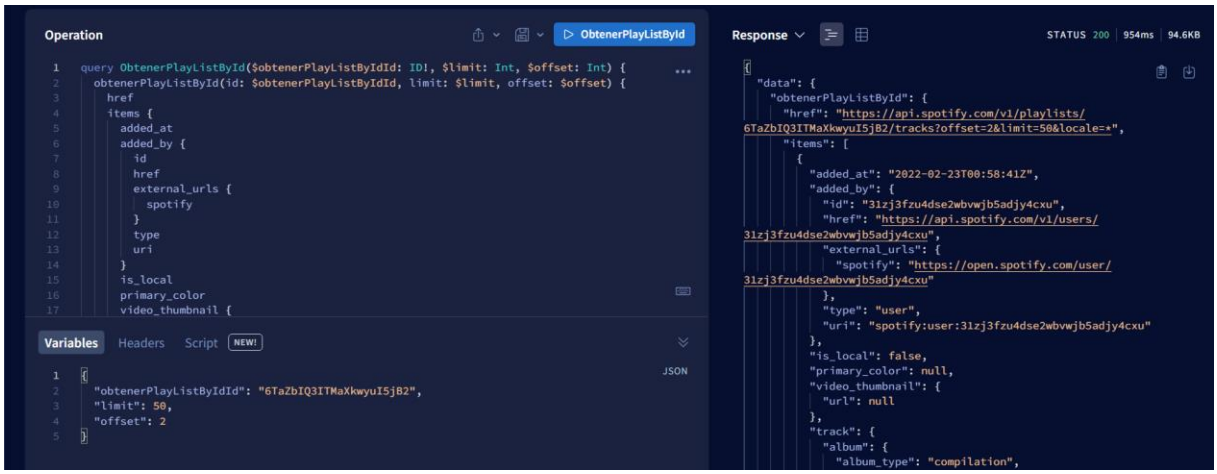
Petición para obtener las canciones de una lista de reproducción



Luego de comprobar cómo se realiza la petición, se creó el envoltorio de esa misma petición obteniendo las canciones de la lista de reproducción, Logrando el resultado que se muestra en la Figura 22.

Figura 22

Petición para obtener las canciones de una lista de reproducción del envoltorio GraphQL



Retrospectiva

La reunión de retrospectiva se realizó con los siguientes datos generales:

- **Fecha:** 20 de diciembre de 2022
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Analizar lo desarrollado y proponer mejoras.

A continuación, en la Tabla 18 se presentan los resultados de la reunión de retrospectiva que permitirá implementar mejoras para el siguiente Sprint.

Tabla 18

Retrospectiva del sprint 1

Título	Descripción
Aciertos (¿Qué salió bien del sprint?)	Se cumplió la creación de las consultas del envoltorio.
Errores (¿Qué no salió como se esperaba?)	Se tienen varias formas de obtener el token, lo que llevo a escoger e investigar la mejor forma para el presente caso.
Mejoras (¿Qué mejoras se implementará?)	Actualmente se está guardando como instancia cuando se obtiene el token, lo ideal sería pasarlo en las cabeceras de la petición.

2.2.3 Sprint 2

Durante el Sprint 2, se llevó a cabo una revisión de la documentación de los siguientes Enpoints de la API-REST de Spotify: Obtener canciones por álbum, obtener canción por identificado y Obtener artista por identificador. A partir de esta revisión, se crearon las correspondientes peticiones para integrar estos endpoints en el envoltorio GraphQL.

Reunión de planificación

La reunión de planificación se desarrolló con los siguientes datos generales:

- **Fecha:** 03 de enero de 2022
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Creación del Sprint Backlog del Sprint 2.

Sprint Backlog del Sprint 2

En la Tabla 19 se observa el Sprint Backlog del presente Sprint, donde se especifican las diferentes tareas a desarrollar con su estimación de tiempo correspondiente en horas.

Tabla 19

Sprint 2 Backlog

Historia de usuario	Fase de desarrollo	Tarea	Estimación Tiempo
HU-04	Análisis	Revisar la documentación de cómo obtener canciones de un álbum en específico.	4
	Pruebas	Realizar pruebas con alguna herramienta de peticiones de API.	2
	Desarrollo	Crear la función en GraphQL capaz de obtener canciones de un álbum en específico.	2
HU-05	Análisis	Revisar la documentación de cómo obtener canción por el identificador.	4
	Pruebas	Realizar pruebas con alguna herramienta de peticiones de API.	2

	Desarrollo	Crear la función en GraphQL capaz de obtener la canción por el identificador.	2
HU-06	Análisis	Revisar la documentación de cómo obtener artista por el identificador.	4
	Pruebas	Realizar pruebas con alguna herramienta de peticiones de API.	2
	Desarrollo	Crear la función en GraphQL capaz de obtener artista por el identificador.	2
Eventos	Planificación	Especificar las tareas del Sprint actual	3
	Revisión	Revisar los resultados del Sprint	2
	Retrospectiva	Analizar los resultados del Sprint	1
		TOTAL HORAS	30

Revisión del Sprint 2

Como resultado del Sprint 2 se completó la revisión detallada de la documentación de lo siguientes endpoints de la API-REST de Spotify: Obtener canciones por álbum, obtener detalle de una canción por el identificador y obtener detalle de un artista por el identificador. Posteriormente, se implementaron las correspondientes peticiones dentro del envoltorio GraphQL. Esto permite una integración eficiente y coherente de dichos endpoints en el flujo de trabajo del proyecto.

Reunión de revisión

La reunión de revisión se desarrolló con los siguientes datos generales:

- **Fecha:** 17 de enero de 2023
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Revisión del incremento del producto.

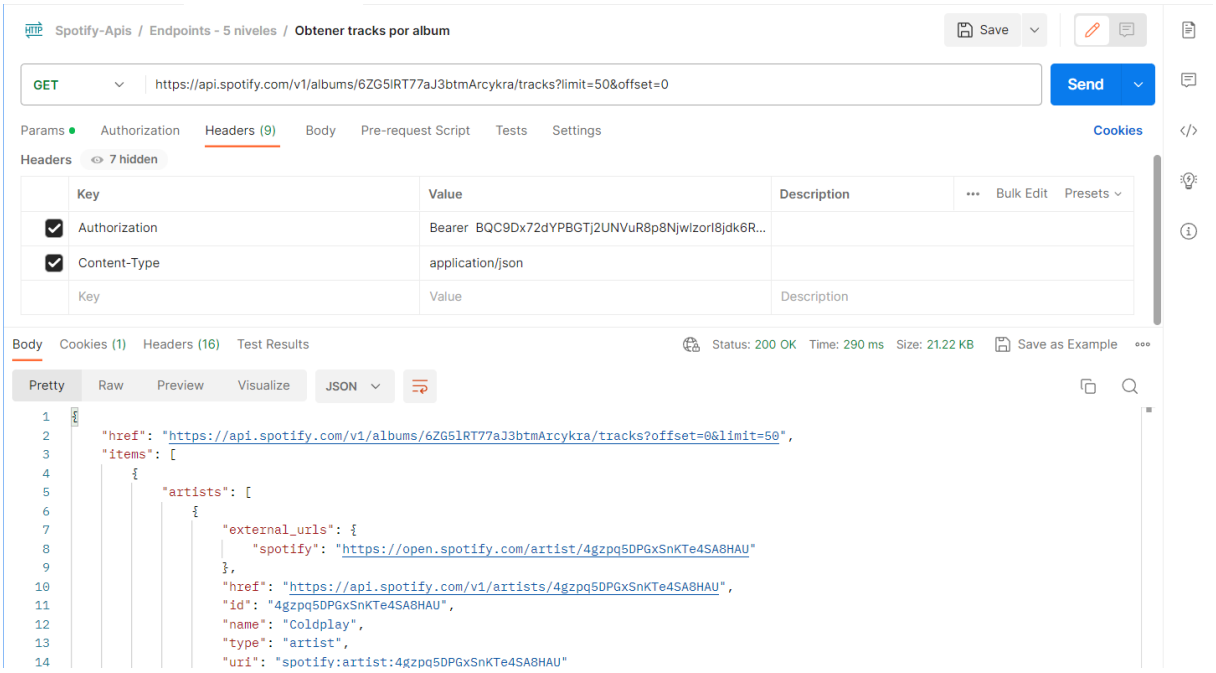
Obtener canciones de un álbum en especifico

La documentación oficial de Spotify (n.d.) específica que, para obtener las canciones de un álbum es necesario pasar el identificador del álbum en la URL de la petición y el token en la cabecera como argumentos obligatorios. Para el manejo de la paginación es necesario

pasarle como parámetros el “limit” para obtener una cantidad de datos específica y el ‘offset para recorrer en la paginación. En la Figura 23 se muestra la forma de realizar la petición en la herramienta Postman.

Figura 23

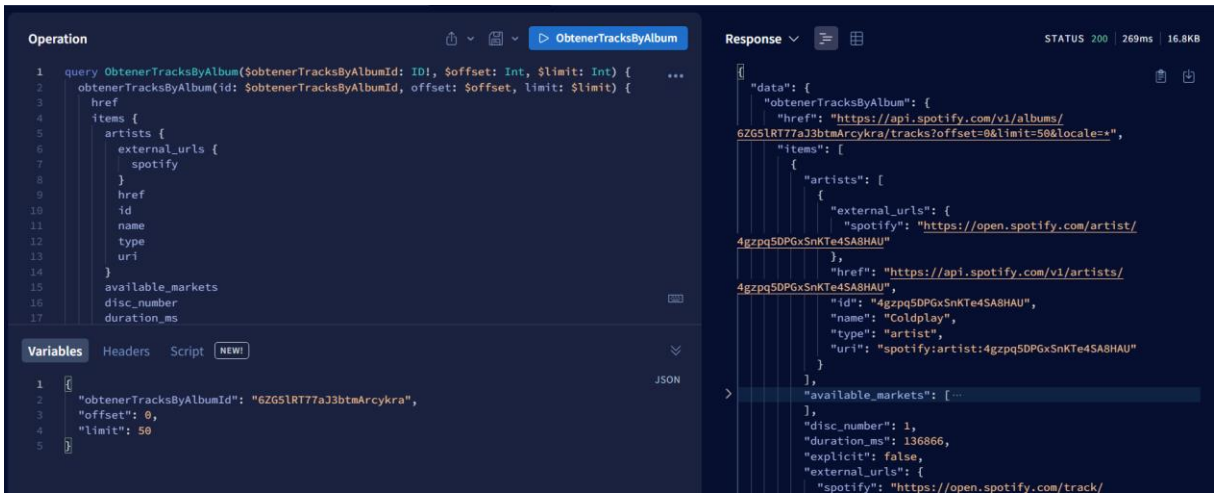
Petición para obtener las canciones de un álbum



Luego de comprobar cómo se realiza la petición, se creó el envoltorio de esa misma petición obteniendo las canciones de un álbum en específico. Logrando el resultado que se muestra en la Figura 24.

Figura 24

Petición para obtener las canciones de un álbum del envoltorio GraphQL

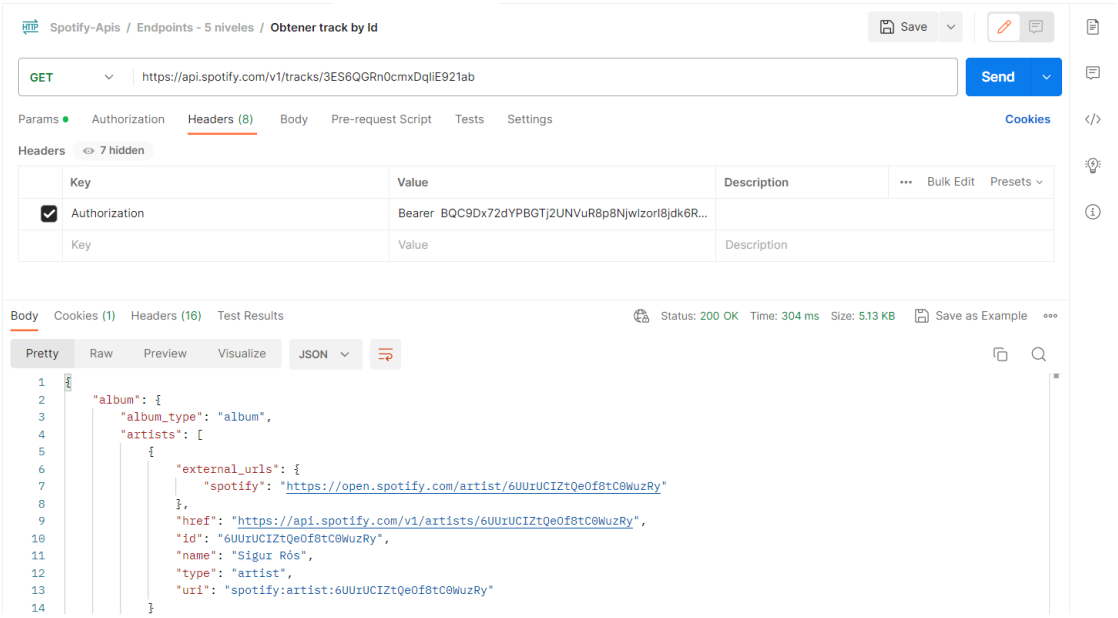


Obtener canción por el identificador.

La documentación oficial de Spotify (n.d.) específica que, para obtener el detalle de una canción es necesario pasar el identificador de la canción en la URL de la petición y el token en la cabecera como argumentos obligatorios. En la Figura 25 se muestra la manera de realizar la petición con la herramienta Postman.

