# UNIVERSIDAD TÉCNICA DEL NORTE


## Facultad de Ingeniería en Ciencias Aplicadas

## Carrera de Software

**Desarrollo de un envoltorio de la API-REST de live streaming Twitch utilizando GraphQL, para mejorar la eficiencia del consumo de datos, basado en la norma ISO/IEC 25023.**

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

Autor:

Alexander Paúl Véliz Sarzosa

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:** | 100481013-9 | | |
| **APELLIDOS Y NOMBRES:** | ALEXANDER PAUL VELIZ SARZOSA | | |
| **DIRECCIÓN:** | COTACACHI, BARRIO CALIENTE | | |
| **EMAIL:** | apvelizs@utn.edu.ec | | |
| **TELÉFONO FIJO:** | 062915793 | **TELÉFONO MÓVIL:** | 0982725478 |

| DATOS DE LA OBRA | |
|---|---|
| **TÍTULO:** | DESARROLLO DE UN ENVOLTORIO DE LA API-REST DE LIVE STREAMING TWITCH UTILIZANDO GRAPHQL, PARA MEJORAR LA EFICIENCIA DEL CONSUMO DE DATOS, BASADO EN LA NORMA ISO/IEC 25023. |
| **AUTOR(ES):** | ALEXANDER PAUL VELIZ SARZOSA |
| **FECHA:** | 07/09/2023 |
| **PROGRAMA:** | ☑ PREGRADO ☐ POSGRADO |
| **TÍTULO POR EL QUE OPTA:** | INGENIERO EN SOFTWARE |
| **DIRECTOR:** | PhD. ANTONIO QUIÑA |
| **ASESOR 1:** | MSc. JORGE CARAGUAY |

## 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

Alexander Paúl Véliz Sarzosa

C.I: 1004810139

# Certificación de director


**UNIVERSIDAD TÉCNICA DEL NORTE**
**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**

## CERTIFICACIÓN DIRECTOR

Por medio del presente yo PhD. Antonio Quiña, certifico que el Sr. **ALEXANDER PAÚL VÉLIZ SARZOSA** portador de la cédula de ciudadanía Nro. 1004810139, ha trabajado en el desarrollo del proyecto de grado: **"Desarrollo de un envoltorio de la API-REST de live streaming Twitch utilizando GraphQL, para mejorar la eficiencia del consumo de datos, basado en la norma ISO/IEC 25023."**, 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. A mis queridos padres, quienes han sido mi mayor fuente de inspiración y mi apoyo incondicional a lo largo de toda mi vida. Su constante aliento y motivación me han impulsado a superar cualquier obstáculo y a cumplir mis metas con determinación. Gracias por enseñarme la importancia del esfuerzo, disciplina, respeto y la humildad, valores por los que he llegado hasta donde estoy ahora.

A mis hermanos, quienes han sido mis compañeros de vida y mis cómplices en cada momento. Gracias por estar siempre a mi lado, apoyándome en cada paso que he dado.

También a mis amigos, aquellos con los que he compartido risas y momentos inolvidables a lo largo de esta travesía. Su amistad sincera y leal ha sido un pilar fundamental. Gracias por toda su ayuda.

Alexander Paúl Véliz Sarzosa

**Agradecimiento**

Quiero expresar mi más profundo agradecimiento a todas las personas que han sido parte fundamental en este importante logro.

A mis padres, quienes han sido un pilar en toda mi vida. Gracias por todo su amor, confianza y sacrificio que fueron la base de mi educación y formación tanto académica como personal. Sus enseñanzas son primordiales para mi crecimiento. Gracias por creer en mí, por alentarme a perseguir mis sueños y por su esfuerzo que me ayudó a alcanzar mis metas propuestas.

Especialmente a mi tutor de tesis, al PhD. Antonio Quiña y al MSc. Jorge Caraguay, por su guía, conocimientos y valiosos consejos que ayudaron con el desarrollo de este trabajo. Su experiencia y compromiso fueron de suma relevancia para orientarme en el camino correcto. Estaré eternamente agradecido con ellos y con los ingenieros de mi carrera por el respeto, las enseñanzas y amistad que me brindaron.

Así mismo agradezco a la Universidad Técnica del Norte, por brindarme la oportunidad de formarme académicamente y proporcionarme con los recursos necesarios para llevar a cabo mi etapa de estudio.

Finalmente, quiero agradecer a mis amigos, con quienes compartí cada etapa en este camino. Su apoyo incondicional, palabras de aliento y momentos de distracción y risas fueron importantes para mantener la motivación en esta travesía.

Alexander Paúl Véliz Sarzosa

## Tabla de Contenido

Certificación de director ..... iii
Dedicatoria ..... iv
Agradecimiento ..... v
Tabla de Contenido ..... vi
índice de Figuras ..... ix
Índice de Tablas ..... xi
Resumen ..... xiii
Abstract ..... xiv
Introducción ..... 1
Tema ..... 1
Problema ..... 1
Antecedentes ..... 1
Situación Actual ..... 1
Prospectiva ..... 2
Planteamiento del problema ..... 2
Objetivos ..... 3
Objetivo General ..... 3
Objetivos Específicos ..... 3
Alcance ..... 3
Metodología ..... 5
Justificación ..... 7
CAPÍTULO 1 ..... 8
Marco Teórico ..... 8
1.1. Fundamentos de arquitecturas orientas a microservicios. ..... 8
1.1.1. Arquitecturas de software ..... 8
1.1.2. Interfaz de programación de aplicaciones (API) ..... 9
1.1.3. Estilo arquitectónico REST ..... 9

1.1.4. Lenguaje de consultas GraphQL ....10
1.1.5. Envoltorios tecnológicos (Wrappers) ....17
1.2. Herramientas tecnológicas. ....18
1.2.1. BackEnd ....18
1.2.2. FrontEnd ....21
1.3. Metodologías de desarrollo de software. ....22
1.3.1. Scrum ....22
1.4. Experimentación en la ingeniería de software (IS) ....25
1.5. Estándar ISO/IEC 25000 ....26
1.5.1. Estándar ISO/IEC 25023 (Medición de calidad) ....27
CAPÍTULO 2 ....29
Desarrollo ....29
2.1. Análisis ....29
2.1.1. Roles de Proyecto ....29
2.1.2. Definición del Product Backlog ....30
2.1.3. Requisitos del proyecto ....31
2.2. Diseño ....37
2.2.1. Arquitectura tecnológica ....38
2.3. Desarrollo ....39
2.3.1. Sprint 1 ....39
2.3.2. Sprint 2 ....45
2.3.3. Sprint 3 ....50
2.3.4. Sprint 4 ....54
2.3.5. Sprint 5 ....57
2.4. Pruebas de aceptación ....59
CAPÍTULO 3 ....62
Validación de resultados ....62
3.1. Entorno del experimento ....62

3.1.1. Objetivo del experimento .....62
3.1.2. Factores y tratamiento .....62
3.1.3. Variable .....62
3.1.4. Hipótesis .....63
3.1.5. Diseño .....64
3.1.6. Tareas experimentales .....64
3.1.7. Instrumentación .....65
3.1.8. Recolección de datos .....66
3.1.9. Análisis .....66
3.2. Ejecución del experimento .....67
3.2.1. Muestra .....67
3.2.2. Preparación .....67
3.2.3. Recolección de datos .....68
3.3. Resultados .....69
3.3.1. Análisis de resultados .....69
3.3.2. Análisis de eficiencia .....70
3.3.3. Análisis de impactos .....77
Conclusiones .....78
Recomendaciones .....79
Bibliografía .....80
Anexos .....86