Figura 25

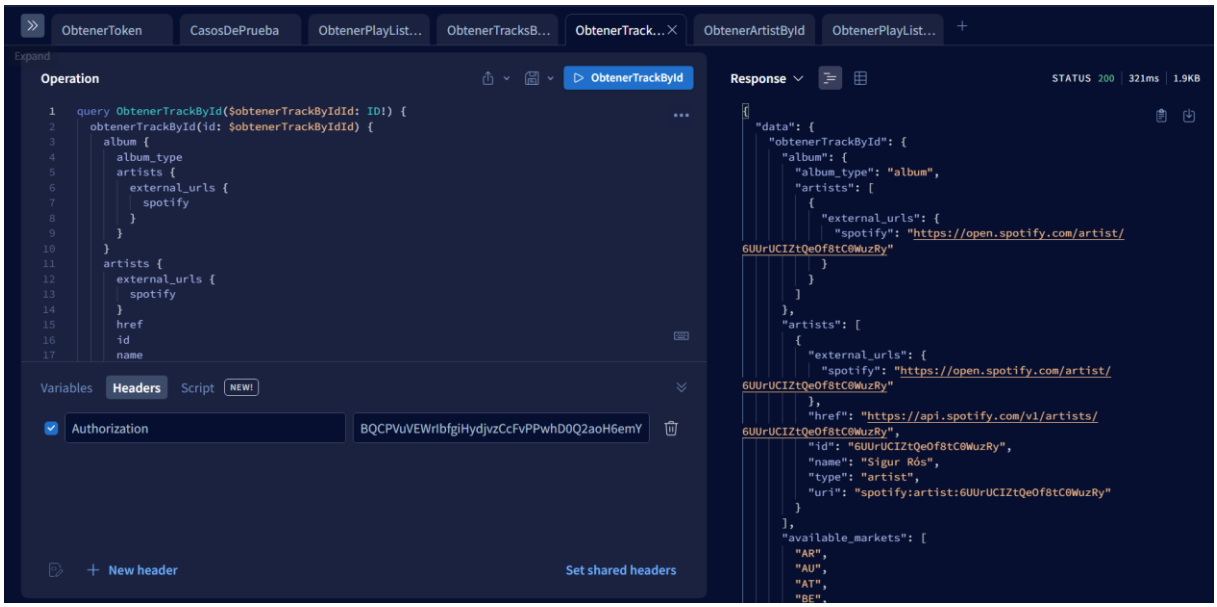
Petición para obtener el detalle de una canción



Después de comprobar cómo se realiza la petición, se procedió a crear un envoltorio para obtener el detalle de una canción por su identificador. Como resultado, se obtuvo la petición del envoltorio GraphQL que se muestra en la Figura 26.

Figura 26

Petición para obtener el detalle de una canción del envoltorio GraphQL

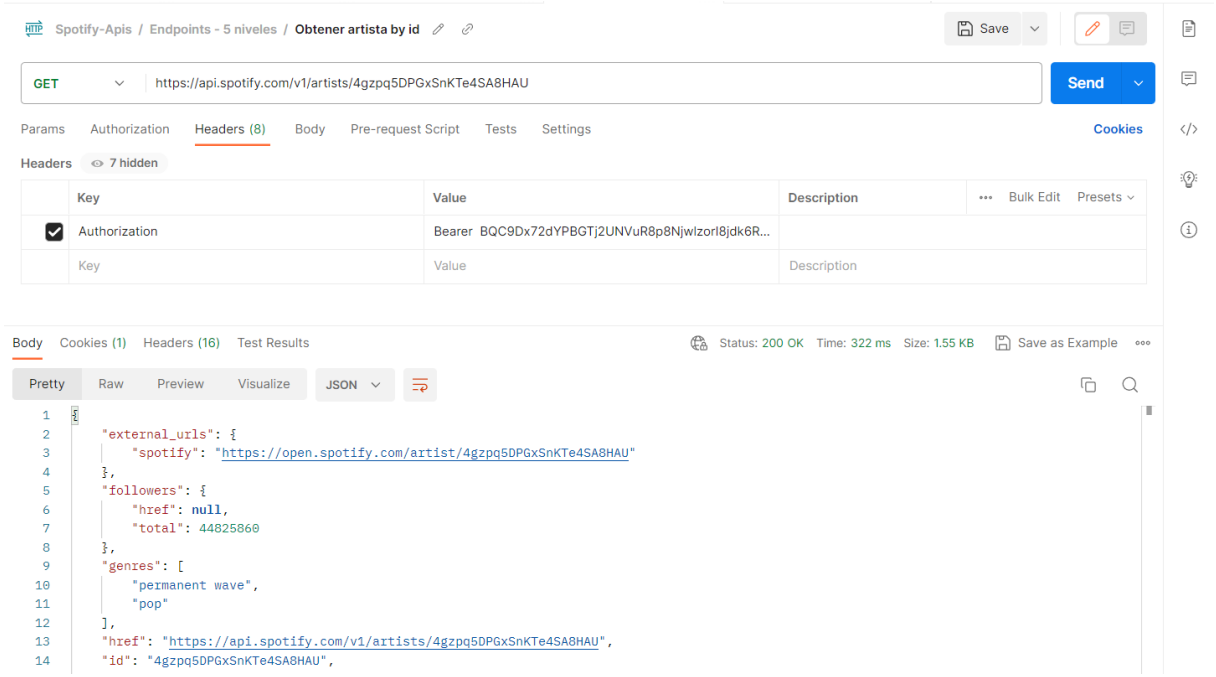


Obtener artista por el identificador.

La documentación oficial de Spotify (n.d.) específica que, para obtener el detalle de un artista es necesario pasar el identificador del artista en la URL de la petición y el token en la cabecera como argumentos obligatorios. En la Figura 27 se muestra la manera de realizar la petición con la herramienta Postman.

Figura 27

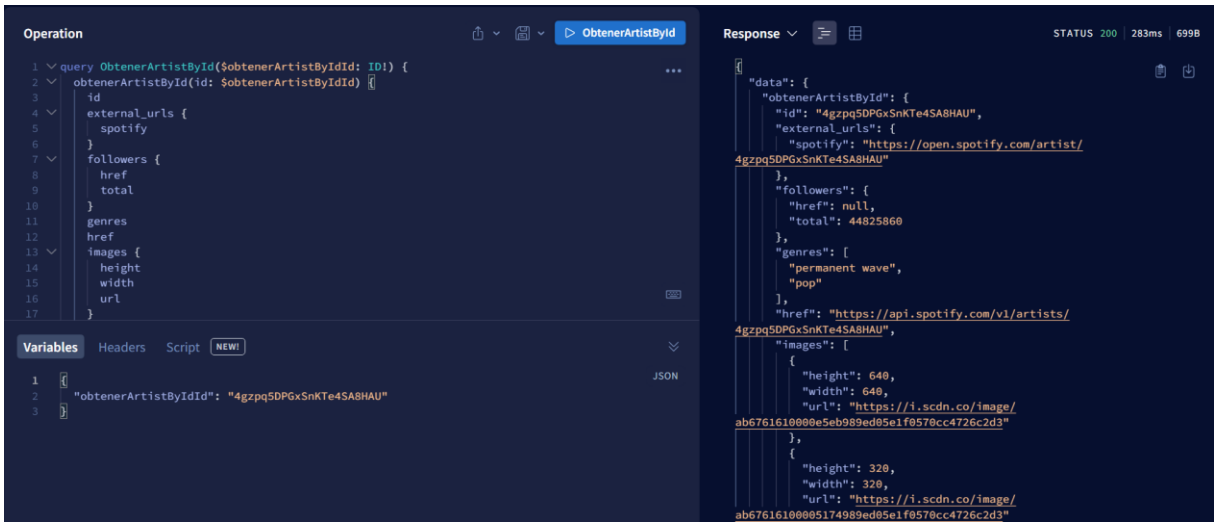
Petición para obtener el detalle de un artista



Después de comprobar cómo se realiza la petición, se procedió a crear un envoltorio para obtener el detalle de un artista por su identificador. Como resultado, se obtuvo la petición del envoltorio GraphQL que se muestra en la Figura 28.

Figura 28

Petición para obtener el detalle de un artista del envoltorio GraphQL



Retrospectiva

La reunión de retrospectiva se realizó con los siguientes datos generales:

- **Fecha:** 17 de enero de 2023
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Analizar lo desarrollado y proponer mejoras.

A continuación, en la Tabla 20 se presentan los resultados de la reunión de retrospectiva que permitirá implementar mejoras para el siguiente Sprint.

Tabla 20

Retrospectiva del sprint 2

Título	Descripción
Aciertos (¿Qué salió bien del sprint?)	Se cumplió la creación de las consultas del envoltorio.
Errores (¿Qué no salió como se esperaba?)	Se tuvo que crear varios schemas por separados en el Envoltorio GraphQL para reducir la cantidad de código.
Mejoras (¿Qué mejoras se implementará?)	Se plantea preparar los schemas y resolvers para crear una consulta que me permita realizar el experimento.

2.2.4 Sprint 3

En el Sprint 3, se realizaron modificaciones en los esquemas y se crearon resolvers anidados para las consultas previamente desarrolladas en los Sprints anteriores, teniendo en cuenta el experimento. Posteriormente, se creó un cliente en Node.js con la configuración básica necesaria para consultar los distintos niveles del experimento tanto en la API-REST de Spotify como en el envoltorio GraphQL.

Reunión de planificación

La reunión de planificación se desarrolló con los siguientes datos generales:

- **Fecha:** 18 de enero de 2022
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Creación del Sprint Backlog del Sprint 3.

Sprint Backlog del Sprint 3

En la Tabla 21 se muestra el Sprint Backlog actual, que detalla las diversas tareas a desarrollar junto con su estimación de tiempo correspondiente en horas.

Tabla 21

Sprint 3 Backlog

Historia de usuario	Fase de desarrollo	Tarea	Estimación Tiempo
HU-07	Análisis	Entender como GraphQL maneja los resolvers anidados.	5
	Análisis	Crear un diagrama para entender de mejor manera como va a funcionar la consulta con los resolvers anidados.	4
	Desarrollo	Desarrollar la consulta con resolvers anidados que permita consultar todos los puntos finales especificados para el experimento en una sola consulta.	7
	Pruebas	Probar con el servidor local de Apollo el funcionamiento correcto de la consulta.	5
HU-08	Desarrollo	Crear el cliente Node.js con la configuración necesaria para beneficio del experimento.	5
	Desarrollo	Crear una interfaz de usuario que permita consultar de manera facil los niveles del experimento.	3
Eventos	Planificación	Especificar las tareas del Sprint actual	3
	Revisión	Revisar los resultados del Sprint	2
	Retrospectiva	Analizar los resultados del Sprint	1
		TOTAL HORAS	35

Revisión del Sprint 3

Durante el Sprint 3, se lograron varios avances importantes. Como resultado, se creó un diagrama que proporciona una comprensión clara de cómo funcionan los resolvers anidados en el envoltorio GraphQL de la API-REST de Spotify. Además, se desarrolló una consulta en

el envoltorio que permite obtener de forma anidada los endpoints especificados en los Sprints anteriores. Por último, se estableció la configuración inicial del cliente en Node.js, con una pantalla de usuario amigable, con el propósito de mejorar la experiencia en el experimento.

Reunión de revisión

La reunión de revisión se desarrolló con los siguientes datos generales:

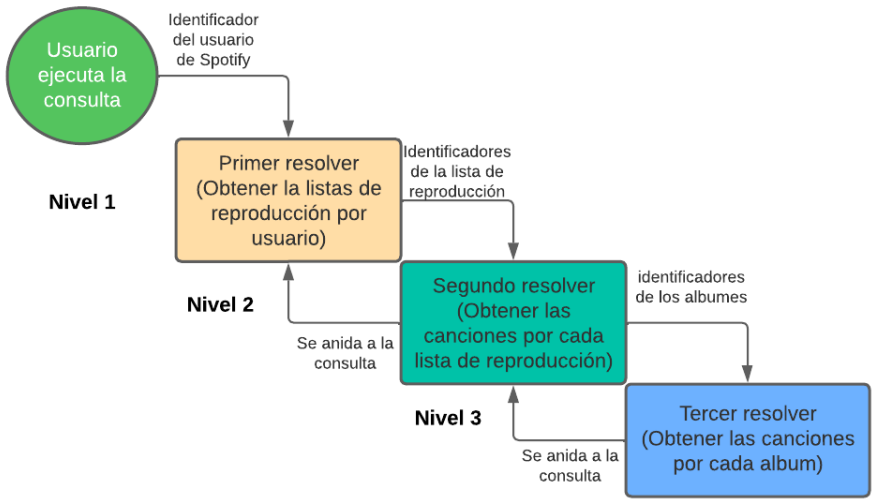
- **Fecha:** 07 de febrero de 2023
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Revisión del incremento del producto.

Diagrama de resolvers anidados del envoltorio GraphQL

Al utilizar los resolvers anidados en el envoltorio GraphQL, permite crear una consulta compleja y obtener datos relacionados por cada uno de los niveles. En la Figura 29 se muestra el diagrama creado en el presente Sprint, donde se especifican los puntos finales anidados de la API-REST de Spotify para gestionarlo desde una sola consulta en el envoltorio GraphQL.

Figura 29

Diagrama de cómo funcionan los resolvers anidados en el envoltorio GraphQL



Pruebas de la consulta con los resolvers anidados

Después de completar el diagrama que describe cómo funcionarán las consultas utilizando resolvers anidados, se procedió al desarrollo de la consulta en el envoltorio GraphQL. En el contexto del experimento, se especificaron cuatro parámetros clave en la consulta.

El primer parámetro, "userId", se utiliza para obtener las listas de reproducción asociadas a un usuario específico. Al proporcionar el identificador del usuario, el resolver anidado realizará la búsqueda y devolverá las listas de reproducción correspondientes.

El segundo parámetro, "limitFirstCase", permite controlar la cantidad de datos recuperados en el primer punto final utilizado para obtener las listas de reproducción por usuario. Este parámetro determina la cantidad de resultados que se devolverán, ajustando así la cantidad de información obtenida según las necesidades del experimento.

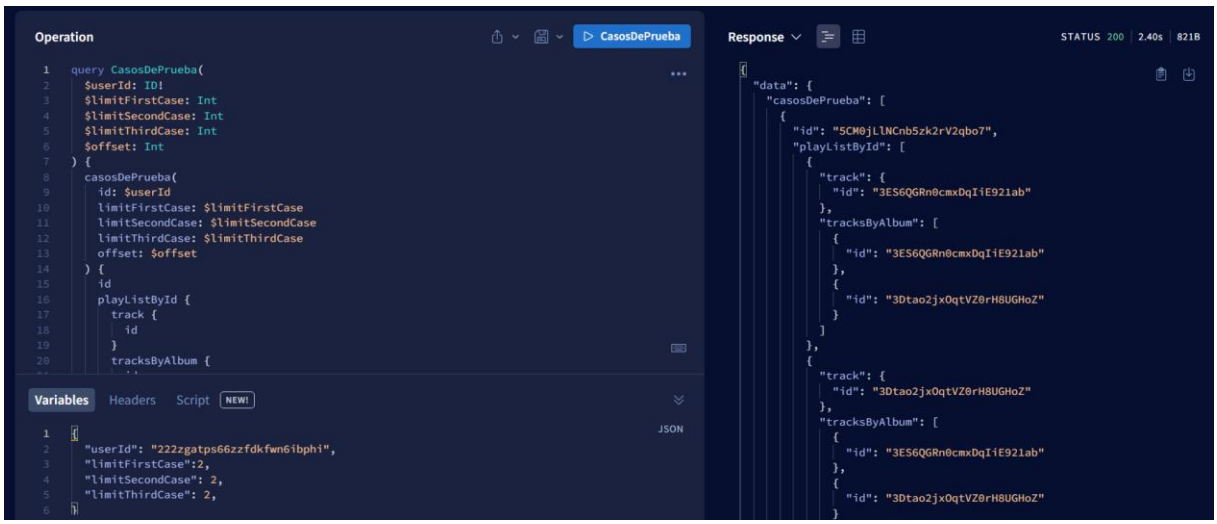
El tercer parámetro, "limitSecondcase", se utiliza para obtener una cantidad específica de datos del segundo punto final, que consiste en obtener las canciones de cada lista de reproducción. Este parámetro define la cantidad exacta de canciones que se recuperarán para cada lista de reproducción, brindando así control y precisión en los resultados obtenidos.

El cuarto parámetro, "limitThirdCase", funciona de manera similar al anterior y se utiliza para obtener una cantidad específica de datos del tercer punto final, que consiste en obtener canciones por álbum. Al ajustar este parámetro, es posible obtener solo la cantidad deseada de canciones de un álbum en particular, ofreciendo así mayor flexibilidad en el experimento.

A continuación, en la Figura 30 se muestra un ejemplo de lo desarrollado, teniendo en cuenta que para una demostración práctica y sencilla no se llaman todos los campos.

Figura 30

Petición con resolvers anidados



Creación de cliente en Node.js con su configuración inicial

Después de implementar los resolvers anidados en el envoltorio GraphQL, se procedió a desarrollar un cliente en Node.js con una configuración inicial. Este cliente está diseñado para interactuar directamente con la API-REST de Spotify y el envoltorio GraphQL, facilitando la consulta de datos de manera eficiente.

El cliente en Node.js se creó con el objetivo de brindar una experiencia de usuario amigable y accesible. Para lograr esto, en la Figura 31 se muestra el diseñó de una interfaz de usuario intuitiva que se ejecuta directamente en la interfaz de línea de comandos (CLI) en el proyecto de Node.js y en la Figura 32 se muestra el consumo de los niveles dependiendo de la primera opción que escoja en la Figura 31.

Figura 31

Interfaz para la ejecución del experimento en Node.js

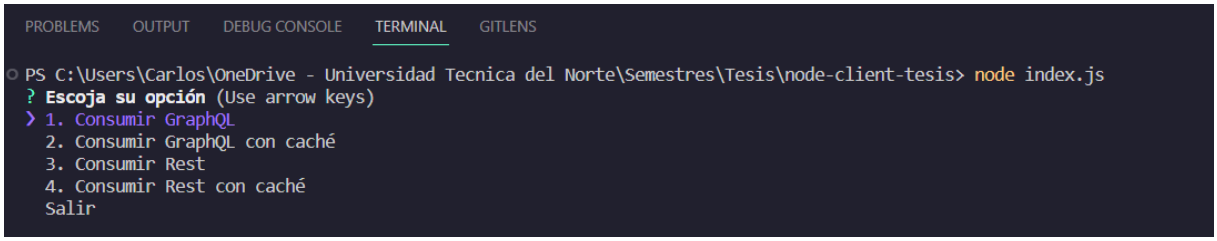
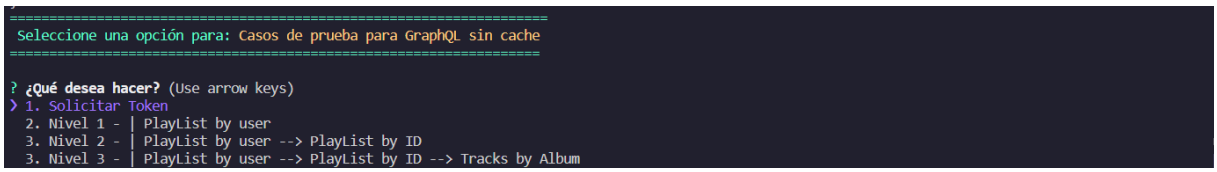


Figura 32

Escoger niveles para su ejecución



Retrospectiva

La reunión de retrospectiva se realizó con los siguientes datos generales:

- **Fecha:** 07 de febrero de 2023
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Analizar lo desarrollado y proponer mejoras.

A continuación, en la Tabla 22 se presentan los resultados de la reunión de retrospectiva que permitirá implementar mejoras para el siguiente Sprint.

Tabla 22

Retrospectiva del sprint 3

Título	Descripción
Aciertos (¿Qué salió bien del sprint?)	Se cumplió con la creación de la consulta con los resolvers anidados y la configuración inicial del cliente en Node.js.
Errores (¿Qué no salió como se esperaba?)	Al utilizar una interfaz de usuario ya no se puede utilizar la herramienta nodemon, se tiene que ejecutar directamente desde el index.js
Mejoras (¿Qué mejoras se implementará?)	Se plantea crear las consultas directamente en el cliente teniendo en cuenta como se plantea ejecutar el experimento.

2.2.5 Sprint 4

En el Sprint 4 se enfocó en la creación de consultas en el cliente de Node.js para la API-REST de Spotify y el envoltorio GraphQL. Se implementaron consultas en dos niveles: el nivel 1 para obtener listas de reproducción por usuario, el nivel 2 que incluye canciones de cada lista de reproducción. Todas estas consultas se desarrollaron con opciones de caché y sin

caché, permitiendo a los usuarios personalizar la forma de obtener y almacenar los datos de Spotify.

Reunión de planificación

La reunión de planificación se desarrolló con los siguientes datos generales:

- **Fecha:** 08 de febrero de 2022
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Creación del Sprint Backlog del Sprint 4.

Sprint Backlog del Sprint 4

En la Tabla 23 se observa el Sprint Backlog del presente Sprint, donde se especifican las diferentes tareas a desarrollar con su estimación de tiempo correspondiente en horas.

Tabla 23

Sprint 4 Backlog

Historia de usuario	Fase de desarrollo	Tarea	Estimación Tiempo
HU-09	Desarrollo	Realizar las peticiones a la API-REST con caché y sin caché del nivel 1.	5
	Desarrollo	Realizar las peticiones al envoltorio GraphQL con caché y sin caché del nivel 1.	4
	Pruebas	Probar con diferentes cantidades de datos la petición del nivel 1	3
HU-10	Desarrollo	Realizar las peticiones a la API-REST con caché y sin caché del nivel 2.	5
	Desarrollo	Realizar las peticiones al envoltorio GraphQL con caché y sin caché del nivel 2.	4
	Pruebas	Probar con diferentes cantidades de datos las peticiones del nivel 2	3
Eventos	Planificación	Especificar las tareas del Sprint actual	3
	Revisión	Revisar los resultados del Sprint	2
	Retrospectiva	Analizar los resultados del Sprint	1

TOTAL HORAS	30

Revisión del Sprint 4

Durante el Sprint 4, se dio especial atención a la creación de consultas directamente en el cliente de Node.js para la API-REST de Spotify y el envoltorio GraphQL desarrollado en los Sprints anteriores. Las consultas implementadas en este sprint se dividen en tres niveles.

En el nivel 1, se realizaron consultas tanto a la API-REST de Spotify como al envoltorio GraphQL, considerando el primer punto final: obtener las listas de reproducción por usuario. Estas consultas se implementaron con la opción de caché y sin caché, brindando flexibilidad al usuario en la forma de obtener y almacenar los datos.

En el nivel 2, se tuvieron en cuenta dos puntos finales: el del nivel 1 y el de obtener las canciones de cada lista de reproducción. Se desarrollaron consultas que permiten obtener esta información de manera eficiente, también con las opciones de caché y sin caché.

Se tiene que aclarar que, para la facilidad de la ejecución del experimento se estableció un límite de datos predefinido para cada nivel de consulta. Este enfoque de límite de datos fue parte de la configuración específica utilizada con el propósito de llevar a cabo el experimento de manera controlada.

Reunión de revisión

La reunión de revisión se desarrolló con los siguientes datos generales:

- **Fecha:** 17 de enero de 2023
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Revisión del incremento del producto.

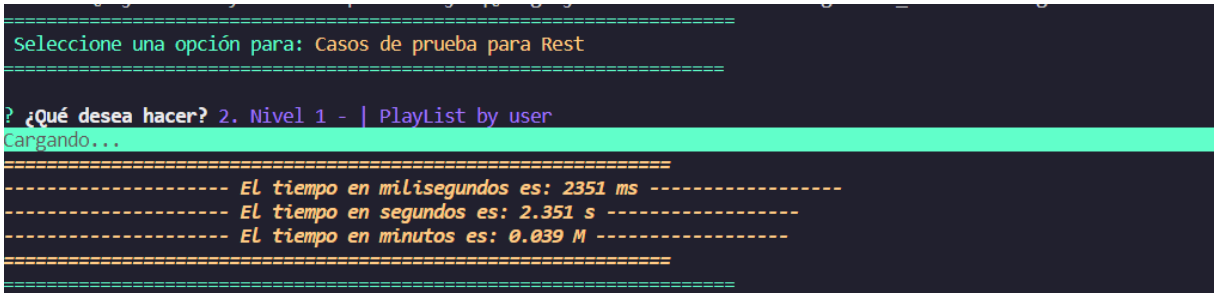
Obtener datos del nivel 1

En el nivel 1, para obtener los datos de la API-REST, se realizó un único llamado para obtener las listas de reproducción por usuario. Se especificó una cantidad de datos predeterminada para recuperar de forma eficiente la información necesaria. Este enfoque

simplifica la obtención de los datos iniciales y garantiza un proceso ágil y eficiente en el nivel 1 de consulta. En la Figura 33 se puede observar un ejemplo de la consulta realizada.

Figura 33

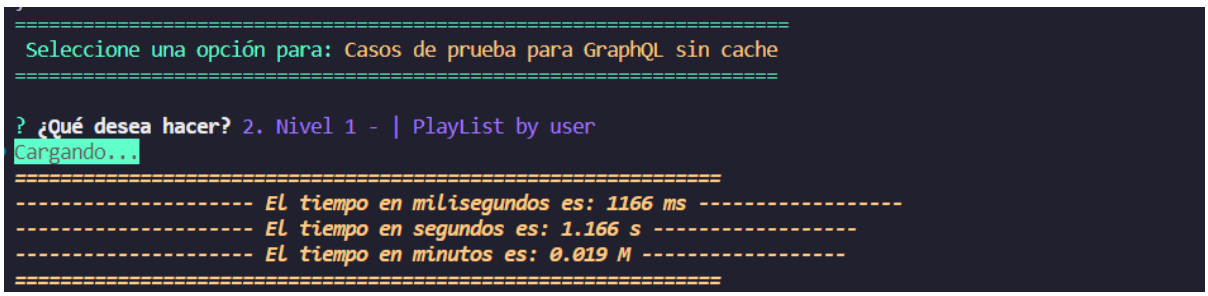
Petición nivel 1 API-REST Spotify



Para la consulta GraphQL en el nivel 1, se crearon los modelos de las consultas correspondientes. Además, se utilizó Apollo Client para realizar la consulta de manera eficiente y precisa. Esto permitió estructurar y ejecutar las consultas en el nivel 1 de forma óptima, utilizando Apollo Client como herramienta principal para interactuar con el envoltorio GraphQL y obtener los resultados deseados. En la Figura 34 se puede observar un ejemplo de la consulta realizada.

Figura 34

Petición nivel 1 Envoltorio GraphQL

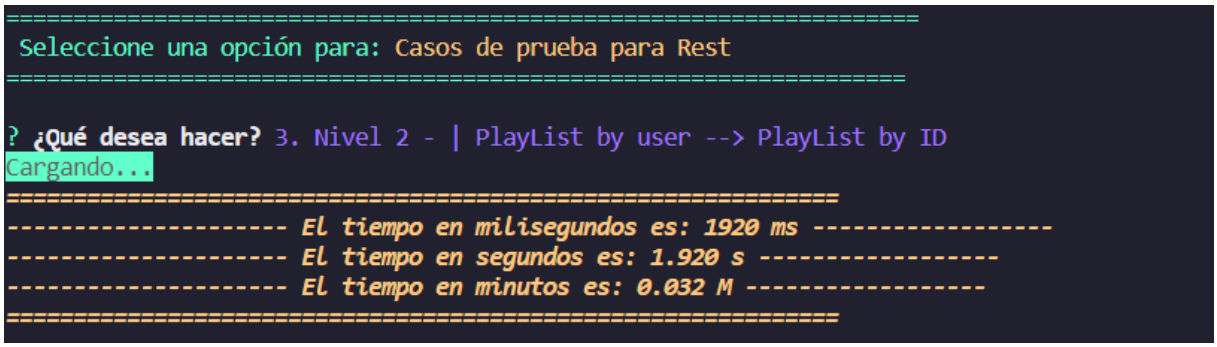


Obtener datos del nivel 2

En el nivel 2, se realizaron consultas adicionales a la API-REST para obtener información detallada. Se obtuvieron las listas de reproducción por usuario, al igual que en el nivel 1, y también se recuperaron las canciones de cada lista. Como se puede observar en la Figura 35.

Figura 35

Petición nivel 2 API-REST Spotify



Para la consulta GraphQL en el nivel 2, se implementaron consultas en GraphQL utilizando los modelos de las consultas creados previamente. Apollo Client se utilizó nuevamente para realizar estas consultas de manera eficiente y precisa. Gracias a Apollo Client, fue posible estructurar y ejecutar las consultas en el nivel 2 de forma óptima, aprovechando las capacidades del envoltorio GraphQL y obteniendo los resultados deseados de manera efectiva. En la Figura 30 se puede observar un ejemplo de la consulta realizada.