**índice de Figuras**

**Figura 1** *Árbol de problemas.* ..... 2

**Figura 2** *Servicios de la API de Twitch.* ..... 3

**Figura 3** *Arquitectura de proyecto.* ..... 4

**Figura 4** *Proceso experimental de Wohlin.* ..... 4

**Figura 5** *Diagrama de flujo sobre el proceso de evaluación del experimento.* ..... 6

**Figura 6** *Campos de una consulta GraphQL* .....11

**Figura 7** *Argumentos en una consulta GraphQL.* .....12

**Figura 8** *Nombre de la operación.* .....12

**Figura 9** *Variables.* .....13

**Figura 10** *Tipo de dato en esquema.* .....14

**Figura 11** *Argumentos.* .....14

**Figura 12** *Type Query y Type Mutation.* .....15

**Figura 13** *Type enum.* .....15

**Figura 14** *Tipo de entrada (input type).* .....16

**Figura 15** *Divisiones de la norma ISO/IEC 25000.* .....27

**Figura 16** *Estructura de fase de desarrollo del proyecto.* .....29

**Figura 17** *Arquitectura tecnológica.* .....38

**Figura 18** *Ejemplo de estructura para obtener un token.* .....41

**Figura 19** *Ejemplo de estructura para obtener transmisiones en vivo.* .....42

**Figura 20** *Ejemplo de estructura para obtener los videos que tiene un juego.* .....43

**Figura 22** *Ejemplo de estructura para obtener la información del canal de un usuario.* ...48

**Figura 23** *Ejemplo de estructura para obtener la información de un juego.* .....49

**Figura 24** *Ejemplo del menú de selección de consumo*. .....52

**Figura 25** *Ejemplo del menú de selección de funcionalidad a consumir.* .....52

**Figura 26** *Ejemplo del menú de selección de distribución.* .....53

**Figura 27** *Ejemplo del modelo para consulta de la funcionalidad Transmisiones en Vivo.* .....53

**Figura 28** *Ejemplo del modelo para consulta de la funcionalidad Videos por Juego.* .....55

**Figura 29** *Ejemplo del modelo para consulta de la funcionalidad Clips por Usuario.* .....58

**Figura 30** *Arquitectura de laboratorio experimental.* .....65

**Figura 31** *Proceso de ejecución del experimento.* .....67

**Figura 32** *Promedios del tiempo de arquitecturas sin caché del Caso de uso 01.* .....70

**Figura 33** *Promedios del tiempo de arquitecturas con caché del Caso de uso 01.* .....71

**Figura 34** *Promedios del tiempo de arquitecturas sin caché del Caso de uso 02.* .....72

**Figura 35** P*romedios del tiempo de arquitecturas con caché del Caso de uso 02.* .....72

**Figura 36** *Promedios del tiempo de arquitecturas sin caché del Caso de uso 03.* .....73

**Figura 37** *Promedios del tiempo de arquitecturas con caché del Caso de uso 03.* .....74

**Figura 38** *Valor promedio de eficiencia entre GraphQL y REST sin caché.* .....74

**Figura 39** *Valor promedio de eficiencia entre GraphQL y REST con caché.* .....75

**Figura 40** *Resultado estadísticos descriptivos entre las arquitecturas.* .....75

## Índice de Tablas

**Tabla 1** Recursos principales de la API de Twitch. ....21

**Tabla 2** Roles de Scrum. ....23

**Tabla 3** Artefactos de Scrum. ....23

**Tabla 4** Fases de Scrum. ....24

**Tabla 5** Comparativa de metodologías de gestión de proyectos. ....25

**Tabla 6** Subcaracterísticas de la usabilidad de la ISO/IEC 25023. ....27

**Tabla 7** Roles del proyecto. ....29

**Tabla 8** Técnica T-Shirt Size. ....30

**Tabla 9** Product Backlog. ....30

**Tabla 10** Historia de Usuario 1. ....31

**Tabla 11** Historia de Usuario 2. ....32

**Tabla 12** Historia de Usuario 3. ....32

**Tabla 13** Historia de Usuario 4. ....33

**Tabla 14** Historia de Usuario 5. ....33

**Tabla 15** Historia de Usuario 6. ....34

**Tabla 16** Historia de Usuario 7. ....35

**Tabla 17** Historia de Usuario 08. ....35

**Tabla 18** Historia de Usuario 09. ....36

**Tabla 19** Historia de Usuario 10. ....36

**Tabla 20** Historia de Usuario 11. ....37

**Tabla 21** Detalle Sprint 0. ....37

**Tabla 22** Planificación Sprint 0. ....37

**Tabla 23** Detalle general de los Sprints. ....39

**Tabla 24** Sprint 1 Backlog. ....40

**Tabla 25** Retrospectiva Sprint 1. ....44

**Tabla 26** Sprint 2 Backlog. ....45

**Tabla 27** Retrospectiva Sprint 2. ....49

**Tabla 28** Sprint 3 Backlog. .....50

**Tabla 29** Retrospectiva Sprint 3. .....54

**Tabla 30** Sprint 4 Backlog. .....54

**Tabla 31** Retrospectiva Sprint 4. .....56

**Tabla 32** Sprint 5 Backlog. .....57

**Tabla 33** Retrospectiva Sprint 5. .....59

**Tabla 34** Pruebas de aceptación. .....59

**Tabla 35** Variables para experimento. .....62

**Tabla 36** Diseño del experimento. .....64

**Tabla 37** Estructura de recolección de datos. .....66

**Tabla 38** Datos Caso de uso 01. .....68

**Tabla 39** Datos Caso de uso 02. .....69

**Tabla 40** Datos Caso de uso 03. .....69

**Tabla 41** Porcentaje de eficiencia del Caso de uso 01. .....70

**Tabla 42** Porcentaje de eficiencia del Caso de uso 02. .....72

**Tabla 43** Porcentaje de eficiencia del Caso de uso 03. .....73

**Tabla 44** Eficiencia entre las arquitecturas. .....77

**Resumen**

Twitch es una plataforma de streaming en vivo popular para videojuegos y otros contenidos. El consumo de servicios de su API se ha convertido en una actividad popular y frecuente para millones de usuarios. El problema radica en la falta de eficiencia en el consumo de datos de las API-REST. Al usar GraphQL como tecnología de envoltura permite optimizar el consumo de datos y permitir que los usuarios realicen consultas personalizadas. El presente trabajo de titulación se centra en el desarrollo de un envoltorio de la API-REST de Twitch utilizando GraphQL, con el objetivo de mejorar la eficiencia del consumo de datos. Para alcanzar este objetivo, se generó un marco teórico a través de una revisión de literatura que proporciona los fundamentos necesarios para el desarrollo del proyecto. Para realizar el desarrollo del proyecto se usa la metodología ágil Scrum y el experimento se basa en la guía Wohlin, que se enfoca en experimentos tecnológicos. Finalmente para la evaluación de la eficiencia es bajo la norma ISO/IEC 25023. Se realizó una evaluación de eficiencia y rendimiento, mediante un laboratorio experimental, tomando en cuenta el tiempo de respuesta de una solicitud. Los resultados han demostrado que GraphQL es más eficiente en la mayoría de los casos. A pesar de ello, hubo un caso donde REST es ligeramente más eficiente en escenarios específicos utilizando caché. En base a los resultados que se registraron, se concluye que GraphQL es una opción viable para mejorar la eficiencia de servicios hechos con REST. Se debe destacar el uso del caché en algunos casos ya que puede ofrecer beneficios significativos. En conclusión, en este trabajo de titulación se ha logrado desarrollar un envoltorio para la API-REST de Twitch, demostrando ventajas y beneficios en términos de eficiencia de rendimiento. También, se recomienda usar el lenguaje de consultas GraphQL en entornos donde se requiera un alto consumo de datos, especialmente donde se requiera realizar consultas personalizadas.

**Abstract**

Twitch is a popular live streaming platform for video games and other content. Consuming services from its API has become a popular and frequent activity for millions of users. The problem lies in the lack of efficiency in API-REST data consumption. Using GraphQL as a wrapper technology allows to optimize data consumption and enable users to perform custom queries. The present degree work focuses on the development of a Twitch live streaming API-REST wrapper using GraphQL, with the goal of improving the efficiency of data consumption. To achieve this objective, a theoretical framework was generated through a literature review that provides the necessary foundations for the development of the project. To carry out the development of the project, the agile Scrum methodology is used and the experiment is based on the Wohlin guide, which focuses on technological experiments. Finally, the efficiency evaluation is based on the ISO/IEC 25023 standard. An evaluation of efficiency and performance was carried out through an experimental laboratory, taking into account the response time of a request. The results have shown that GraphQL is more efficient in most cases. Despite this, there was one case where REST is slightly more efficient in specific scenarios using cache. Based on the results that were recorded, it is concluded that GraphQL is a viable option to improve the efficiency of services made with REST. The use of caching should be highlighted in some cases as it can offer significant benefits. In conclusion, in this degree work it has been possible to develop a wrapper for the Twitch API-REST, demonstrating advantages and benefits in terms of performance efficiency. Also, it is recommended to use GraphQL query language in environments where high data consumption is required, especially where custom queries are required.

# Introducción

## Tema

Desarrollo de un envoltorio de la API-REST de live streaming Twitch utilizando GraphQL, para mejorar la eficiencia del consumo de datos, basado en la norma ISO/IEC 25023.

## Problema

## Antecedentes

Twitch es una plataforma de live streaming (transmisión en vivo), su propósito es realizar transmisiones en vivo sobre videojuegos, como también contenido de entretenimiento. La plataforma dispone de un área de desarrollo en la cual se cuenta con una API RESTful que permite a los desarrolladores crear integraciones creativas para la comunidad de Twitch en general (Twitch Developers, 2022).

Para solicitar algún dato a la API, se deben incluir los encabezados necesarios para obtener una respuesta correcta en el menor tiempo posible. En caso de no contar con los encabezados correctos, se genera un error de autentificación y se tendrá que obtener una nueva autentificación lo cual limita el proceso de obtención y uso de las funcionalidades (Amat, 2021).

Hoy en día, existen muchas empresas que desarrollan sus servicios a través de las API-REST. Sin embargo, se han descubierto limitaciones en el uso de REST y, dado que los productos se han vuelto volátiles y necesitan evolucionar rápidamente, se ha vuelto cada vez más complejo más allá del simple CRUD, y están impulsando cambios en la forma en que interactúan y consumen (Delgadillo Hinojosa, 2019).

## Situación Actual

Los inconvenientes que presenta la API REST de Twitch como su complejidad de uso y escasa flexibilidad, resultan ser prejudiciales ya que al usarlo, representa una molestia o disgusto por su complejidad de adaptación con el producto a realizar (aplicaciones de escritorio, páginas web o aplicaciones móviles). Por lo cual se plantea nuevas tecnologías orientados a servicios como lo es GraphQL que permitirá fortalecer el entorno para poder consumir los servicios disponibles en la plataforma.

Además, no gestiona servicios personalizados. Por lo cual se creó una versión donde el acceso a las funcionalidades es más sencillo. A pesar de eso, aún existe el inconveniente de personalizar una solicitud de los datos.

## Prospectiva

Por medio de la implementación de GraphQL como un envoltorio para la API-REST de Twitch, es posible mejorar la eficiencia, el rendimiento, la escalabilidad y la flexibilidad de la API, así como la experiencia de los desarrolladores que la usan.

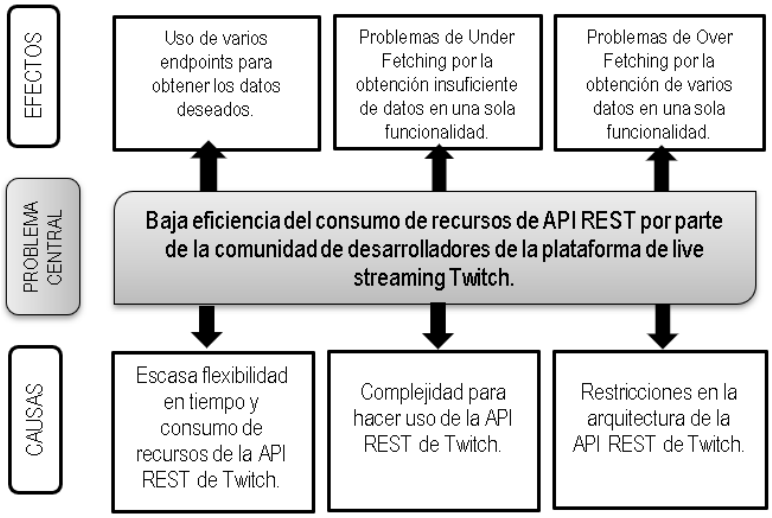
GraphQL incluye una búsqueda de endpoint que permite a los clientes recuperar metadatos sobre preprints con un título determinado. Por el contrario, API REST devuelve todos los datos que se encuentren al realizar una consulta a dicho endpoint (Brito et al., 2019). La expresividad y flexibilidad de GraphQL lo han convertido en un candidato atractivo para proveedores de API en muchas industrias, especialmente a través de la web (Mavroudeas et al., 2021).

## Planteamiento del problema

Hoy en día las empresas y organizaciones implementan las API-REST en sus proyectos para sus negocios. A medida que se requiere realizar de manera ágil y eficiente el desarrollo de productos, se ha vuelto cada vez más complejo que va más allá de las operaciones básicas CRUD (crear, leer, actualizar y eliminar). Por lo cual se plantea nuevas tecnologías orientadas a servicios como lo es GraphQL, que permite fortalecer el entorno, y así, consumir los servicios de manera personalizada y flexible.

**Figura 1**

*Árbol de problemas.*



## Objetivos

### Objetivo General

Desarrollar un envoltorio del API-REST de live streaming Twitch utilizando GraphQL, para mejorar la eficiencia del consumo de datos, basado en la norma ISO/IEC 25023.

### Objetivos Específicos

- Establecer un marco teórico para el desarrollo del proyecto.
- Desarrollar un envoltorio GraphQL de la API REST de la plataforma de Twitch que permita el consumo de funcionalidades.
- Evaluar el envoltorio a través de un experimento para comparar la eficiencia entre la API GraphQL y API REST bajo la norma la ISO/IEC 25023.
- Analizar los resultados obtenidos del proyecto.

### Alcance

En este proyecto de trabajo de titulación se desarrolla un envoltorio a la API-REST Twitch usando el Lenguaje de Consultas GraphQL, con el objetivo de que el consumo de sus funcionalidades sea más eficiente, mediante el uso de herramientas de programación. También, se realizará un experimento donde se compare el tiempo de respuesta, comparando la API-REST y el envoltorio, se usará la metodología de Wohlin para llevar a cabo este caso.

Para empezar con este proyecto, se tomará en cuenta 5 funcionalidades de la API de Twitch, las cuales serán niveles de consumo. Se realizará una distribución de datos para realizar el experimento. Los servicios para usar se muestran en la Figura 2.

**Figura 2**

*Servicios de la API de Twitch.*


*Nota.* (Twitch Developers, 2022).

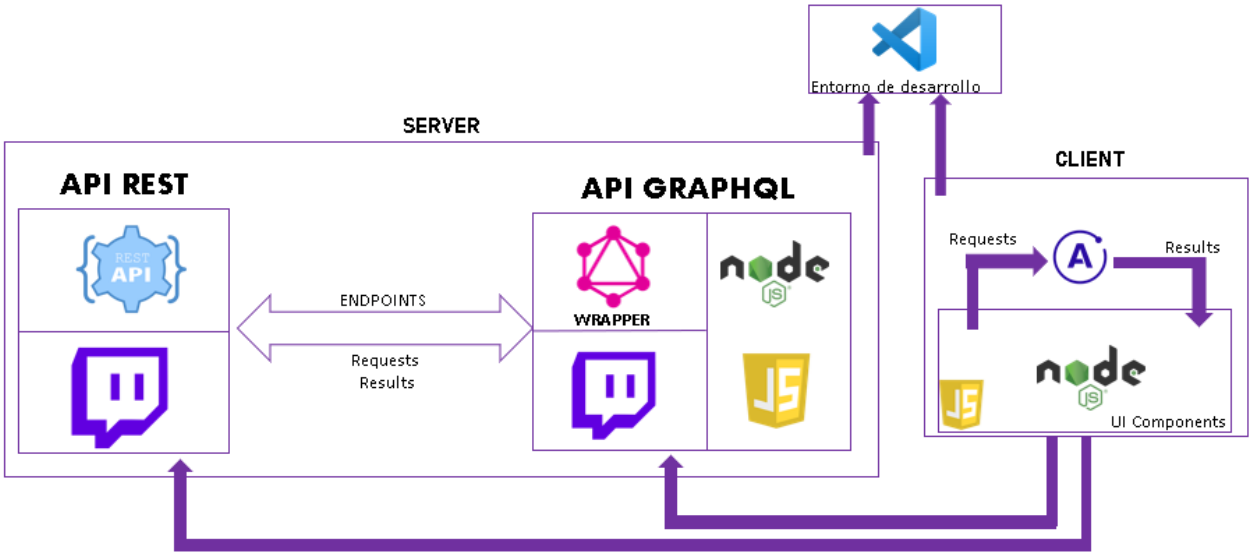
Después de obtener los endpoints o servicios que se van a usar, se inicia el proceso de desarrollo del envoltorio, con el fin de realizar la comparación. Para esto, se utilizará Node.js

como entorno de tiempo de ejecución. Usa el lenguaje de programación JavaScript, el cual es código abierto que se usa frecuentemente para crear aplicaciones de red. Viene con un administrador de paquetes NPM para la administración moderna de dependencias de aplicaciones lo que ayuda a los desarrolladores a crear proyectos Node.js (Koishybayev & Kapravelos, 2020). Esta herramienta se usará para el envoltorio y para el desarrollo del cliente donde los resultados se lo muestran por consola.

Finalmente, para hacer pruebas del envoltorio y mostrar los datos o resultados se empleará Apollo Client. Esta es una biblioteca de JavaScript que ayuda a la conexión de un servidor GraphQL desde una aplicación cliente. La arquitectura correspondiente al proyecto se lo muestra en la Figura 3.

**Figura 3**

*Arquitectura de proyecto.*




Con respecto al experimento, se realiza un diagrama que se observa en la Figura 4, donde se muestra el proceso del experimento haciendo uso de la metodología de Wohlin.

**Figura 4**

*Proceso experimental de Wohlin.*


**Metodología**

El proceso para realizar este trabajo estará compuesto por etapas, las cuales ayudarán y permitirán cumplir con los objetivos establecidos.

En una de estas etapas, para cumplir el primer objetivo se establece el desarrollo de un marco teórico por medio de una investigación de literatura. El proceso de realizar una revisión de literatura no solo ayuda a comprender el marco teórico de los estudios e investigaciones realizadas, si no también sus casos analíticos centrales, conceptos básicos y principales resultados (Morales Inga & Morales Tristán, 2020). Permite comprender los principales aspectos de cómo crear envoltorios a partir de múltiples datos, aparentemente desde una API REST; además, los beneficios de usar el lenguaje de consulta GraphQL y las métricas relevantes en este proyecto.

Para llevar a cabo el segundo objetivo, se toma en cuenta la metodología Scrum, el cual es un marco de trabajo que permite a equipos de trabajo, personas y organizaciones a generar valor con soluciones adaptativas que se usan en problemas complejos. Se basa en el empirismo y el pensamiento *Lean*. El empirismo menciona que el conocimiento se gana con la experiencia y en base a lo observado se hace la toma de decisiones. El pensamiento Lean se enfoca en lo que es esencial (Schwaber & Sutherland, 2020).

También, se escoge la API-REST de la plataforma de live streaming Twitch porque es un servicio público que brinda sus funcionalidades principalmente a su comunidad de desarrolladores y personas externas interesadas (Twitch Developers, 2022).

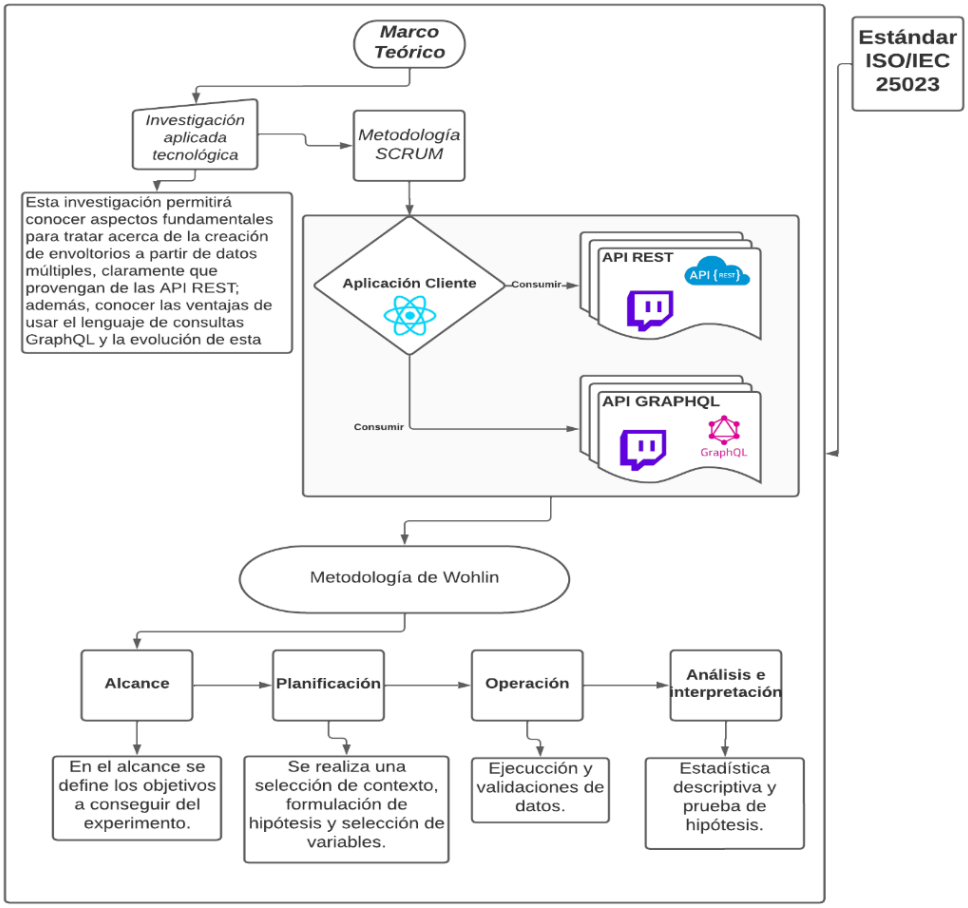
Para el tercer objetivo, se usará la metodología de Wohlin para realizar el experimento porque el objetivo es evaluar diferentes niveles de formalidad al recolectar datos de esfuerzo.

Consiste en 4 fases, la primera es el alcance donde se definen los objetivos para definir los Procesos de Software Personales (PSP). La segunda fase es la de planificación en la cual se formulan las hipótesis, se realiza una selección de variables y sujetos, diseño de experimentos, instrumentación y finalmente la evaluación de validez. La tercera fase es la de operación, en este caso se realiza una preparación, ejecución y validación de los datos. Y la fase cuatro es análisis e interpretación, con el objetivo realizar la estadística descriptiva, reducción de datos y las pruebas de hipótesis para obtener datos claros y precisos (Mera, 2022; Research et al., 2019).

Junto con la guía de Wohlin también se considerará la norma ISO/IEC 25023. Esta establece un conjunto de características de calidad del software relacionados con el rendimiento, la eficiencia y la capacidad de respuesta del sistema. Tiene un enfoque estructurado y completo, así como a su enfoque en aspectos clave de rendimiento y calidad de datos (Aziz et al., 2018). En la Figura 5 se muestra un flujo sobre este proceso.

**Figura 5**

*Diagrama de flujo sobre el proceso de evaluación del experimento.*



## Justificación

El presente proyecto tiene como objetivo realizar un trabajo de investigación, donde se compare la efectividad y la eficiencia que tiene utilizar el lenguaje de consultas en los proyectos de software. Además se enfoca hacia los Objetivos de Desarrollo Sostenible (ODS), en donde se contemplan los siguientes:

- **Educación de calidad (Objetivo 4):** Se pretende incentivar y motivar a seguir por un camino para la formación que tenga que ver con la tecnología y proveer información para fortalecer el conocimiento (Naciones Unidas, 2018).
- **Industria, innovación e infraestructura (Objetivo 9):** Las tecnologías de la información y comunicación han estado al frente en respuesta de las necesidades, proyectos o innovaciones de las personas. Este proyecto puede brindar conocimiento para desarrollar infraestructuras fiables, sostenibles, resilientes y de calidad, para apoyar el desarrollo económico y el bienestar humano (Organización Internacional del Trabajo, 2017).

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 (SNP, 2021).

**Justificación Institucional. -** El presente trabajo busca apoyar a la Universidad Técnica del Norte en la visión de constituirse como una institución superior de investigación que sea líder en la generación de ciencia y tecnología e innovación con indicadores de calidad nacional e internacional (Universidad Técnica del Norte, 2020).