Retrospectiva

La reunión de retrospectiva se realizó con los siguientes datos generales:

- **Fecha:** 21 de febrero de 2023
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Analizar lo desarrollado y proponer mejoras.

A continuación, en la Tabla 24 se presentan los resultados de la reunión de retrospectiva que permitirá implementar mejoras para el siguiente Sprint.

Tabla 24

Retrospectiva del sprint 4

Título	Descripción
Aciertos (¿Qué salió bien del sprint?)	Se cumplió la creación de los niveles que permitirán ejecutar el experimento.
Errores (¿Qué no salió como se esperaba?)	Spotify tiene sus límites de llamadas a su API-REST lo que impide el llamado de muchos datos.
Mejoras (¿Qué mejoras se implementará?)	Se plantea añadir el tercer nivel faltante para la ejecución del experimento

2.2.6 Sprint 5

Durante el Sprint 5, se enfocó en la creación de una consulta en el cliente de Node.js para la API-REST de Spotify y el envoltorio GraphQL. Se implemento un nivel adicional de consulta: el nivel 3. En el nivel 3, se trabajó en la obtención de las canciones por álbum.

Reunión de planificación

La reunión de planificación se desarrolló con los siguientes datos generales:

- **Fecha:** 22 de febrero de 2022
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Creación del Sprint Backlog del Sprint 2.

Sprint Backlog del Sprint 5

En la Tabla 25 se observa el Sprint Backlog del presente Sprint, donde se especifican las diferentes tareas a desarrollar con su estimación de tiempo correspondiente.

Tabla 25

Sprint 5 Backlog

Historia de usuario	Fase de desarrollo	Tarea	Estimación Tiempo
HU-11	Desarrollo	Realizar las peticiones a la API-REST con caché y sin caché del nivel 3.	5
	Desarrollo	Realizar las peticiones al envoltorio GraphQL con caché y sin caché del nivel 3.	5
	Pruebas	Probar con diferentes cantidades de datos las peticiones del nivel 3	4
Eventos	Planificación	Especificar las tareas del Sprint actual	3
	Revisión	Revisar los resultados del Sprint	2
	Retrospectiva	Analizar los resultados del Sprint	1
		TOTAL HORAS	20

Revisión del Sprint 5

Como resultado del Sprint 5, se implementó la consulta del nivel 3 en el cliente Node.js. En el nivel 3, que incluye todas las consultas de los niveles anteriores, se agregó un punto final adicional para obtener las canciones por álbum.

Con la implementación de esta opción, los investigadores pueden ejecutar el experimento con diferentes configuraciones de caché, evaluar el rendimiento y comparar los tiempos de ejecución entre consultas con y sin caché.

Reunión de revisión

La reunión de revisión se desarrolló con los siguientes datos generales:

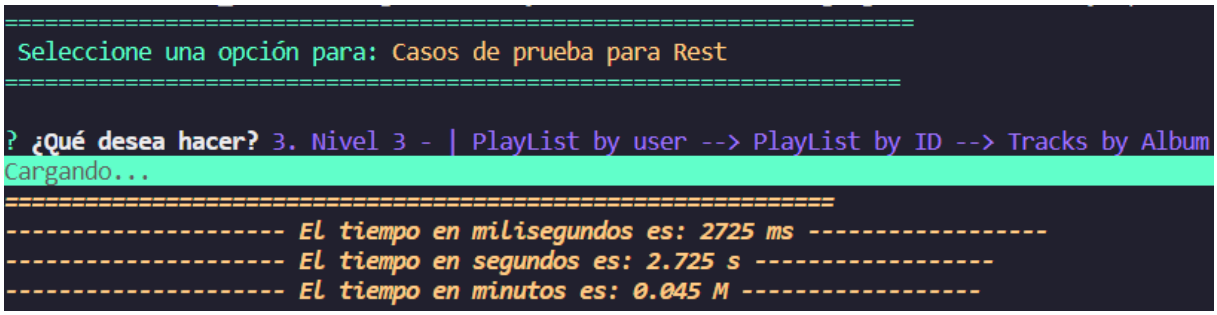
- **Fecha:** 07 de marzo de 2023
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Revisión del incremento del producto.

Obtener datos del nivel 3

En el nivel 3, se llevaron a cabo consultas adicionales en la API-REST de Spotify para obtener datos más específicos. Además de obtener las listas de reproducción y las canciones de cada lista en el nivel 2, se incluyó la consulta para obtener canciones por álbum. En la Figura 36 se puede observar un ejemplo de la consulta realizada.

Figura 36

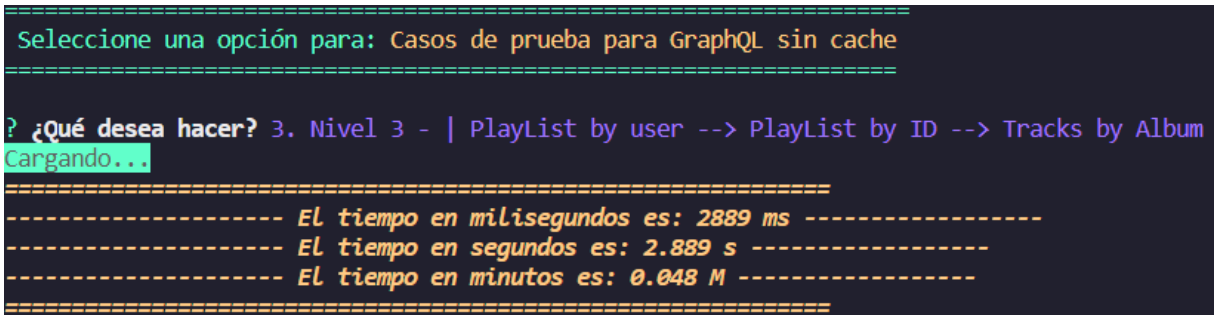
Petición nivel 3 API-REST Spotify.



En el nivel 3, se continuó utilizando los modelos de las consultas previamente creados para la consulta GraphQL. Apollo Client se mantuvo como la herramienta principal para realizar estas consultas de manera eficiente y precisa. Gracias a Apollo Client, se logró estructurar y ejecutar las consultas en el nivel 3 de forma óptima, aprovechando las funcionalidades del envoltorio GraphQL y obteniendo los resultados deseados de manera efectiva. En la Figura 37 se puede observar un ejemplo de la consulta realizada.

Figura 37

Petición nivel 3 Envoltorio GraphQL



Retrospectiva

La reunión de retrospectiva se realizó con los siguientes datos generales:

- **Fecha:** 07 de marzo de 2023
- **Asistentes:** Scrum Master, Product Owner, Team Development
- **Objetivo:** Analizar lo desarrollado y proponer mejoras.

A continuación, en la Tabla 26 se presentan los resultados de la reunión de retrospectiva que permitirá dar por finalizado el proyecto.

Tabla 26

Retrospectiva del sprint 5

Título	Descripción
Aciertos (¿Qué salió bien del sprint?)	Se cumplió la creación de los niveles que permitirán ejecutar el experimento.
Errores (¿Qué no salió como se esperaba?)	Spotify tiene sus límites de llamadas a su API-REST lo que impide el llamado de muchos datos y más en el nivel 3.
Mejoras (¿Qué mejoras se implementará?)	Ninguna mejora

2.3 Pruebas de aceptación.

Las pruebas de aceptación para el envoltorio GraphQL y el cliente Node.js son una parte fundamental del proceso de desarrollo. Estas pruebas tienen como objetivo validar el correcto funcionamiento de las funcionalidades implementadas y asegurar que el envoltorio GraphQL de Spotify y el cliente Node.js cumplan con los requisitos establecidos.

En la Tabla 27 se presenta un conjunto de funcionalidades junto con su correspondiente estado de aceptación por parte del Product Owner. Esta tabla permite visualizar de manera clara y concisa el nivel de aceptación que ha sido asignado a cada funcionalidad por parte del responsable del producto.

Tabla 27

Pruebas de aceptación

Historia de usuario	Nombre	Funcionalidad	Aceptación	
			Si	No
HU-01	Obtener token.	Adquirir el token de forma sencilla y eficaz en el envoltorio GraphQL.	**X**	
HU-02	Obtener listas de reproducción de un usuario en específico.	Obtener las listas de reproducción de un usuario en específico en el envoltorio GraphQL	**X**	
HU-03	Obtener canciones de la lista de reproducción.	Obtener canciones de la lista de reproducción en el envoltorio GraphQL.	**X**	
HU-04	Obtener canciones por álbum.	Obtener canciones de un álbum en el envoltorio GraphQL.	**X**	
HU-05	Obtener canción por el identificador.	Adquirir el detalle de la canción por el identificador en el envoltorio GraphQL.	**X**	

HU-06	Obtener artista por el identificador.	Obtener el artista por el identificador en el envoltorio GraphQL.	**X**
HU-07	Especificar la cantidad de registros por cada nivel.	Crear una consulta con resolvers anidados para obtener hasta 3 respuestas en una consulta.	**X**
HU-08	Escoger el número de caso.	Crear en el cliente Node.js una interfaz de usuario que permita escoger el nivel a probar y presentar el tiempo.	**X**
HU-09	Nivel Nro. 1	Desde el cliente Node.js generar la consultar el Nivel Nro.1 tanto para la API-REST como para el envoltorio GraphQL y presentar el tiempo que se demora la consulta.	**X**
HU-10	Nivel Nro. 2	Desde el cliente Node.js generar la consultar el nivel nro.2 tanto para la API-REST como para el envoltorio GraphQL y presentar el tiempo que se demora la consulta.	**X**
HU-11	Nivel Nro. 3	Desde el cliente Node.js generar la consultar el nivel nro.3 tanto para la API-REST como para el envoltorio GraphQL y presentar el tiempo que se demora la consulta.	**X**

CAPÍTULO 3: Validación de resultados

En el presente capítulo, se lleva a cabo un experimento controlado basado en la guía de *"Experimentación en Ingeniería de Software"* de Wohlin (2012), con el objetivo de responder a la pregunta de investigación: ***PI***: ¿En qué contexto es más apropiado utilizar arquitecturas híbridas que combinan REST y GraphQL? Para medir los efectos, se utiliza la norma ISO/IEC 25023, enfocada en la característica de eficiencia de rendimiento de la calidad del software.

3.1 Entorno experimental

3.1.1 Objetivo del experimento

Comparar la eficiencia de rendimiento entre la arquitectura híbrida API-REST/GraphQL y la API-REST en un entorno Node.js localhost, con relación a la calidad del producto de software.

3.1.2 Factores y tratamientos

El factor por investigar es la arquitectura de software, concretamente en el desarrollo de APIs. Los tratamientos que se aplican a este factor son:

- Arquitectura híbrida REST/GraphQL para el desarrollo de APIs.
- Arquitectura REST para el desarrollo de APIs

Para el desarrollo de la arquitectura híbrida REST/GraphQL, se propone implementar un envoltorio GraphQL que se integre con la API-REST de Spotify. Además, como arquitectura REST se utiliza directamente la API-REST de Spotify como parte de una arquitectura REST.

3.1.3 Variables

En el marco de este estudio, se considera la arquitectura como variable dependiente, mientras que la eficiencia de rendimiento se establece como variable independiente. La arquitectura, en este caso, se refiere a la elección entre la arquitectura híbrida REST/GraphQL y la arquitectura REST pura. Se pretende evaluar y comparar el impacto de estas dos arquitecturas en términos de eficiencia de rendimiento. Esta variable independiente será medida y evaluada utilizando métricas establecidas y la norma ISO/IEC 25023 en relación con la característica de eficiencia de rendimiento. En la Tabla 28 se observa un resumen de las variables utilizadas en el presente experimento.

Tabla 28

Variables usadas en el experimento

Variable Independiente	Arquitectura híbrida REST/GraphQL para el desarrollo de APIs.
	Arquitectura REST para el desarrollo de APIs
Variable Dependiente	Eficiencia de rendimiento (Calidad del producto de software)

A continuación, se presenta la descripción de la medida relacionada con la característica de “Eficiencia de Rendimiento” en la calidad de software según los lineamientos establecidos en la norma ISO/IEC 25023.

Tiempo medio de respuesta

Hace referencia al tiempo en el que se demora una petición al completar, es decir el tiempo medio que emplea para completar un proceso asíncrono. En el contexto del experimento, se aplicó la siguiente función de medición obtenida para capturar y cuantificar el tiempo de respuesta de las solicitudes tanto en la arquitectura híbrida REST/GraphQL como en la arquitectura REST.

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

Donde, $\boldsymbol{A_i}$ es el tiempo para iniciar el trabajo $\boldsymbol{i}$; $\boldsymbol{B_i}$ es el tiempo para completar el trabajo $\boldsymbol{i}$; y $\boldsymbol{n}$ = número de mediciones.

3.1.4 Hipótesis

En esta sección se establecieron las preguntas de investigación que se originan a partir de la pregunta de investigación principal ***(PI)*.**

- **PI_1:** ¿Cuál es el efecto de la adopción de una arquitectura híbrida REST/GraphQL en la calidad de software?

Basándose en esta pregunta de investigación, se determinó las hipótesis para el experimento.

- H_0 **(Hipótesis nula):** No hay diferencia significativa en la calidad de software en términos de tiempo de respuesta al implementar una arquitectura híbrida REST/GraphQL en comparación con una arquitectura REST tradicional.

- H_1 **(Hipótesis alternativa 1):** La implementación de una arquitectura híbrida REST/GraphQL mejora la calidad del software en términos de tiempo de respuesta en comparación con una arquitectura REST tradicional.

- H_2 **(Hipótesis alternativa 2):** La adopción de una arquitectura híbrida REST/GraphQL no presenta mejoras notables en la calidad del software en cuanto al tiempo de respuesta, en comparación con una arquitectura REST convencional.