**Justificación Tecnológica. -** Desarrollar un envoltorio GraphQL de la API REST de la plataforma de Twitch para mejorar y agilizar algunos de los servicios para el consumo de la API y su rendimiento, además, un entorno de cliente donde se realice un laboratorio experimental para comparar las herramientas.

**Justificación Académica. -** Brindar una oportunidad de realizar nuevos proyectos y generar un nuevo campo de investigación sobre la eficiencia de nuevas herramientas en la Universidad Técnica del Norte.

# CAPÍTULO 1

# Marco Teórico

## 1.1. Fundamentos de arquitecturas orientas a microservicios.

### 1.1.1. Arquitecturas de software

La arquitectura de software se ha consolidado como una disciplina que intenta contrarrestar los efectos negativos que pueden surgir durante el desarrollo de un producto informático, ocupando un rol significativo en la estrategia de negocio de una organización que basa sus operaciones en el software, volviéndose necesaria para todo tipo de desarrollo, incluyendo los videojuegos (A. Hernández et al., 2018).

Los métodos de descubrimiento de arquitectura en tiempo de ejecución utilizan un enfoque centralizado, en el que el proceso se lleva a cabo desde una única ubicación. Estos métodos son inadecuados para grandes sistemas de software distribuidos porque no se escalan bien y tienen un único punto de falla (Porter et al., 2021).

Es esencial realizar una arquitectura antes de realizar este proyecto para proporcionar una organización y estructura del sistema, lo que llega a facilitar la compresión del proyecto. Establece una escalabilidad que permite un aumento eficiente y genera una base sólida para construir aplicaciones que satisfagan los requisitos y al cliente.

#### 1.1.1.1. Arquitectura Monolítica

Las empresas para su continuo funcionamiento dependen de aplicaciones, las cuales se acumulan con el paso del tiempo. Entre los problemas que actualmente se evidencian en las empresas, es que la mayoría de estas aplicaciones legadas son monolíticas y tienen obsolescencia, lo que aumenta la complejidad para su mantenimiento y evolución (Torres, 2018).

El desarrollo de software se lo realiza con arquitectura monolítica, en donde la estructura del software está conformada de tal manera que todos los aspectos funcionales quedan acoplados y sujetos en un mismo programa. Por esta razón se presenta la arquitectura de microservicios.

#### 1.1.1.2. Arquitectura Microservicios

Microservicio es un estilo arquitectónico que separa grandes sistemas en pequeñas unidades funcionales para proporcionar un mejor modularidad. Un desafío clave del diseño de arquitectura de microservicios que se analiza con frecuencia en la literatura es la identificación y descomposición de los módulos de servicio (Karabey Aksakalli et al., 2021).

La arquitectura de microservicios es una forma arquitectónica parcialmente nueva que poco a poco se está volviendo más habitual. Cada uno de los microservicios es un código que puede estar implementado con diferente lenguaje de programación. Se comunican entre sí por medio de APIs y contiene sus propios sistemas de almacenamiento. Con esto, se previene la sobrecarga y fallas en la aplicación.

### 1.1.2. Interfaz de programación de aplicaciones (API)

Las Interfaces de Programación de Aplicaciones (APIs) son un conjunto de protocolos y definiciones que se usa para el desarrollo e integración del software de las aplicaciones, permitiendo así, la comunicación y enlace entre dos aplicaciones de software haciendo uso de un conjunto de reglas.

Además, exponen servicios o datos de una aplicación de software a través de un conjunto de recursos predefinidos, como métodos, objetos o URI (identificador de recursos uniforme). Mediante el uso de estos recursos, otras aplicaciones pueden acceder a los datos o servicios sin tener que implementar los objetos y procedimientos subyacentes (Meng et al., 2018).

El funcionamiento consiste en la comunicación entre un servidor y clientes web, se lo hace a través de intercambio de mensajes HTTP en un entorno seguro de interacción. Estos mensajes son de dos tipos:

- Peticiones. - las cuales son enviadas por el cliente hacia el servidor, con el objetivo de requerir el inicio de una operación para conectarse con recursos de información.
- Respuestas. – consiste en la materialización de la acción que fue solicitada por el cliente.

### 1.1.3. Estilo arquitectónico REST

Transferencia de estado representacional (REST), es una arquitectura que provee un estándar de comunicaciones entre dispositivos presentes en la red, haciendo más fácil la comunicación entre ellos. REST basa su funcionamiento en el protocolo HTTP, el cual utiliza el modelo pregunta-respuesta ejecutado por un cliente y un servidor, los cuales se

caracterizan principalmente por ser independientes el uno del otro, es decir, que la implementación del cliente y del servidor se puede hacer de forma aislada (Benavidez & García, 2019).

REST incorpora mejoras sustanciales lo que permite descomponer las arquitecturas tradicionales en partes más pequeñas, los servicios que implementa una arquitectura de microservicios usando un marco de mensajería común, es API RESTful (L. M. A. Hernández et al., 2021).

Una preferencia en el desarrollo de software es la Arquitectura Orientada a Servicios (SOA) porque ofrece eficiencia, fácil crecimiento y agilidad. El diseño arquitectónico que se usa con frecuencia para realizar los proyectos es REST; sin embargo, se han encontrado algunos problemas de gestión de datos. Para cambiar este hecho, aparecen varias opciones como SPARQL, Cypher, Gremlin y las más conocida GraphQL (Quiña-Mera, Fernández-Montes, et al., 2022).

### 1.1.4. Lenguaje de consultas GraphQL

En la última década, REST se ha convertido en el enfoque más común para proporcionar servicios web, aunque originalmente no se diseñó para manejar aplicaciones modernas típicas (por ejemplo, aplicaciones móviles (Chaves-Fraga et al., 2020a). Actualmente, GraphQL se está convirtiendo en el lenguaje de consultas de referencia para desarrollar APIs web, el cual plantea mejorar problemas de las APIs RESTful con el acceso a datos (Quiña-Mera, García, et al., 2022).

Es una tecnología desarrollada por la compañía Facebook publicada para su uso gratuito en el año 2015. Esta iniciativa se desarrolla debido a la necesidad identificada por la compañía de rediseñar el modelo de búsqueda de datos a una API que no tomara gran esfuerzo de parte del servidor para preparar la data y de parte del cliente para transformarla y adecuarla a un uso (Alfredo Guillen-Drija & Quintero, 2018).

Los servidores de GraphQL están disponibles para múltiples lenguajes, como JavaScript, Perl, Python, Ruby, Java, entre otros. Incluso es una herramienta que no depende de la fuente de datos, esto quiere decir que los datos pueden obtenerse desde un fichero JSON, una base de datos o incluso un archivo de texto plano (Grado et al., 2021).

### 1.1.4.1. Estructura de consultas y mutaciones

- **Operaciones**

  Tanto GraphQL como REST se utilizan para recuperar datos a través de una API. Además, ambos protocolos utilizan el protocolo HTTP. Sin embargo, las API REST usan métodos HTTP (GET, POST, PUT, DELETE) para implementar operaciones CRUD (Crear, Leer, Actualizar, Eliminar), mientras que GraphQL usa métodos específicos de API. GraphQL admite solicitudes GET y POST para realizar consultas (Hietala et al., 2020). Las operaciones que se dan en GraphQL son: las Consultas (querys) que se usa con la finalidad de obtener los datos; las Mutaciones (mutations) las cuales ayudan a realizar las operaciones CRUD.

- **Campos**

  GraphQL se trata de solicitar campos específicos en los objetos. Estos campos pueden representar tipos de datos escalares (ID, Float, String, Int y Boolean) u objetos especiales. Cada campo se ejecuta de acuerdo con el Resolver correspondiente (Angulo Vásquez, 2020).

  Por ejemplo, en la Figura 6 se muestra el objeto *hero* y sus campos escalares de tipo String *name*, donde la respuesta será un JSON y tendrá la misma estructura de la solicitud.

  **Figura 6**

  *Campos de una consulta GraphQL*

- **Argumentos**

  En un sistema como REST, solo puede pasar un único conjunto de argumentos: los parámetros de consulta y los segmentos de URL en su solicitud (Foundation, 2022). Mientras que en GraphQL, cada campo y objeto anidado puede tener su propio conjunto de argumentos y esto ayuda a realizar diversas búsquedas de API (Ghebremicael, 2017).

Por ejemplo, en la Figura 7 se observa el argumento *id* que filtra un humano en específico. Además, cada dato escalar puede tener su propio argumento, los cuales pueden ser de diferentes tipos.

**Figura 7**

*Argumentos en una consulta GraphQL.*

```
query{
  human(id: "1") {
    name
    height
  }
}
```

- **Nombre de la operación**

  El nombre de operaciones es un nombre que se usa para definir la operación. Estos nombres pueden ser un accesorio de depuración muy eficiente a la hora de realizar una petición para reconocer diferentes solicitudes de GraphQL o en otro caso, sus errores.

  Por ejemplo, en la Figura 8 se muestra el nombre de la operación como *NombredePersonajeyAmigos* para identificar la consulta que devuelve el nombre de un personaje y sus amigos.

  **Figura 8**

  *Nombre de la operación.*

```
query NombredePersonajeyAmigos {
  character(id:"1"){
    name
     friends{
      name
    }
  }
}
```

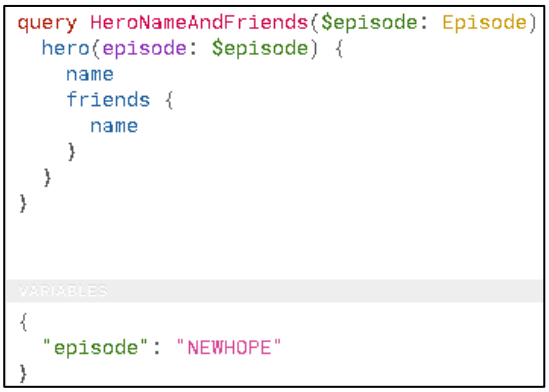
- **Variables**

  Son datos de entrada que se usan en una operación de consulta o mutación, los cuales sustituyen a los valores estáticos que están fijos en las operaciones. Así, estos valores se vuelven dinámicos.

Estas pueden ser agregados a los argumentos; se usan nombres que en GraphQL siempre van precedidos por un carácter $; serán de distintos tipos de datos, como String, Int, Date Time, etc. Por ejemplo, en la Figura 9 se observa la variable *episode* que se usa como argumento en la consulta, donde el valor es *NEWHOPE*; los datos de entrada se envían como un objeto JSON, asegurándose que el nombre de la variable sea correcto como la clave JSON (Banks & Porcello, 2018).

**Figura 9**

*Variables.*




### 1.1.4.2. Esquemas y tipos

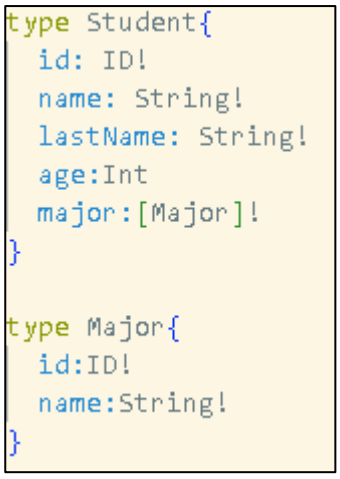
- **Tipos de datos y objetos**

  En GraphQL, un tipo significa un objeto personalizado, los cuales reflejan las caracteristicas más importantes de la aplicación. Además, son el componente principal de cualquier esquema GraphQL. Un tipo contiene campos los cuales son datos relacionados a cada objeto; cada campo es de un tipo escalar específico (Int, Float, String, Boolean, ID, Date). Estos esquemas se pueden escribir en archivos JavaScript en forma de cadena o un archivo que contenga la extensión *.graphql* (Banks & Porcello, 2018).

  Por ejemplo, en la Figura 10 se observan dos tipos, los cuales son *Student* y *Major* con sus diferentes campos y tipos escalares. Cuando un tipo escalar contiene el símbolo “!” significa que ese campo no admite valores *null*, es decir, no debe estar vacío. En el caso de “[Major]!”, es una matriz del tipo *Major*, el cual no debe estar vacío.

**Figura 10**

*Tipo de dato en esquema.*



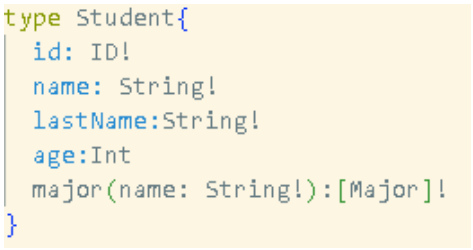

- **Argumentos**

  En un sistema REST existe la posibilidad de añadir un conjunto de argumentos, pero en GraphQL, se puede agregar un conjunto de argumentos a los campos, incluso, pasar argumentos en los tipos escalares es beneficioso ya que permite implementar transformaciones en los datos, directamente en el servidor (Román, 2018).

  Definir argumentos en el esquema permite que la información que se conseguirá sea filtrada a través del valor de estos argumentos. Cada campo en un tipo de objeto GraphQL puede tener cero o más argumentos (Foundation, 2022). Por ejemplo, en la Figura 11 se muestra el argumento *name* de tipo String y obligatorio por el símbolo *!*.

**Figura 11**

*Argumentos.*



- **Tipos de consulta y mutación (type Query, type Mutation)**
  Muchos de los tipos definidos en los esquemas serán solo tipo de objetos normales, pero existen dos tipos que son particulares dentro de un esquema; estos son el type Query y type Mutation (Foundation, 2022).

  Un *type Query* es una solicitud que se envía desde la aplicación cliente al servidor GraphQL para la obtención de datos. Es uno de los tipos de nivel raíz en GraphQL. Por otro lado, las mutaciones son operaciones enviadas al servidor para crear, actualizar y eliminar datos. Son similares a las acciones CREATE, PUT y DELETE para llamar a las APIs basadas en REST (Malinowski et al., 2019). A continuación, en la Figura 12 se muestra la sintaxis que pertenece a *type Query* y *type Mutation:*

  **Figura 12**

  *Type Query y Type Mutation.*

```
type Query {
  getStudents: [Student]
  getMajorById(id: Int!): Major
}

type Mutation{
  updateStudent(id:Int!): Student
  deleteMajor(id:Int!):String
}
```

- **Tipos de enumeración (enum)**
  Los *enum* son un tipo especial de GraphQL que se utiliza para representar un conjunto de nombres simbólicos asociados con valores y constantes únicos (Schuh, 2018). En la Figura 13 se observa cómo se vería una definición de *enum*. Significa que si el esquema usa el tipo *StatusStudentRegistration*, exactamente se obtendrá uno de los parámetros COMPLETE, PENDING o CANCELLED.

  **Figura 13**

  *Type enum.*

```
enum StatusStudentRegistration{
  COMPLETE
  PENDING
  CANCELLED
}
```

- **Tipos de entrada (input type)**

  En GraphQL se define un conjunto de campos de entrada; pueden ser valores escalares, enumeraciones o cadenas (Quiña-Mera et al., 2023). Pero también existen un tipo que posibilita pasar informacion o datos muy fácilmente. Esto es relevante al usar mutaciones, donde en algunos casos, es posible pasar un objeto completo para crearlo (Foundation, 2022). Se convierten en objetos de entrada de datos como se muestra en la Figura 14.

  Los tipos de entrada son la clave para poder organizar y escribir claramente un esquema GraphQL (Banks & Porcello, 2018).

**Figura 14**

*Tipo de entrada (input type).*

```
input StudentInput{
    id: ID!
    name: String!
    lastName:String!
    age: Int
}

type Mutation{
    createStudent(input: StudentInput): Student
}
```

### 1.1.4.3. Ejecución

En GraphQL, para mostrar los resultados, un servidor lleva a cabo una consulta GraphQL que devuelve un resultado donde se muestra la forma de la consulta solicitada, por lo común como JSON. Este proceso se lo hace por medio de resolutores (resolvers). Cada campo de consulta es representado como una función o método (Foundation, 2022).

**Campos raíz y resolutores**

Un resolutor (resolver) GraphQL describe la relación entre los tipos/campos GraphQL definidos con las fuentes de datos (Priyatna et al., 2019).

Son funciones de extracción de datos responsable de acceder a los conjuntos de datos subyacentes. Estas funciones las escriben programadores que utilizan un lenguaje de programación específico y luego las utilizan los motores GraphQL para traducir las consultas de GraphQL al lenguaje de consulta subyacente adecuado (Chaves-Fraga et al., 2020a).

Así mismo, existen funciones asíncronas las cuales son fundamentales cuando existe una carga desde una base de datos o una API, esto devuelve una *Promesa*; en el lenguaje JavaScript las promesas se usan para trabajar con valores asíncronos (Foundation, 2022).

Según Foundation, (2022), una función resolutora conlleva cuatro argumentos:

- **obj:** este es el objeto anterior; este parámetro es poco usado en el campo raíz.
- **args:** argumentos que se pasan al campo en la consulta.
- **context:** es un valor que provee a cada resolutor; este argumento abarcar la información de resolución de cada consulta como la autenticación del usuario.
- **Info:** un campo que contiene la informacion específica del campo que se usa en la consulta actual como los datos del nombre y la ruta jerárquica correspondiente al campo.

Al ser una nueva tecnología, sus propiedades están establecidas a la mejora de la eficiencia de un proyecto. Su flexibilidad, su naturaleza para consultas precisas y personalizadas, ofrecen un rendimiento optimizado y una mejor experiencia para el usuario.

### 1.1.5. Envoltorios tecnológicos (Wrappers)

Un envoltorio (wrapper) se define como un software que extrae, automática y repetidamente, datos de diferentes fuentes, ya sea páginas web, archivos de texto, bases de datos, entre otros. Este contenido es cambiante, y entrega los datos extraídos a una base de datos o a otra aplicación (Novella & Holubová, 2017).

Los envoltorios son los responsables de transformar la información semiestructurada (información expuesta en la web) en información estructurada; tienen la capacidad de distinguir y sacar objetos que son de interés. Además, son responsables de obtener los datos que fueron extraídos y devolverlos de manera ordenada por medio del lenguaje XML (Extensible Markup Language) (David et al., 2010).

#### 1.1.5.1. Envoltorios basados en API-REST

Las API-REST actualmente son una perspectiva y prestación de servicios web que se usa más en el área de desarrollo. El estilo REST conlleva al uso del protocolo HTTP (Protocolo de Transferencia de Hipertexto), que brinda una sencilla conectividad entre diferentes aplicaciones (Haselmann et al., 2010).

Los lenguajes REST se basan en sustantivos y verbos. Son independientes de cualquier protocolo. Lo datos o recursos pueden ser videos, imágenes, páginas web, cualquier cosa que esté permitida en un sistema informático. No tiene limitación en el formato de presentación (Soni & Ranga, 2019).

A través de esta investigación se ha extraídos información sobre el uso de GraphQL en envoltorios tecnológicos. En el proyecto realizado por Rodríguez, (2020) muestra que estos envoltorios demuestran ser una solución eficiente y flexible. Al tener una manipulación de datos fuerte y personalizada por parte de GraphQL, esto facilita la producción de interfaces entendibles y coherentes para los clientes. Sin embargo es esencial evaluar los requisitos del proyecto, y realizar un diseño y planificación adecuada para garantizar una integración aceptable y efectiva.

## 1.2. Herramientas tecnológicas.

### 1.2.1. BackEnd

#### 1.2.1.1. Lenguaje de programación JavaScript

JavaScript es un lenguaje de programación y su propósito principal es crear páginas web dinámicas. Técnicamente, es interpretado, lo que significa que los programas no necesitan compilarse para ejecutarse. En otras palabras, los programas que son escritos en JavaScript se pueden probar directamente en cualquier navegador sin pasar por procesos intermedios (Eguíluz Pérez, 2019).

Con la mejora y diversificación en tecnologías de red, cada vez más sitios web están cambiando la forma en que brindan sus servicios, lo que resulta en aumentos en la cantidad de aplicaciones web. Como lenguaje con extensas funciones, JavaScript se usa ampliamente en el desarrollo frontend (Fang et al., 2022). Las últimas versiones de JavaScript permiten que sea un lenguaje tanto del lado del cliente como del servidor (Fernando, 2019).

Las ventajas que se dan al usar JavaScript son:

- Es un lenguaje sencillo.
- Es versátil (se lo usa para el desarrollo web dinámico y de aplicaciones móviles).
- Es multiplataforma (computadoras, tablets, móviles, hardware).
- Disminuye el consumo de ancho de banda ya que elimina el peso adicional que un framework genera.

#### 1.2.1.2. Herramienta de desarrollo Node.js

Node.js es una plataforma de código abierto. Permite trabajar con JavaScript a nivel del servidor, con una arquitectura orientada a eventos y basada en el motor Google V8. Esta herramienta está diseñada para ejecutarse en el navegador y ejecutar código JavaScript rápidamente (Flores, 2016).

Es un ambiente de ejecución de JavaScript asíncrono y controlado por eventos (Haro et al., 2019). Utiliza una guía de funciones de entada/salida; convirtiéndolo en fácil y eficiente, útil para el monitoreo en tiempo real de grandes conjuntos de datos generados por aplicaciones que se ejecutan en dispositivos distribuidos (D. Gómez, 2018).

#### 1.2.1.3. Manejador de paquetes de node (npm)

El Manejador de Paquetes de Node (Node Package Manager- npm) es el administrador de recursos predeterminado para Node.js. Está escrito por completo en lenguaje JavaScript. Principalmente, permite la instalación y distribución de módulos de JavaScript en el registro. Los paquetes se procesan en formato CommonJS y se reconocen por la presencia del archivo *package.json* y todos los módulos se clasifican y almacenan en una carpeta llamada *node_modules* (Guanabara et al., 2018).