3.1.5 Diseño

El diseño experimental se enfoca en establecer el ambiente adecuado para llevar a cabo una comparación exhaustiva entre la eficiencia de la arquitectura híbrida REST/GraphQL y la arquitectura REST. Con este fin, se desarrollará un laboratorio computacional que permitirá analizar de manera rigurosa el rendimiento de ambas arquitecturas en diversos escenarios. Además, se han considerado cuatro tareas experimentales que comparten condiciones similares de consulta de datos. La primera tarea implica consumir datos de la API-REST sin utilizar la caché, la segunda tarea implica consumir datos de la API-REST utilizando la caché, la tercera tarea implica consumir datos del envoltorio GraphQL (arquitectura híbrida REST/GraphQL) con caché, y la cuarta tarea implica consumir datos del envoltorio GraphQL (arquitectura híbrida REST/GraphQL) sin utilizar la caché. El diseño del experimento se encuentra detallado en la Tabla 29, donde se especifican los diferentes casos de uso que corresponden a los niveles explicados en la sección 2.2.1. Por ejemplo, el primer caso de uso utiliza únicamente un punto final de la API-REST de Spotify, mientras que el segundo caso de uso utiliza dos puntos finales, y así sucesivamente. Para evaluar la eficiencia de las APIs, se empleó la métrica definida en la sección 3.1.3, la cual permitió obtener el tiempo de respuesta en cada caso.

Tabla 29

Diseño del experimento

Casos de uso	Repeticiones	REST (Sin Caché)	REST (Caché)	GraphQL (Sin Caché)	GraphQL (caché)
CU-01	3	**Registros:** 1,50,250, 3150,32800,100000	**Registros:** 1,50,250, 3150,32800,100000	**Registros:** 1,50,250, 3150,32800,100000	**Registros:** 1,50,250, 3150,32800,100000
CU-02	3	**Registros:** 1,36,256, 3136,32761,99856	**Registros:** 1,36,256, 3136,32761,99856	**Registros:** 1,36,256, 3136,32761,99856	**Registros:** 1,36,256, 3136,32761,99856
CU-03	3	**Registros:** 1,27,216, 3375,32768,97336	**Registros:** 1,27,216, 3375,32768,97336	**Registros:** 1,27,216, 3375,32768,97336	**Registros:** 1,27,216, 3375,32768,97336

3.1.6 Tareas experimentales

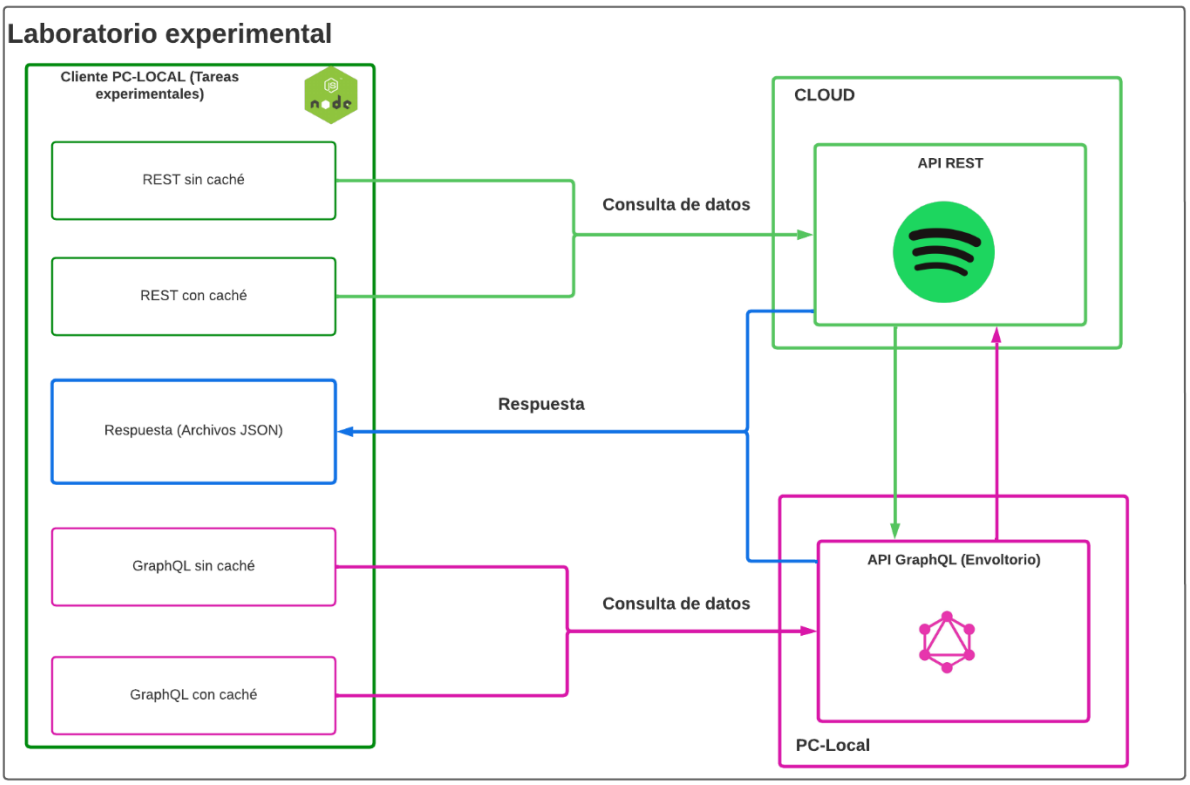
Las opciones implementadas para realizar las consultas de datos en el CAPITULO 2 (REST con caché, REST sin caché, GraphQL con caché y GraphQL sin caché) han sido utilizadas como tareas experimentales en el presente estudio, las tareas experimentales son las siguientes.

- Consultar datos al API-REST de Spotify sin caché.
- Consultar datos al API-REST de Spotify con caché.
- Consultar datos al API GraphQL (envoltorio) sin caché.
- Consultar datos al API GraphQL (envoltorio) con caché.

Estas tareas experimentales han permitido la creación de un laboratorio computacional, tal como se ilustra en la Figura 38.

Figura 38

Arquitectura del laboratorio experimental



3.1.7 Instrumentación

En esta sección se detallan los componentes, frameworks y tecnologías empleadas en el entorno de laboratorio computacional.

Descripción de PC-local:

- Sistema Operativo Windows 10 Home 64-bit
- Procesador AMD Ryzen 5 2500U con Radeon Vega Mobile Gfx 2.00 Ghz
- Memoria Ram 16 GB (14,9 GB usable)

Descripción de conexión a Internet:

- **Velocidad de bajada:** 244.91 Mbps
- **Velocidad de subida:** 162.5 Mbps

Ambiente de Desarrollo:

- **Entorno de desarrollo:** Visual Studio Code v1.79.2
- **Lenguaje de programación**: JavaScript
- **Entorno de tiempo de ejecución de JavaScript:** NodeJS v19.1.0
- **Manejador de paquetes:** npm v8.19.3

- **Librerías npm para el API GRAPHQL:** apollo-server v3.11.1, graphql v15.8.0, @/apollo/cliente v3.7.14, apollo-link-rest v0.9.0, graphql-tag v2.12.6, qs v6.11.2
- **Librerías npm para el cliente Node.JS:** @apollo/client v3.7.7, @lifeomic/axios-fect v3.0.1, apollo-link v1.2.14, apollo-link-http v1.5.17, pollo-link-rest v0.9.0, axios v1.4.0, colors v1.4.0, dotenv v16.0.3, graphql-tag v2.12.6, inquirer v9.1.4.

Recolección y análisis de datos:

- Microsoft Excel 365
- IBM SPSS V27.0

3.1.8 Recolección de datos

A continuación, en la Tabla 30 se exhibe el esquema de datos empleado en el archivo Excel para el registro de los resultados.

Tabla 30

Estructura de recolección de datos

Campo	Descripción
Nro.	Número de la muestra (autonumérico)
Caso de uso	Caso de uso que se ejecuta
Complejidad	Nivel de complejidad de la consulta del caso de uso
Nro. Registros	Cantidad de registros consultados en el caso de uso
Repetición	Número de repetición de ejecución (valores entre 1 y 3)
Enfoque arquitectónico	Rest caché, GraphQL caché, Rest sin cache, GraphQL sin caché
Tiempo	Tiempo de respuesta de la ejecución del caso de uso medido en milisegundos.

3.2 Ejecución del experimento

Durante la ejecución del experimento, se siguieron los pasos establecidos en el diseño experimental para comparar la eficiencia de la arquitectura híbrida REST/GraphQL y la arquitectura REST de la sección 3.1.

3.2.1 Explicación de la ejecución

En el transcurso del experimento, se llevaron a cabo tres repeticiones para cada número de registros necesarios en cada caso de uso, tanto para las arquitecturas con caché como para las arquitecturas sin caché. Este enfoque garantizó obtener mediciones más precisas y representativas de los tiempos de respuesta.

Cada repetición consistió en ejecutar la consulta correspondiente con el número específico de registros correspondiente y registrar el tiempo de respuesta obtenido. Se mantuvieron las mismas condiciones experimentales en cada repetición para asegurar la consistencia de los resultados. Por ejemplo, en el caso de uso uno, que involucraba la obtención de listas de reproducción de un usuario, se estableció la necesidad de obtener 1, 50, 250, 3.150, 32.800 y 100.000 registros.

En cada repetición, se ejecutó la consulta correspondiente para obtener las listas de reproducción con el número de registros especificado y luego se registró los tiempos de respuesta obtenidos. Este proceso se repitió dos veces más, obteniendo tres conjuntos de tiempos de respuesta para el caso de uso uno por cada número de registros necesarios. Al realizar estas tres repeticiones, se obtuvieron múltiples conjuntos de datos que permitieron calcular promedios.

3.2.2 Recolección de datos

A medida que se ejecutaba el entorno localhost en Node.js se tomaban los tiempos respectivos (milisegundos, segundos y minutos) y se pasaban directamente al archivo de Microsoft Excel 365 para su tabulación y posterior análisis. En la Figura 39, Figura 40 y Figura 41 se muestra la ejecución del caso uno para 256 registros en la arquitectura REST sin utilizar caché.

Figura 39

Primera repetición de ejecución para el caso de uso 1

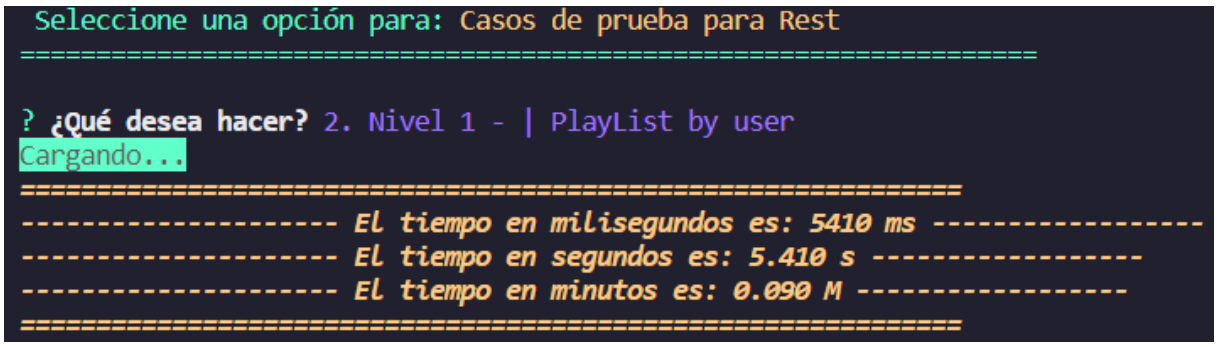


Figura 40

Segunda repetición de ejecución para el caso de uso 1

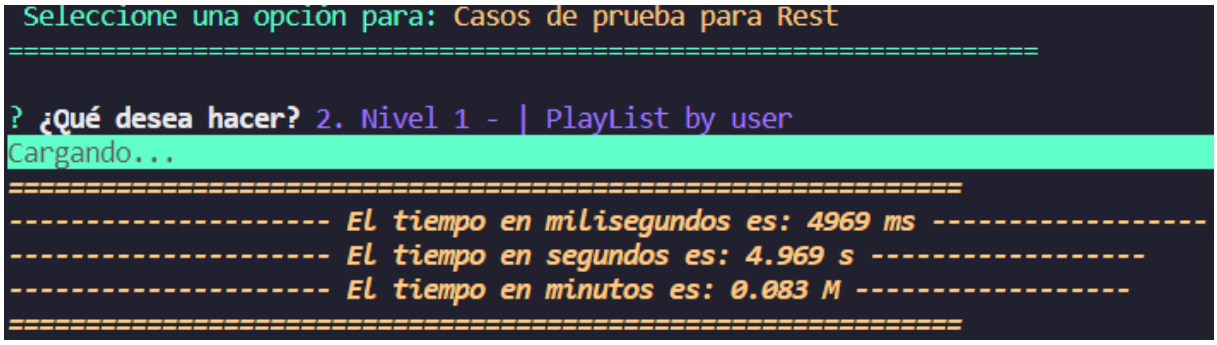
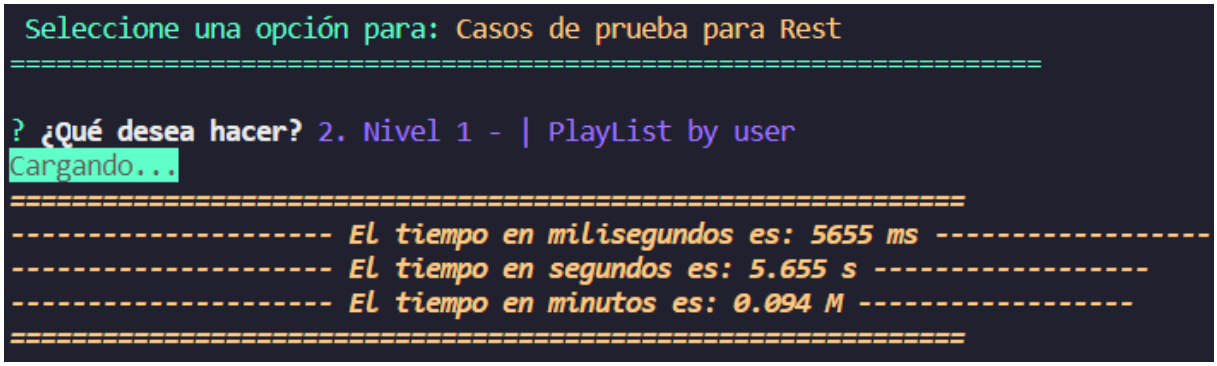


Figura 41

Tercera repetición de ejecución para el caso de uso 1



Luego de obtener los datos, se procedió a registrar los datos en la hoja de Excel, tal y como se muestra en la Tabla 31.

Tabla 31

Tabulación de los registros obtenidos del caso de uso 1

Nro.	Caso de uso	Complejidad	Nro. Registros	Repetición	Enfoque arquitectónico	Tiempo (ms)	Tiempo (s)	Tiempo (min)
1	CU-01	1	256	1	Rest sin caché	5410	5.410	0.090
2	CU-01	1	256	2	Rest sin caché	4969	4.969	0.083
3	CU-01	1	256	3	Rest sin caché	5655	5.665	0.094

3.3 Análisis de los Resultados

En esta sección, se llevó a cabo un análisis estadístico utilizando la herramienta SPSS para examinar los resultados obtenidos en relación con la característica de eficiencia de la norma ISO/IEC 25023, específicamente en términos del tiempo medio de respuesta.

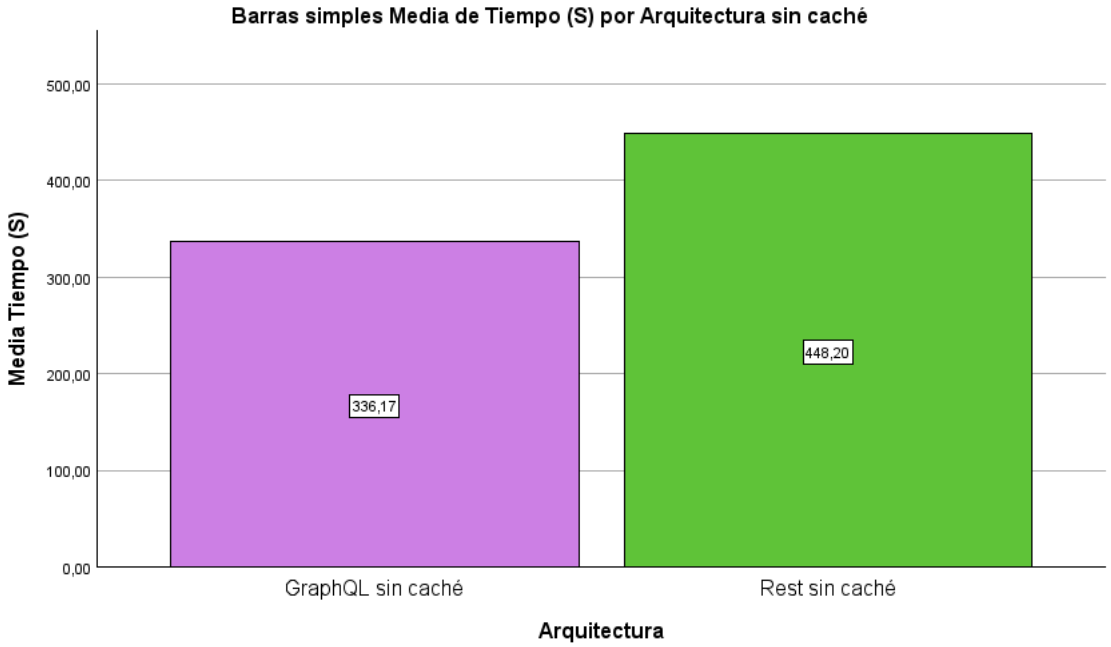
3.3.1 Análisis estadísticos de la eficiencia por cada caso

Caso de uso 1 sin caché

En el primer caso (Consultar la lista de reproducción de un usuario especifico) se obtuvo un valor promedio de respuesta de 336,17 segundos para la arquitectura GraphQL/REST sin caché y un valor de 448,20 segundos para la arquitectura REST sin caché, tal y como se muestra en la Figura 42.

Figura 42

Medias caso de uso 1 sin caché



Como se puede observar en la Figura 42, el tiempo promedio de respuesta de la arquitectura REST es más alto que el de GraphQL/REST, siendo GraphQL/REST más eficiente por un 59,16%, tal como se puede apreciar en la Tabla 32.

Tabla 32

Porcentaje de eficiencia caso de uso 1 sin caché

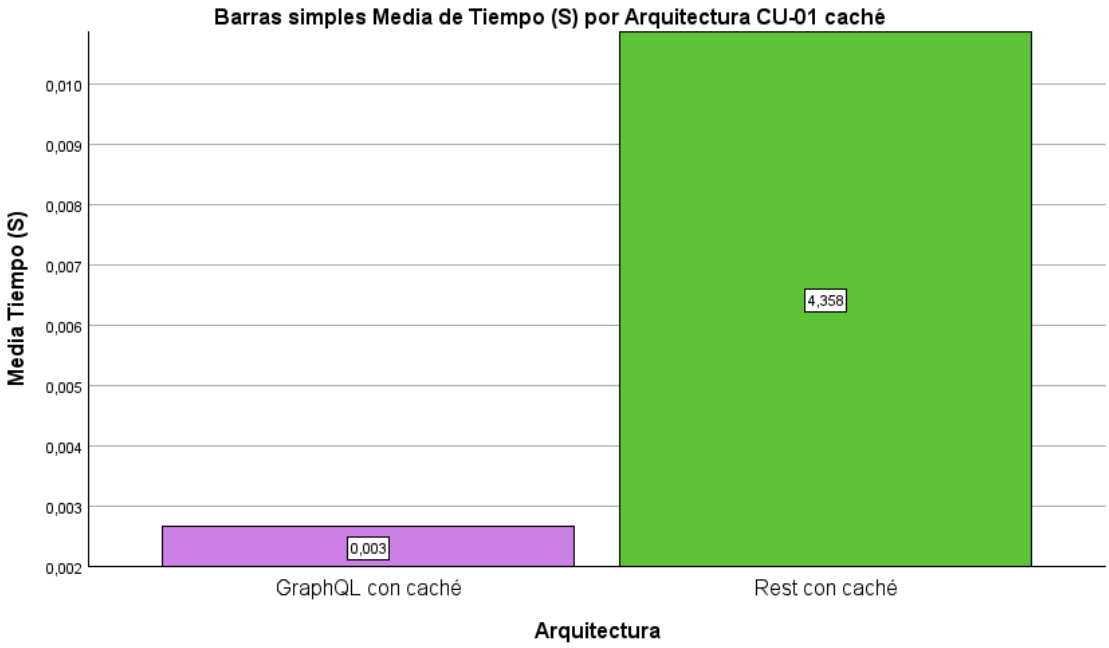
Arquitectura	**Media (Segundos)**
GraphQL/REST sin caché	336,17
REST sin caché	448,20
Eficiencia GraphQL/REST	**25%**

Caso de uso 1 con caché

En el mismo caso de uso 1, para la opción de la arquitectura GraphQL/REST con caché se obtuvo un promedio de 0,003 segundos y para la arquitectura REST con caché un valor de 4,358 segundos, como se puede observar en la Figura 43.

Figura 43

Medias caso de uso 1 con caché



Según se puede constatar en la Figura 43, el tiempo promedio de respuesta de la arquitectura REST es mucho más alto que el de GraphQL/REST, siendo GraphQL/REST más eficiente por un 99,94%, tal como se puede apreciar en la Tabla 33.

Tabla 33

Porcentaje de eficiencia caso de uso 1 con caché

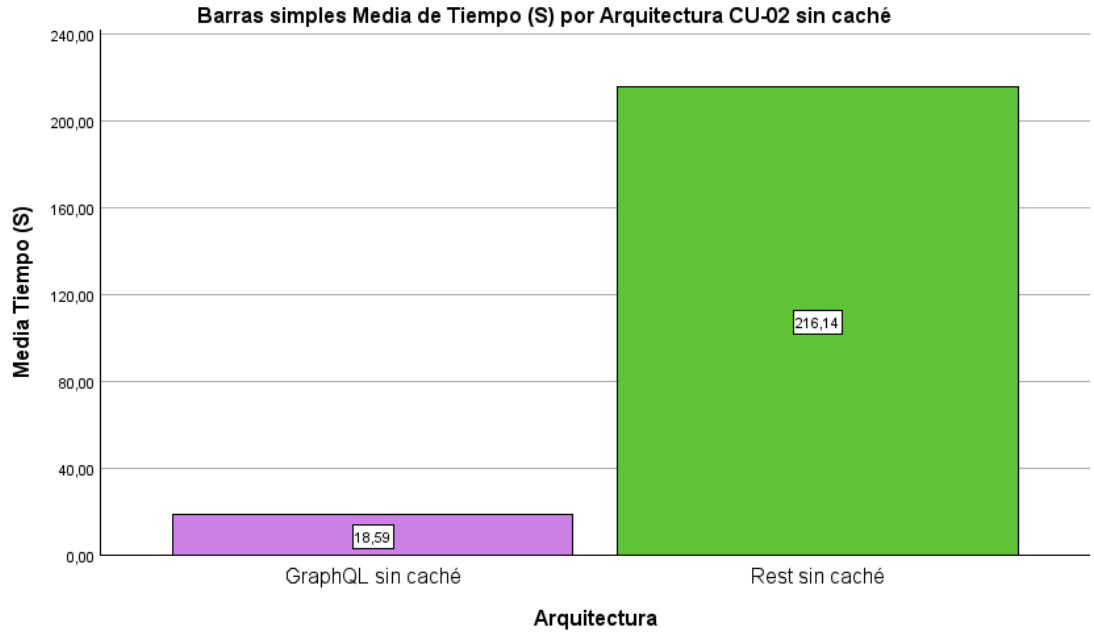
Arquitectura	**Media (Segundos)**
GraphQL/REST con caché	0,003
REST con caché	4,358
Eficiencia GraphQL/REST	**99,94%**

Caso de uso 2 sin caché

En el segundo caso (Consultar la lista de reproducción de un usuario especifico, seguido de consultar las canciones de las listas de reproducción) se obtuvo un valor promedio de respuesta de 18,59 segundos para la arquitectura GraphQL/REST sin caché y un valor de 216,14 segundos para la arquitectura REST sin caché, tal y como se muestra en la Figura 44.

Figura 44

Medias caso de uso 2 sin caché



De manera evidente, en la Figura 44 el tiempo promedio de respuesta de la arquitectura REST es mucho más alto que el de GraphQL/REST, siendo GraphQL/REST más eficiente por un 91,40%, tal como se puede apreciar en la Tabla 34.

Tabla 34

Porcentaje de eficiencia caso de uso 2 sin caché

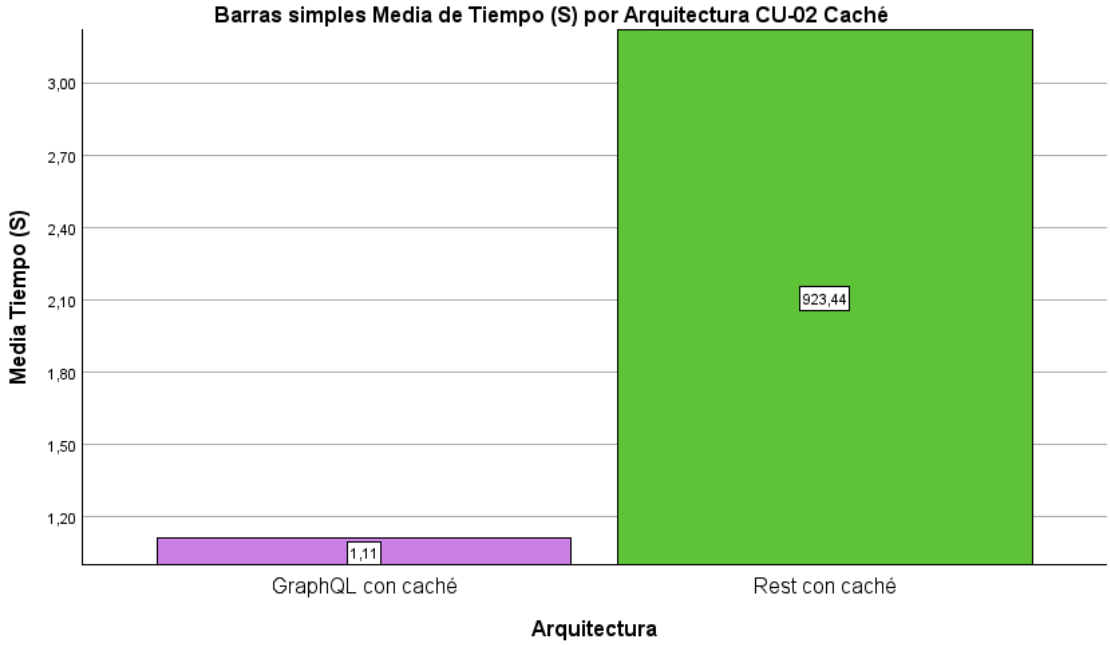
Arquitectura	**Media (Segundos)**
GraphQL/REST sin caché	18,59
REST sin caché	216,14
Eficiencia GraphQL/REST	**91,40%**

Caso de uso 2 con caché

En el mismo caso de uso 2, para la opción de la arquitectura GraphQL/REST con caché se obtuvo un promedio de 1,11 segundos y para la arquitectura REST con caché un valor de 923,44 segundos, como se puede notar en la Figura 45.

Figura 45

Medias caso de uso 2 con caché



De acuerdo con lo evidenciado en la Figura 45, el tiempo promedio de respuesta de la arquitectura REST es mucho más alto que el de GraphQL/REST, siendo GraphQL/REST más eficiente por un 99,88%, tal como se puede apreciar en la Tabla 35.