Cada paquete instalado tiene un conjunto de dependencias que forman la estructura de dependencias de los módulos. Sin embargo, muchos de los paquetes dependen de muchos otros paquetes, por lo que npm trabaja con una jerarquía de versiones requerida para cada

uno de ellos. También calcula la versión específica para cumplir con todos los requisitos que se usan en el proyecto (Lixandru et al., 2022).

#### 1.2.1.4. Twitch

Una de las principales funciones de Twitch es la transmisión de videojuego, eSports (deportes electrónicos) y eventos relacionados con otros videojuegos. Actualmente, la plataforma empieza a tener una audiencia habitual con otros contenidos, como los denominados estilos de vida; es propiedad de Amazon desde 2014 (Antolín-Prieto et al., 2021).

Tras obtener la posición líder entre las plataformas de transmisión en vivo, ha revolucionado los procesos de comunicación de los medios tradicionales con el avance tecnológico. Esta nueva tendencia atrae la atención de viejos y nuevos medios, que intentan adaptar su contenido a la red (Gutiérrez, 2022).

**API REST de la plataforma**

Twitch brinda una interfaz de línea de comandos para la gestión de su contenido. Se lo puede usar para consumir puntos finales, obtener tokens de acceso OAuth e inspeccionar eventos EventSub. La API de Twitch proporciona herramientas y datos necesarios para desarrollar integraciones de la plataforma. Los modelos y sistemas de datos se diseñaron para facilitar el acceso a datos relevantes de una manera sencilla, consistente y confiable (Twitch Developers, 2022).

En el año 2017 se desarrolló una nueva versión de la API de Twitch ya que la comunidad de desarrolladores ha creado herramientas usando esta plataforma, pero en el proceso, se generaron problemas que están relacionados con la confiabilidad, algunos cambios importantes que no se anunciaron e inconsistencias frecuentemente (Dallas Teste, 2017). Por ejemplo, ((Eddie, 2020)) menciona en el blog de la plataforma sobre errores que se manifiestan, como el tiempo de espera. Al surgir este error, se produce una tasa de error que alcanza el 10% desde que comienza el evento, donde antes era del 0%.

**Recursos básicos**

Los principales recursos para usarse de la API-REST de Twitch se detallan en la Tabla 1.

**Tabla 1**

*Recursos principales de la API de Twitch.*

| Recurso | Descripción | Url |
|---|---|---|
| **Transmisiones en vivo** | Esta funcionalidad obtiene una lista de todas las transmisiones que se transmiten en vivo. | https://api.twitch.tv/helix/streams |
| **Videos por juego** | Es un recurso que ayuda a obtener la información sobre uno o más videos publicados relacionados a un juego en específico. | https://api.twitch.tv/helix/videos?game_id=491931 |
| **Clips por usuario** | Se trata de un conjunto de clips de video (videos cortos) que fueron capturados durante una transmisión de un usuario en específico. | https://api.twitch.tv/helix/clips?broadcaster_id=95792026 |
| **Información del canal de un usuario** | En este caso, se consigue la información de uno o más canales a través de un id de un usuario. | https://api.twitch.tv/helix/channels?broadcaster_id=95792026 |
| **Información de un juego** | Gracias a este recurso, se adquiere información sobre un juego requerido. | https://api.twitch.tv/helix/games?id=491931 |

*Nota.* (Twitch Developers, 2022)

García & Hidalgo, (2021) realizaron un trabajo donde se comprueba Node.js con JamStack, la cual es una arquitectura de desarrollo web. Al realizar el respectivo desarrollo se concluye que JamStack es eficiente para proyectos web, pero Node.js es útil en aquellas implementaciones que requieran una gran demanda de procesamiento. Por lo cual, Node.js es fundamental para realizar este envoltorio ya que se hace uso de funcionalidades del API-REST, donde se extrae una gran cantidad de datos.

### 1.2.2. FrontEnd

#### 1.2.2.1. Apollo Client

Apollo Client es una biblioteca parala gestión de estado de JavaScript que permite administrar datos de GraphQL, tanto remotos como locales. Ofrece el almacenamiento en caché y modificación de los datos de la aplicación, realizar todo este proceso mientras actualiza automáticamente la interfaz de usuario. Además, ayuda a estructurar el código de una manera rentable, declarativa y predecible que es consistente en las prácticas de desarrollo usadas actualmente (Apollo, 2022).

Ayuda a la parte del cliente del proyecto (Frontend) a comunicarse los recursos y funcionalidades de GraphQL. Ayuda a los más destacados framework, tal como Angular, Vue, React, Android o iOS (Engineering, 2020).

Además, tiene un apartado de cache, en donde se pueden almacenar los datos extraídos de una consulta, con el objetivo de que al realizar nuevamente una consulta, se lo haga de manera rápida. Esto permite reducir la cantidad de solicitudes al servidor, el tiempo de respuesta de las consultas y mejorar el rendimiento de la aplicación (Glasbergen et al., 2018). El uso de Apollo y su cache en el presente trabajo, proporciona una solución para mejorar el rendimiento de las aplicaciones GraphQL. La cache integrada ayuda a minimizar la sobrecarga de solicitudes y mejorar la experiencia de usuario.

### 1.3. Metodologías de desarrollo de software.

La facilidad y el dinamismo actual en la industria del software genera un replanteamiento de los fundamentos a cerca del desarrollo de software clásico. Investigaciones recientes establecen tendencias en el desarrollo del enfatizando la necesidad de velocidad, flexibilidad y aspectos externos que son esenciales para crear una ventaja competitiva de una mayor productividad y capacidad de respuesta a las necesidades del cliente en el menor tiempo posible con el objetivo de proporcionar mayor valor comercial (Maida & Pacienzia, 2015).

Los métodos de desarrollo tradicionales más utilizados son los de tipo cascada y modelos espirales, mientras que entre los métodos ágiles modernas, son los más manejados, incluida la programación extrema, scrum, el desarrollo orientado a funcionalidades y las basadas en componentes (Milena Velásquez Restrepo et al., 2019).

La principal ventaja de los métodos ágiles es la flexibilidad, los proyectos en desarrollo se desglosan en proyectos más pequeños, incorpora una comunicación frecuente con el usuario, son muy colaborativos y es adaptable a los cambios (Molina Montero et al., 2018). Algunas de las metodologías usadas son: las más común Scrum, Cascada, Kanban, Spiral, etc.

#### 1.3.1. Scrum

Scrum es la metodología de desarrollo ágil más popular en la actualidad, se ha utilizado en una amplia gama de entornos y se utiliza en una variedad de entornos y para diversos propósitos, tanto dentro como fuera del contexto tradicional de desarrollo de software (Hron & Obwegeser, 2022). Organizaciones escogen Scrum para cumplir con sus requisitos de software. Así mismo, está facilitando a las industrias de software el desarrollo aplicaciones

que cumplan los requisitos planteados (Butt et al., 2022). El marco técnico de Scrum se conforma de:

- **Roles**

  Los roles de Scrum se muestran en la Tabla 2.

**Tabla 2**

*Roles de Scrum.*

| Rol | Descripción |
|---|---|
| **Scrum Master** | Es la persona responsable de administrar el proceso de Scrum y garantizar que el proceso funcione. Así mismo, la eliminación de obstáculos que impiden conseguir el objetivo. |
| **Product Owner** | Es el responsable de maximizar y optimizar el valor del producto y el trabajo del equipo de desarrollo. Además, administra el flujo de valor del producto por medio del Product Backlog. |
| **Equipo de desarrollo** | Este equipo generalmente consta de 3 a 9 profesionales que trabajan en la creación de los incrementos o requisitos del producto al final de cada sprint. |

*Nota*. (Baham, 2019).

- **Artefactos**

  Los artefactos que se usan en Scrum se muestran en la Tabla 3.

**Tabla 3**

*Artefactos de Scrum.*

| Artefacto | Descripción |
|---|---|
| **Product Backlog o Pila de producto** | Es una lista ordenada sobre lo que es necesario para mejorar el producto. Esta fuente de trabajo es realizada por el *Scrum Team*. Se realiza un refinamiento del Product Backlog con el objetivo de dividir en y definir elementos más pequeños . |
| **Sprint Backlog o Pila de Sprint** | Está compuesto por los Objetivos del *Sprint* (por qué), por el conjunto de componentes del *Product Backlog* que fueron seleccionados para el *Sprint* (qué) y un plan de acción para entregar el *Increment* (cómo). |

| | |
|---|---|
| **Incremento** | El *Sprint Backlog* es un proyecto hecho por y para desarrolladores ya que muestra lo que se hará para cumplir el Objetivo del *Sprint.* Un *Increment* es un paso concreto para llegar al Objetivo del producto. Se puede generar varios dentro de un *Sprint.* |

*Nota*. (Schwaber & Sutherland, 2020).

- **Fases**

  Las fases que se aplican en Scrum se observan en la Tabla 4.

**Tabla 4**

*Fases de Scrum.*

| Nombre | Descripción |
|---|---|
| **Sprint** | Este es el elemento central de la metodología de desarrollo y lleva menos de 4 semanas completarlo. El Sprint comienza inmediatamente después del final del anterior hasta que se cumplan los requisitos funcionales contemplados en el Product Backlog. |
| **Sprint Planning Meeting** | En esta reunión se planifican las actividades a realizar en el próximo Sprint. El rol de Scrum Master es fundamental ya que es responsable de que todos los participantes entiendan claramente el propósito y la finalidad de este. |
| **Daily Scrum** | Esta es una de las características clave de gestión de la metodología Scrum. Es una reunión que dura hasta 15 minutos, en la cual se planean actividades para las próximas 24 horas. |
| **Sprint Review** | Esta reunión se debe realizar con un tiempo que no sobrepase las 4 horas. Uno de los objetivos es buscar factores que se puedan mejorar en el próximo Sprint, optimizando así el proceso de desarrollo. |
| **Sprint Retrospective** | Esta reunión, tiene lugar después del Sprint Review y antes de iniciar un nuevo Sprint. Debe tener una duración máxima de tres 3 horas, durante las cuales se analiza el comportamiento y la relación entre los miembros del equipo. |

*Nota*.(Arias & Durango, 2018) .

Para la gestión de este proyecto se usa Scrum, en la Tabla 5 se muestra las ventajas de usar Scrum frente a otras metodologías.

**Tabla 5**

*Comparativa de metodologías de gestión de proyectos.*

| Scrum | Cascada (waterfall) | Spiral |
|---|---|---|
| Adaptabilidad al cambio | Menor adaptabilidad al cambio | Enfoque iterativo y manejo del riesgo |
| Enfoque en la entrega de valor | Enfoque en la entrega final del producto | Mayor flexibilidad en el desarrollo |
| Mayor colaboración y comunicación | Menor colaboración y comunicación | Mayor enfoque en la mejora continua |
| Iteraciones cortas y frecuentes | Secuencialidad rígida de fases | Identificación temprana de problemas y ajustes |
| Mayor transparencia y visibilidad | Menor visibilidad del progreso | Mayor adaptabilidad a cambios en los requisitos |
| Mayor flexibilidad en el desarrollo | Menor flexibilidad en el desarrollo | Mayor enfoque en la mitigación del riesgo |
| Enfoque en la mejora continua | Enfoque en la planificación inicial | Mayor enfoque en la resolución progresiva de riesgos |
| Mayor enfoque en la calidad | Mayor riesgo de problemas post entrega | |

## 1.4. Experimentación en la ingeniería de software (IS)

Los primeros intentos para la aplicación de este enfoque se remontan a la década de los 60’s y 70’s, cuando varios investigadores evaluaron tecnologías de soporte de ingeniería de software. El objetivo radica en conocer las causas por las que se producen ciertos efectos. (O. S. Gómez et al., 2018). La experimentación ocurre cuando el objetivo es controlar situaciones y manipular el comportamiento de manera directa, precisa y sistemática. Existen lineamientos para realizar experimentos en ingeniería de software, como los por Wohlin, Jedlitschka, Singer, Juristo y Moreno y Kitchenham (Mera, 2022).

### Metodología de Wohlin

El proceso de realizar un experimento se puede llevar a cabo de varias maneras. La ventaja de esto es que los resultados de la recopilación de datos se pueden obtener directamente en la reunión, sin necesidad de contactar a los participantes y luego solicitar sus resultados

individuales. También, realiza un experimento en tecnologías existentes sobre las nuevas. Sus fases ayudan a tener una planificación estructurada, un diseño eficiente del experimento, una ejecución y análisis preciso. Las fases de la metodología de Wohlin son las siguientes (Wohlin et al., 2012).

- **Alcance**

  Se realiza la definición de objetivos que se va a alcanzar en el experimento.

- **Planificación**

  Se realiza la selección del contexto, formulación de hipótesis, selección de variables, selección de sujetos, diseño de experimento, instrumentación y evaluación de validez.

- **Operación**

  Se prepara el experimento, se realiza la ejecución y la validación de datos.

- **Análisis e interpretación**

  Se aplica la estadística descriptiva, se hace una reducción de datos y la prueba de hipótesis.

Gracias a que la metodología de Wohlin se usa para evaluar tecnologías emergentes y nuevas, y determinar su efectividad y eficiencia en relación con tecnologías existentes, lo hace importante para aplicarla en el desarrollo del envoltorio. Esto porque se hace una comparación de un envoltorio realizado por una nueva tecnología que es GraphQL frente a una API-REST, la cual se usa actualmente.

### 1.5. Estándar ISO/IEC 25000

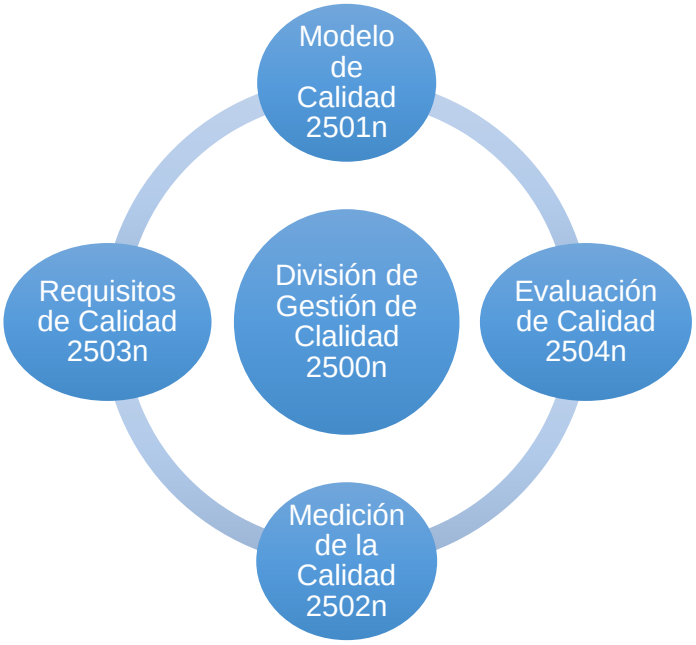
La calidad del software ha cobrado importancia, dando como resultado el continuo desarrollo de estándares internacionales para asegurar dicha calidad en cualquier producto de software. Una de las más adoptadas, la familia de normas ISO/IEC 25000 tiene como fin implantar un sistema de evaluación de la calidad de los productos de software. La calidad de los elementos de una utilidad de software define su calidad (Oliveira et al., 2019).

Las organizaciones se esfuerzan por mejorar el desarrollo y mantenimiento de productos de software cumpliendo las necesidades de los usuarios por medio de métodos y sistemas. Las instituciones de educación superior, en particular, se topan con desafíos como mantener

una infraestructura de TI estable, flexible y segura para respaldar la mejora continua de los procesos institucionales (Guerrero et al., 2022). La norma 25000 contiene algunas divisiones y se lo muestra en la Figura 15.

**Figura 15**

*Divisiones de la norma ISO/IEC 25000.*




*Nota*: (Perdomo & Zapata, 2021)

### 1.5.1. Estándar ISO/IEC 25023 (Medición de calidad)

Para hacer una evaluación de software se debe desarrollar a partir de un modelo que permita medir su nivel de calidad (Llamuca-Quinaloa et al., 2021). La ISO/IEC 25023 (Medición de la calidad del sistema y del producto de software) define específicamente métricas para medir la calidad de sistemas y del producto de software. Estas métricas forman parte de los atributos de calidad de desempeño y son: comportamiento en el tiempo, utilización de recursos y capacidad (Iso 25000, 2018).

La usabilidad se basa en la ISO/IEC 25000 y se utiliza para determinar hasta qué punto los usuarios pueden utilizar un sistema. Esta se compone de 6 subcaracterísticas, como se observa en la Tabla 6.

**Tabla 6**

*Subcaracterísticas de la usabilidad de la ISO/IEC 25023.*

| SUBCARACTERÍSTICA | DESCRIPCIÓN |
|---|---|
| **Appropriateness Recognisability (Idoneidad Reconocibilidad)** | Se usa para comprobar si el sistema está acorde a los requisitos del usuario. |
| **Learnability (Capacidad de aprendizaje)** | Se usa para medir si un sistema puede ser usado con el fin de alcanzar objetivos. |
| **Operability (Operatividad)** | Se lo aplica para comprobar si el sistema contiene varios atributos que puedan hacer su uso más sencillo para el usuario. |
| **User Error Protection (Protección contra errores del usuario)** | Se emplea con el objetivo de proteger a los usuarios de los errores que pueda generar el sistema. |
| **User Interface Aesthetics (Estética de la interfaz de usuario)** | Se aplica para aprobar la interfaz de usuario, que sea de agrado y satisfacción. |
| **Accessibility (Accesibilidad)** | Se lo usa para verificar si el sistema puede ser accesible para usuarios con discapacidad. |

*Nota*. (Pradanita et al., 2019).

Frente a las demás métricas, Aziz et al., (2018) menciona en su proyecto realizado sobre una medición de la calidad de software que en base a los resultados en la evaluación que consta de 3 etapas se concluye en tres puntos de recomendación: prevención de la corrupción de datos, validez de accesos a las matrices y conformidad. Lo que hace útil realizar un análisis de un proyecto basado en la ISO/IEC 25023.

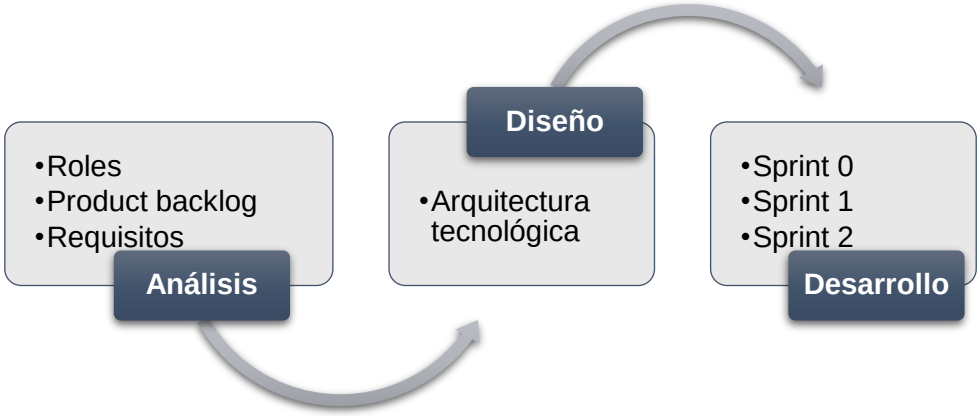
# CAPÍTULO 2

## Desarrollo

La fase de desarrollo del presente proyecto se enfoca en dos proyectos hechos en base al marco Scrum. El primero es un servidor GraphQL que envuelve los servicios REST de la API de Twitch. El segundo es un cliente, el cual consume los servicios y recursos del envoltorio GraphQL y la API-REST de Twitch, con el objetivo de realizar un experimento donde se compare las dos herramientas y el principal factor será le tiempo de respuesta, se usará la metodología de Wohlin para el proceso experimental. En la Figura 16 se muestra el proceso de desarrollo del proyecto.

**Figura 16**

*Estructura de fase de desarrollo del proyecto.*




### 2.1. Análisis

#### 2.1.1. Roles de Proyecto

El primer paso es identificar el equipo de trabajo y las personas que están interesadas en el proyecto, para la asignación de responsabilidades y roles. Para el desarrollo del envoltorio GraphQL de Twitch y el experimento, se compone de un grupo de trabajo que se muestra en la Tabla 7.

**Tabla 7**

*Roles del proyecto.*

| Miembro | Rol | Descripción |
|---|---|---|
| PhD. Antonio Quiña | Propietario del Producto (Product Owner) | Responsable de evaluar los requisitos del proyecto, director del trabajo y |

| | | |
|---|---|---|
| | | docente de la carrera de Ingeniería en Software de la Universidad Técnica del Norte. |
| Sr. Alexander Véliz | Jefe del proyecto (Scrum Master) | Responsable de revisar el avance del proyecto y estudiante de la carrera de Ingeniería en Software de la Universidad Técnica del Norte. |
| Sr. Alexander Véliz | Equipo de desarrollo (Development Team) | Encargado del desarrollo del software y estudiante de la carrera de Ingeniería en Software de la Universidad Técnica del Norte. |

### 2.1.2. Definición del Product Backlog

En el product backlog se determinan las características para el desarrollo. En este caso se utiliza la técnica T-Shirt Sizes. Esta es una técnica de evaluación donde el programador clasifica el tamaño de cada función, se refiere al tamaño de la camiseta en relación con el esfuerzo requerido para completar la historia de usuario, que está determinada por el tiempo (Caillamara Encalada & Lalangui Campoverde, 2023). A continuación, en la Tabla 8 se muestra la estimación de tiempo y talla

**Tabla 8**

*Técnica T-Shirt Size.*

| Talla de camiseta (T-Shirt Size) | Estimación de esfuerzo |
|---|---|
| XS | 0-10 horas |
| S | 10-20 horas |
| M | 20-30 horas |
| L | 30-40 horas |
| XL | 40-48 horas |
| XXL | 48-60 horas |

En la Tabla 9 se exponen los requerimientos definidos en el orden según el ID de la Historia de Usuario.