Tabla 35

Porcentaje de eficiencia caso de uso 2 con caché

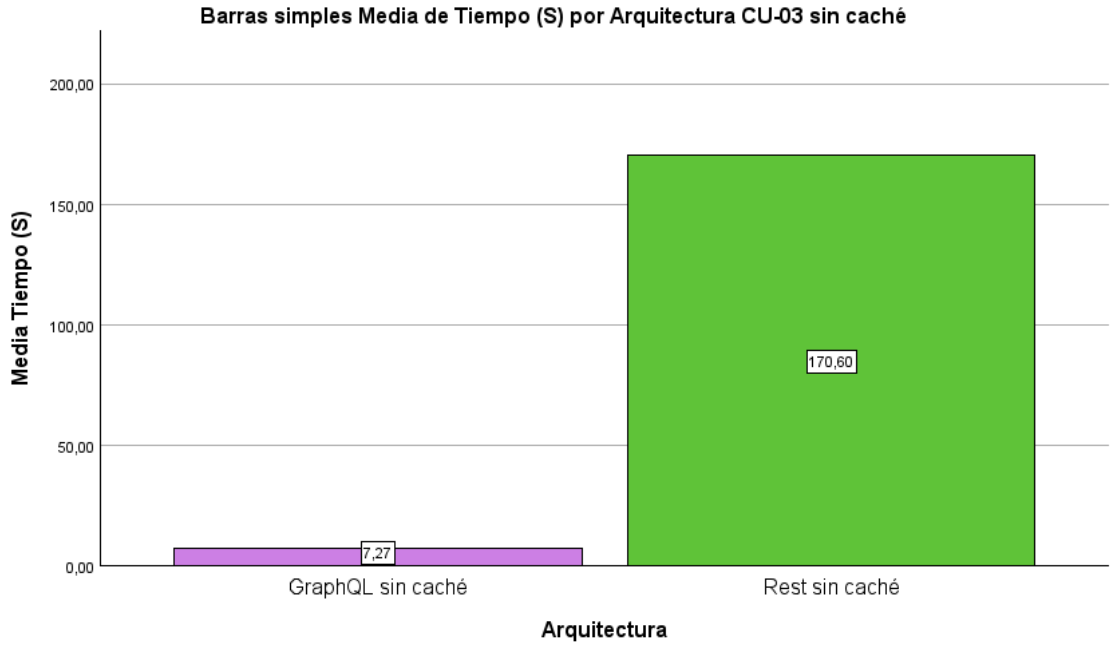
Arquitectura	**Media (Segundos)**
GraphQL/REST sin caché	1,11
REST sin caché	923,44
Eficiencia GraphQL/REST	**99,88%**

Caso de uso 3 sin caché

En el tercer caso (Consultar la lista de reproducción de un usuario especifico, seguido de consultar las canciones de las listas de reproducción y consultar canciones por álbum) se obtuvo un valor promedio de respuesta de 7,27 segundos para la arquitectura GraphQL/REST sin caché, y un valor de 170,60 segundos para la arquitectura REST sin caché, tal y como se muestra en la Figura 46.

Figura 46

Medias caso de uso 3 sin caché



De acuerdo con lo evidenciado en la Figura 46, el tiempo promedio de respuesta de la arquitectura REST es mucho más alto que el de GraphQL/REST, siendo GraphQL/REST más eficiente por un 95,74%, tal como se puede apreciar en la Tabla 36.

Tabla 36

Porcentaje de eficiencia caso de uso 3 sin caché

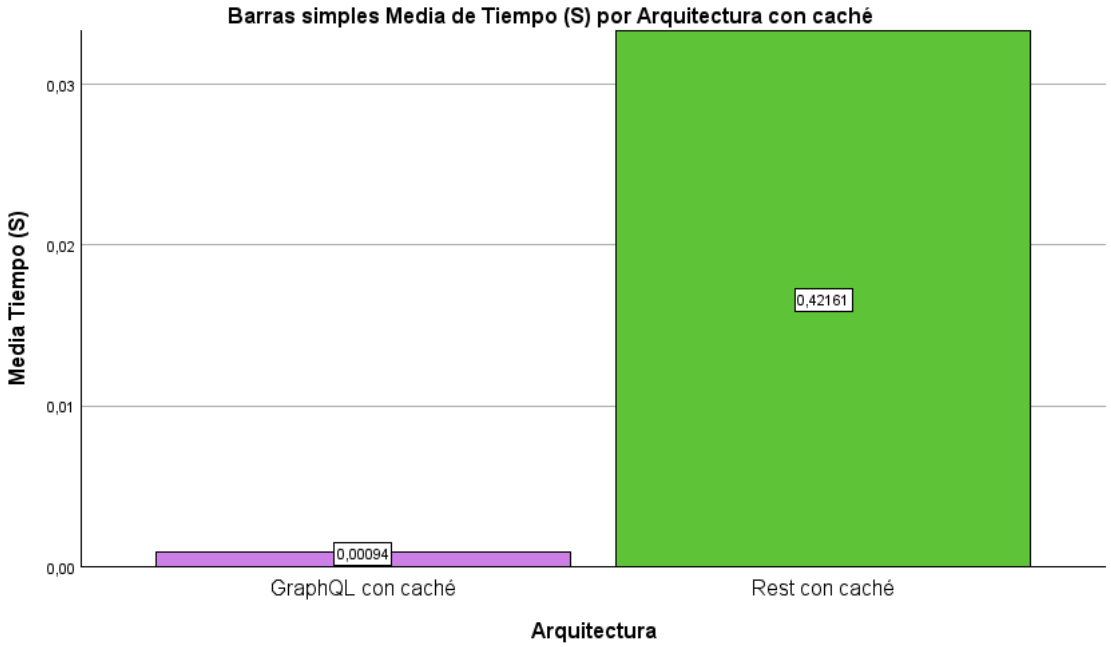
Arquitectura	**Media (Segundos)**
GraphQL/REST sin caché	7,27
REST sin caché	170,60
Eficiencia GraphQL/REST	**95,74%**

Caso de uso 3 con caché

En el mismo caso de uso 3, para la opción de la arquitectura GraphQL/REST con caché se obtuvo un promedio de 0,00094 segundos y para la arquitectura REST con caché un valor de 0,42161 segundos, como se puede notar en la Figura 47.

Figura 47

Medias caso de uso 3 con caché



A simple vista, en la Figura 47 el tiempo promedio de respuesta de la arquitectura REST es mucho más alto que el de GraphQL/REST, siendo GraphQL/REST más eficiente por un 99,78%, tal como se puede apreciar en la Tabla 37.

Tabla 37

Porcentaje de eficiencia caso de uso 3 con caché

Arquitectura	**Media (Segundos)**
GraphQL/REST con caché	0,00094
REST con caché	0,42161
Eficiencia GraphQL/REST	**99,78%**

3.3.2 Resumen y media total por arquitectura

Luego de mostrar los resultados por cada arquitectura, en la Figura 48 se puede observar un resumen de los promedios obtenidos por arquitectura sin caché y caso de uso. De la misma manera, en la Figura 49 se evidencia el resumen de los promedios obtenidos por arquitectura con caché y caso de uso.

Figura 48

Resumen de medias de los casos de uso sin caché

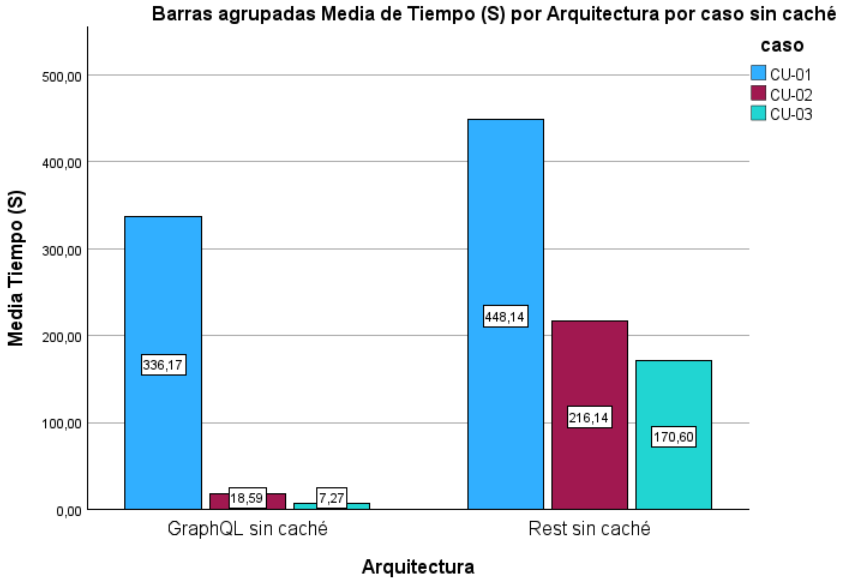
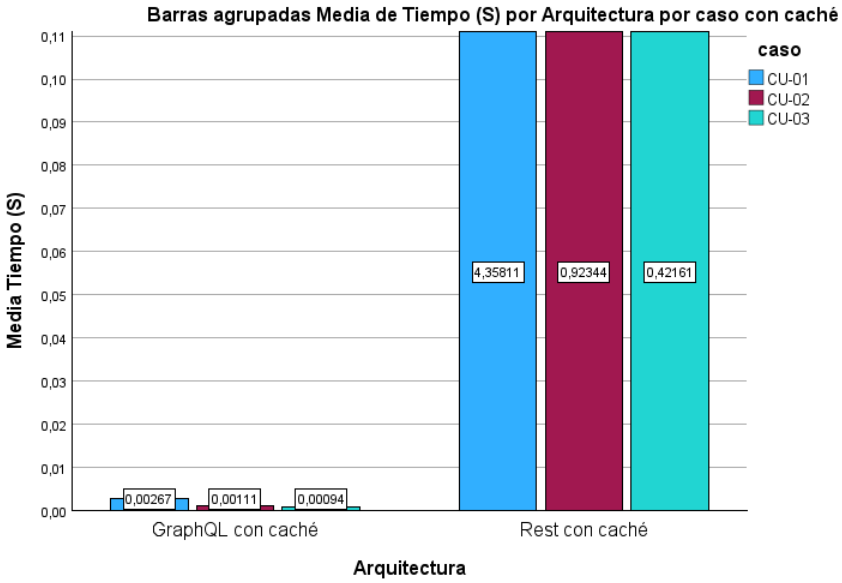


Figura 49

Resumen de medias de los casos de uso con caché



Por último, en la Tabla 38 se muestra la media total por cada arquitectura, tanto con caché como sin caché teniendo en cuenta un intervalo de confianza del 95%. Acto seguido, en la Figura 50 y Figura 51 se puede apreciar más datos obtenidos del analisis descriptivo.

Tabla 38

Analisis descriptivo por arquitectura

Arquitectura	**Media (Segundos)**	**Intervalo de confianza (límite inferior, límite superior)**	**Desviación estándar**
GraphQL/REST con caché	0,00157	[0,0096; 0,00219]	0,002245
REST con caché	1,90106	[0,58346; 3,21865]	4,827276
GraphQL/REST sin caché	120,67659	[26,07048; 215,28270]	346,609242
REST sin caché	278,31380	[144,05883; 412,56876]	491,871093

Figura 50

Análisis descriptivo por arquitectura GraphQL/REST

Descriptivos

Arquitectura				Estadístico	Error estándar
GraphQL con caché	Tiempo (S)	Media		,00157	,000306
		95% de intervalo de confianza para la media	Límite inferior	,00096	
			Límite superior	,00219	
		Media recortada al 5%		,00119	
		Mediana		,00100	
		Varianza		,000	
		Desv. estándar		,002245	
		Mínimo		,000	
		Máximo		,012	
		Rango		,012	
		Rango intercuartil		,001	
		Asimetría		3,331	,325
		Curtosis		11,796	,639
GraphQL sin caché	Tiempo (S)	Media		120,67659	47,167543
		95% de intervalo de confianza para la media	Límite inferior	26,07048	
			Límite superior	215,28270	
		Media recortada al 5%		52,56044	
		Mediana		3,37900	
		Varianza		120137,966	
		Desv. estándar		346,609242	
		Mínimo		,736	
		Máximo		1590,105	
		Rango		1589,369	
		Rango intercuartil		26,346	
		Asimetría		3,457	,325
		Curtosis		11,489	,639

Figura 51

Análisis descriptivo total por arquitectura REST

Rest con caché	Tiempo (S)	Media		1,90106	,656909
		95% de intervalo de confianza para la media	Límite inferior	,58346	
			Límite superior	3,21865	
		Media recortada al 5%		1,00688	
		Mediana		,03950	
		Varianza		23,303	
		Desv. estándar		4,827276	
		Mínimo		,001	
		Máximo		20,934	
		Rango		20,933	
		Rango intercuartil		,615	
		Asimetría		3,144	,325
		Curtosis		9,321	,639
Rest sin caché	Tiempo (S)	Media		278,31380	66,935178
		95% de intervalo de confianza para la media	Límite inferior	144,05883	
			Límite superior	412,56876	
		Media recortada al 5%		201,67258	
		Mediana		26,97800	
		Varianza		241937,172	
		Desv. estándar		491,871093	
		Mínimo		1,140	
		Máximo		2135,729	
		Rango		2134,589	
		Rango intercuartil		310,662	
		Asimetría		2,327	,325
		Curtosis		5,478	,639

3.3.3 Análisis de impactos total

De acuerdo con el análisis descriptivo realizado, en la Tabla 39 se hace evidente que la arquitectura GraphQL/REST con caché es más eficiente en tiempo medio de respuesta en un 99,91% que la arquitectura REST tradicional con caché, y de la misma manera la arquitectura GraphQL/REST sin caché es más eficiente en un 56,64% que la arquitectura REST tradicional sin caché.

Tabla 39

Porcentaje de eficiencia caso de uso 3 con caché

Arquitectura	**Media (Segundos)**	**Arquitectura**	**Media (Segundos)**
GraphQL/REST con caché	**0,00157**	**GraphQL/REST sin caché**	**120,67659**
REST con caché	1,90106	REST sin caché	278,31380
Eficiencia GraphQL/REST con caché	**99,91%**	**Eficiencia GraphQL/REST sin caché**	**56,64%**

CONCLUSIONES

- El marco teórico de esta tesis ha proporcionado una base sólida de conocimientos sobre los fundamentos de arquitecturas orientadas a microservicios, arquitecturas de servicios REST y lenguaje de consultas GraphQL. Estos conocimientos han sentado las bases para la realización del estudio experimental y el análisis de los resultados obtenidos.

- Utilizando los conocimientos adquiridos, se diseñó y desarrolló un envoltorio GraphQL para la API-REST de Spotify. Esta implementación permitió obtener de forma anidada datos variados provenientes de diferentes endpoints de la API-REST de Spotify. Además, se creó un cliente Node.js que facilitó la ejecución de consultas y la medición de tiempos tanto en el envoltorio GraphQL como en la API-REST original de Spotify.

- Una vez creado el envoltorio GraphQL y el cliente Node.js para medir el rendimiento de ambos enfoques, se evaluó mediante un experimento computacional siguiendo la guía de Wohlin, donde se planteó la siguiente pregunta de investigación (***PI)***: ¿En qué contextos es más adecuado utilizar arquitecturas híbridas que combinan REST y GraphQL? Para responder a esta pregunta, se diseñaron y se implementaron tres casos de uso en el laboratorio experimental. Estos casos de uso permitieron comparar

la eficiencia de la arquitectura híbrida GraphQL/REST (envoltorio) y la arquitectura REST, utilizando como métrica el "tiempo medio de respuesta" definido por la norma ISO/IEC 25023.

- Finalmente, se analizó los resultados del experimento computacional, en donde, por un lado, se obtuvo que la API GraphQL es 56,64% más eficiente que la API-REST tradicional de Spotify cuando no se utiliza caché, por otro lado, al momento de utilizar caché se pudo observar que la API GraphQL es hasta 99,91% más eficiente que la API-REST. Por lo cual, se acepta la hipótesis alternativa 1 del estudio: “La implementación de una arquitectura híbrida REST/GraphQL mejora la calidad del software en términos de tiempo de respuesta en comparación con una arquitectura REST tradicional”.

RECOMENDACIONES

- Se recomienda explorar la posibilidad de implementar el envoltorio GraphQL en un lenguaje de programación diferente y repetir el experimento para evaluar si se obtienen resultados similares. Esto permitiría corroborar la validez de los hallazgos y proporcionar una perspectiva más amplia sobre el rendimiento de la arquitectura híbrida REST/GraphQL en diferentes entornos.

- Se sugiere replicar el mismo experimento, pero esta vez centrándose en medir el consumo de memoria de la arquitectura híbrida REST/GraphQL en comparación con la arquitectura REST tradicional. Esto proporciona información adicional sobre el rendimiento y eficiencia de ambos enfoques en términos de usos de recursos. Al evaluar el consumo de memoria, se podrán identificar posibles diferencias significativas entre las dos arquitecturas y comprender cómo afecta el uso de GraphQL en la asignación de recursos.

- Se recomienda llevar a cabo una replicación del experimento en ambientes de producción controlados, donde se simulen condiciones similares a las que se encuentran en sistemas distribuidos reales. Esto implica realizar pruebas en entornos con una mayor carga de usuarios, volumen de datos y tráfico de red, para obtener una perspectiva más precisa del rendimiento y la eficiencia de la arquitectura híbrida REST/GraphQL en escenarios más cercanos a la realidad.

BIBLIOGRAFÍA

Alfonso, D., & Contreras, B. (2018). Tecnología, Investigación y Academia TIA Arquitectura de microservicios Microservice architecture. *Tecnología Investigación y Academia*, *6*, 36–46. http://revistas.udistrital.edu.co/ojs/index.php/tia/issue/archive

Andrés, C., & Solano, R. (2019). *ANÁLISIS COMPARATIVO ENTRE LOS ESTÁNDARES ORIENTADO A SERVICIOS WEB SOAP, REST Y GRAPHQL.*

Angele, K., Meitinger, M., Bußjäger, M., Föhl, S., & Fensel, A. (2022). GraphSPARQL: A GraphQL Interface for Linked Data. *Proceedings of the ACM Symposium on Applied Computing*, 778–785. https://doi.org/10.1145/3477314.3507655

Apollo. (n.d.). *Apollo Docs - Apollo GraphQL Docs*. Retrieved April 22, 2023, from https://www.apollographql.com/docs/

Apollo Community. (n.d.). *Apollo GraphQL - Apollo GraphQL community*. Retrieved April 22, 2023, from https://community.apollographql.com/

Ballesteros, L. (2021). IMPLEMENTACIÓN Y PRÁCTICA DE SCRUM EN LA ASIGNATURA DE FORMULACIÓN Y EVALUACIÓN DE PROYECTOS EN LA FACULTAD DE CIENCIAS ECONÓMICAS Y ADMINISTRATIVAS DE LA UNIVERSIDAD EL BOSQUE. *Panorama*, *15*(29), 127–140. https://doi.org/10.15765/pnrm.v15i29.2538

Barsoti, N., & Gibertoni, D. (2020). IMPACTO QUE O SEQUELIZE TRAZ PARA O DESENVOLVIMENTO DE UMA API CONSTRUÍDA EM NODE.JS COM EXPRESS.JS. *Revista Interface Tecnológica*, *17*(2), 231–243. https://doi.org/10.31510/infa.v17i2.964

Baumgart, R., Hummel, M., Holten, R., & Research, C. (2015). *PERSONALITY TRAITS OF SCRUM ROLES IN AGILE SOFTWARE DEVELOPMENT TEAMS-A QUALITATIVE ANALYSIS*. http://aisel.aisnet.org/ecis2015_crhttp://aisel.aisnet.org/ecis2015_cr/16

Carrizo, D., & Alfaro, A. (2018). Método de aseguramiento de la calidad en una metodología de desarrollo de software: un enfoque práctico. *Revista Chilena de Ingeniería*, *26*(1), 114–129.

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2016). *Microservices: yesterday, today, and tomorrow*. http://arxiv.org/abs/1606.04036

Express. (n.d.). *Express - Infraestructura de aplicaciones web Node.js*. Retrieved April 22, 2023, from https://expressjs.com/es/

Fernández, E., Pollo, M., Amatriain, H., Dieste, O., Pesado, P., & García, R. (2010). *Aplicabilidad de los Métodos de Síntesis Cuantitativa de Experimentos en Ingeniería de Software*.

Fielding, R. (2000). *Architectural Styles and the Design of Netwrok-based Software Architectures*.

Gómez, O. S. (2010). *Clasificación de Replicaciones para la Síntesis de Experimentos en Ingeniería*. https://www.researchgate.net/publication/269168777

González, L., & Rafael, J. (2021). *Arquitectura basada en microservicios para un sistema de producción de hidrocarburos en la empresa BLC Venezuela, C.A.*

GraphQL. (n.d.). *Queries and Mutations | GraphQL*. Retrieved April 17, 2023, from https://graphql.org/learn/queries/

Halili, F., & Ramadani, E. (2018). Web Services: A Comparison of Soap and Rest Services. *Modern Applied Science*, *12*(3), 175. https://doi.org/10.5539/mas.v12n3p175

Hostinger. (n.d.). *Qué es Node.js - Guía para principiantes (+ casos de uso)*. Retrieved April 22, 2023, from https://www.hostinger.es/tutoriales/que-es-node-js#%C2%BFQue_es_Nodejs

Huerta Guijarro, J. (2015). *UNIVERSITATJAUME I DE CASTELLÓN*.

ISO25000. (n.d.). *NORMAS ISO 25000*. Retrieved April 23, 2023, from https://iso25000.com/index.php/normas-iso-25000

Jiménez Builes, J. A., Ramírez Bedoya, D. L., & Branch Bedoya, J. W. (2019). Metodología de desarrollo de software para plataformas educativas robóticas usando ROS-XP. *Revista Politécnica*, *15*(30), 55–69. https://doi.org/10.33571/rpolitec.v15n30a6

López. (2017). *Arquitectura de software basada en microservicios para desarrollo de aplicaciones web de la asamblea nacional*.

Marii, B., & Zholubak, I. (2022). FEATURES OF DEVELOPMENT AND ANALYSIS OF REST SYSTEMS. *ADVANCES IN CYBER-PHYSICAL SYSTEMS*, *7*(2). https://doi.org/10.23939/acps2022.__.___

Mascheroni, M. A., & Irrazábal, E. (2016). *Framework para la creación y ejecución de pruebas automatizadas sobre servicios REST*.

Microsoft. (2023). *Web API design best practices - Azure Architecture Center | Microsoft Learn*. https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

Molina, B., Cevallos, H., & Dávila, J. (2018). *Metodologias agiles frente a las tradicionales en el proceso de desarrollo de software*.

Moreno, D. (2019). *Generación automática de APIs GraphQL a partir de mappings RML con MongoDB*.

Nazareno, R., Gonnet, S., & Leone, H. (2015). *Trazabilidad de Procesos Scrum*.

Nebel Andrés. (2018). *Arquitectura de Microservicios para Plataformas de Integración*.

Newman, S. (2015). *Building Microservices*. http://safaribooksonline.com

Oracle. (2021). *Creating RESTful Web Services in NetBeans 7: Part 1*. https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/RESTfulWebServices/RESTfulWebservices.htm

Pressman Roger. (2002). *Software Engineering: A Practitioner's Approach* (Vol. 6). McGraw-Hill. www.mhhe.com/pressman.

RedHat. (2019). *¿Qué es GraphQL?* https://www.redhat.com/es/topics/api/what-is-graphql

Richardson, L., & Ruby, S. (2007). *RESTful Web Services*.

Schmidt, D. C. (2000). *Pattern-oriented software architecture. Volume 2, Patterns for concurrent and networked objects*. Wiley.

Spotify. (n.d.). *Web API | Spotify for Developers*. Retrieved April 16, 2023, from https://developer.spotify.com/documentation/web-api

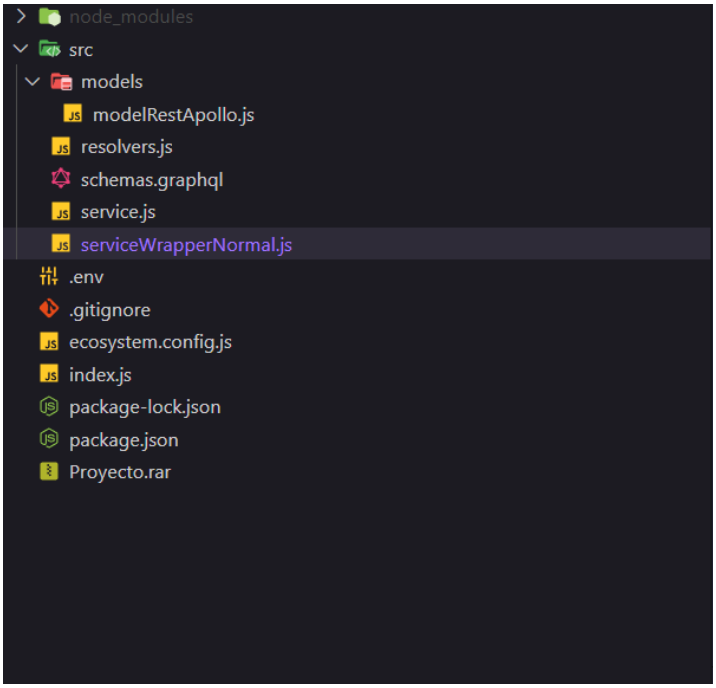
Tinoco, O., Pedro, I., Rosales López, P., & Salas Bacalla, I. J. (2010). Criterios de selección de metodologías de desarrollo de software. *Ind. Data*, *13*(2).

Toro, A. (2022). *¿Qué es Scrum? Conoce el Framework que agiliza el Trabajo en Equipo*. https://www.escueladenegociosydireccion.com/revista/business/scrum-framework-agiliza-trabajo-equipo/

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

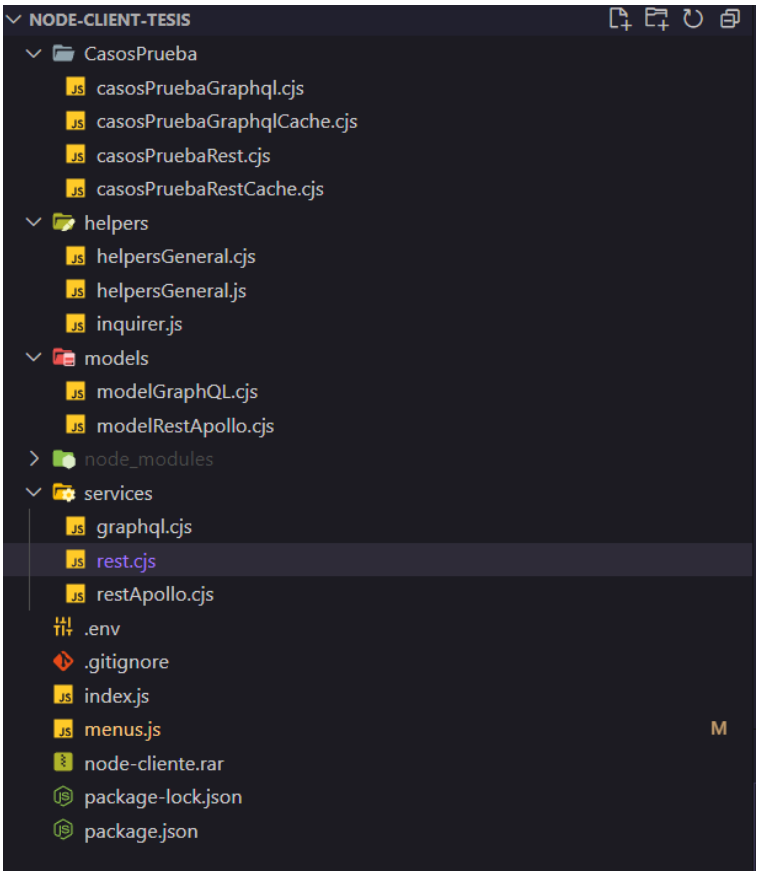
ANEXOS

ANEXO A: Estructura de carpetas del envoltorio GraphQL



ANEXO B: Paquetes externos utilizados en el envoltorio GraphQL

ANEXO C: Estructura de carpetas del cliente Node.js



ANEXO D: Paquetes externos utilizados en el cliente Node.js

```
  "main": "index.js",
  Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "type": "module",
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@apollo/client": "^3.7.7",
    "@lifeomic/axios-fetch": "^3.0.1",
    "apollo-link": "^1.2.14",
    "apollo-link-http": "^1.5.17",
    "apollo-link-rest": "^0.9.0",
    "axios": "^1.4.0",
    "colors": "^1.4.0",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "graphql-tag": "^2.12.6",
    "inquirer": "^9.1.4",
    "node-fetch": "^2.6.11",      You, 2 months ago • cambi
    "open": "^8.4.0"
  }
}
```