**Tabla 9**

*Product Backlog.*

| ID HU | Prioridad | Historia de Usuario | Estimación |
|---|---|---|---|
| HU-01 | Alta | Definición de estructura del software | S |
| HU-02 | Alta | Gestión del recurso del token de aplicación | XS |
| HU-03 | Alta | Gestión del recurso de transmisiones en vivo | S |
| HU-04 | Alta | Gestión del recurso de videos que tiene un juego | M |
| HU-05 | Alta | Gestión del recurso de clips que tiene un usuario | M |
| HU-06 | Alta | Gestión del recurso de información del canal | S |
| HU-07 | Alta | Gestión del recurso de información de un juego | S |
| HU-08 | Media | Gestión de Menús de selección | XS |
| HU-09 | Alta | Gestión del Nivel 1 | L |
| HU-10 | Alta | Gestión del Nivel 2 | L |
| HU-11 | Alta | Gestión del Nivel 3 | L |

### 2.1.3. Requisitos del proyecto

**Historias de Usuario (HU)**

Los requisitos para el envoltorio fueron levantados por el propietario del proyecto usando Historias de Usuario, las cuales son una herramienta de Scrum.

**Tabla 10**

*Historia de Usuario 1.*

| Historia de Usuario | |
|---|---|
| **Número:** HU-01 | **Nombre:** Definición de estructura del software |
| **Usuario:** | |
| **Prioridad en negocio:** Alta | **Estimación:** S |
| **Descripción:**<br>Establecer las principales funcionalidades de la API-REST de Twitch para el desarrollo del envoltorio a través de pruebas de consumo. Así, detallando la información para poder hacer uso de las funciones seleccionadas que sean convenientes para el software. | |

| |
|---|
| **Criterios de aceptación:**<br>• La autenticación de la API de Twitch debe estar documentada de manera adecuada para que sea entendible.<br>• Las funciones seleccionadas deben ser sencillas de entender para su posterior uso.<br>• Conocer la información que no es fundamental pero que son necesarios para el funcionamiento de la plataforma Twitch. |
| **Dependencias:** Ninguna |

**Tabla 11**

*Historia de Usuario 2.*

| **Historia de Usuario** | |
|---|---|
| **Número:** HU-02 | **Nombre:** Gestión del recurso del token de aplicación |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** XS |
| **Descripción:**<br>Cómo desarrollador quiero obtener el servicio de la API-REST de la plataforma Twitch y el servicio GraphQL (envoltorio) para tener el token o clave para acceder a los datos de las funcionalidades a usarse para el proyecto. | |
| **Criterios de aceptación:**<br>• Tener una opción que permita generar el token o la clave.<br>• Mostrar el token obtenido para usarse en las próximas funcionalidades en caso de ser necesario. | |
| **Dependencias:** Ninguna | |

**Tabla 12**

*Historia de Usuario 3.*

| **Historia de Usuario** | |
|---|---|
| **Número:** HU-03 | **Nombre:** Gestión del recurso de transmisiones en vivo |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** S |
| **Descripción:**<br>Cómo desarrollador quiero generar y obtener los datos de las transmisiones en vivo de Twitch a través de la API-REST y el servicio GraphQL (envoltorio). Este servicio debe obtener los datos de los usuarios que están transmitiendo en vivo en la plataforma. Además, debe tener la capacidad de devolver la cantidad de datos especificada. Así este recurso se lo establece como el Nivel 1 de consumo. | |
| **Criterios de aceptación:** | |

- Poseer opciones donde se elija la opción de consumo como el consumo de los servicios de la API-REST tanto sin caché como con caché. De la misma manera con el servicio de GraphQL.
- Tener una opción para obtener los datos de transmisiones en vivo.
- Brindar una opción donde se escoja la distribución del límite de datos para cada nivel.
- La obtención del listado de datos generará una respuesta satisfactoria e información donde se muestre el tiempo de respuesta en milisegundos, segundo y minutos.
- Mostrar la cantidad de solicitudes totales realizadas.
- Indicar la cantidad de datos total.

**Dependencias:** HU-02

**Tabla 13**

*Historia de Usuario 4.*

| **Historia de Usuario** | |
|---|---|
| **Número:** HU-04 | **Nombre:** Gestión del recurso de videos que tiene un juego |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** M |
| **Descripción:**<br>Cómo desarrollador quiero generar y obtener los datos de los videos que tiene un juego a través de la API-REST de Twitch y el servicio GraphQL (envoltorio), el cual debe tener los datos de los videos que tiene un juego en específico. Además, debe tener la capacidad de devolver la cantidad de datos especificada y en caso de que el id dado no tenga datos, mostrar un mensaje informativo. Así este recurso se lo establece como el Nivel 2 de consumo. | |
| **Criterios de aceptación:**<br>• Poseer opciones donde se elija la opción de consumo como el consumo de los servicios de la API-REST tanto sin caché como con caché. De la misma manera con el servicio de GraphQL.<br>• Tener una opción para obtener los datos de los videos que tiene un juego.<br>• Brindar una opción donde se escoja la distribución del límite de datos para cada nivel.<br>• La obtención del listado de datos generará una respuesta satisfactoria e información donde se muestre el tiempo de respuesta en milisegundos, segundo y minutos.<br>• Mostrar la cantidad de solicitudes totales realizadas.<br>• Indicar la cantidad de datos total y la distribución que se proporcionó por nivel. | |
| **Dependencias:** HU-02, HU-03 | |

**Tabla 14**

*Historia de Usuario 5.*

| Historia de Usuario | |
|---|---|
| **Número:** HU-05 | **Nombre:** Gestión del recurso de clips que tiene un usuario |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** M |
| **Descripción:**<br>Cómo desarrollador quiero generar y obtener los datos de los clips o cortos videos que tiene un usuario a través de la API-REST de Twitch y el servicio GraphQL (envoltorio). Este servicio debe obtener los clips que tiene un usuario en específico. Además, el id ingresado debe tener la capacidad de devolver la cantidad de datos ingresada, caso contrario mostrar un mensaje de error. Así este recurso se lo establece como el Nivel 3 de consumo. | |
| **Criterios de aceptación:**<br>• Poseer opciones donde se elija la opción de consumo como el consumo de los servicios de la API-REST tanto sin caché como con caché. De la misma manera con el servicio de GraphQL.<br>• Tener una opción para obtener los datos de los clips que tiene un usuario.<br>• Brindar una opción donde se escoja la distribución del límite de datos para cada nivel.<br>• La obtención del listado de datos generará una respuesta satisfactoria e información donde se muestre el tiempo de respuesta en milisegundos, segundo y minutos.<br>• Mostrar la cantidad de solicitudes totales realizadas.<br>• Indicar la cantidad de datos total y la distribución que se proporcionó por nivel. | |
| **Dependencias:** HU-02, HU-03, HU-04 | |

**Tabla 15**

*Historia de Usuario 6.*

| Historia de Usuario | |
|---|---|
| **Número:** HU-06 | **Nombre:** Gestión del recurso de información del canal de un usuario |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** S |
| **Descripción:**<br>Cómo desarrollador quiero extraer los datos relacionados a la información del canal de un usuario a través de la API-REST de Twitch y el servicio GraphQL (envoltorio). Este servicio debe obtener la información de un usuario específico. Así este recurso se lo establece como el Nivel 4 de consumo. | |
| **Criterios de aceptación:**<br>• Poseer opciones donde se elija la opción de consumo como el consumo de los servicios de la API-REST tanto sin caché como con caché. De la misma manera con el servicio de GraphQL.<br>• Tener una opción para obtener la información del canal de un usuario.<br>• Brindar una opción donde se escoja la distribución del límite de datos para cada nivel. | |

- La obtención del listado de datos generará una respuesta satisfactoria e información donde se muestre el tiempo de respuesta en milisegundos, segundo y minutos.
- Mostrar la cantidad de solicitudes totales realizadas.
- Indicar la cantidad de datos total y la distribución que se proporcionó por nivel.

**Dependencias:** HU-02, HU-03, HU-04, HU-05

**Tabla 16**

*Historia de Usuario 7.*

| **Historia de Usuario** | |
|---|---|
| **Número:** HU-07 | **Nombre:** Gestión del recurso de información de un juego |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** S |
| **Descripción:**<br>Cómo desarrollador deseo extraer los datos relacionados a la información de un juego a través de la API-REST de Twitch y el servicio GraphQL (envoltorio). Este servicio debe obtener la información un juego específico. Así este recurso se lo establece como el Nivel 5 de consumo. | |
| **Criterios de aceptación:**<br>• Poseer opciones donde se elija la opción de consumo como el consumo de los servicios de la API-REST tanto sin caché como con caché. De la misma manera con el servicio de GraphQL.<br>• Tener una opción para obtener la información de un juego.<br>• Brindar una opción donde se escoja la distribución del límite de datos para cada nivel.<br>• La obtención del listado de datos generará una respuesta satisfactoria e información donde se muestre el tiempo de respuesta en milisegundos, segundo y minutos.<br>• Mostrar la cantidad de solicitudes totales realizadas.<br>• Indicar la cantidad de datos total y la distribución que se proporcionó por nivel. | |
| **Dependencias:** HU-02, HU-03, HU-04, HU-05, HU-06 | |

**Tabla 17**

*Historia de Usuario 08.*

| **Historia de Usuario** | |
|---|---|
| **Número:** HU-08 | **Nombre:** Gestión de Menús de selección |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Media | **Estimación:** XS |
| **Descripción:**<br>Cómo desarrollador quiero elegir el tipo de consumo que deseo hacer, el nivel y el número de caso requerido. | |

| |
|---|
| **Criterios de aceptación:**<br>• El número máximo de niveles son 5.<br>• El límite de datos se lo ingresa en cada nivel.<br>• El número de opciones de consumo son 4.<br>• Tener opción para consumir desde la API-REST o el envoltorio GraphQL. Así mismo como escoger si se consume con cache. |
| **Dependencias:** Ninguna |

**Tabla 18**

*Historia de Usuario 09.*

| **Historia de Usuario** | |
|---|---|
| **Número:** HU-09 | **Nombre:** Gestión del Nivel 1 |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** L |
| **Descripción:**<br>Cómo desarrollador quiero consultar el Nivel 1 con la cantidad de datos especificada del API-REST y del envoltorio GraphQL de Twitch. Además, quiero obtener el tiempo de respuesta para realizar su respectiva comparación. | |
| **Criterios de aceptación:**<br>• Mostrar el tiempo de respuesta al consumir.<br>• El tiempo debe mostrarse en milisegundos, segundos y minutos.<br>• El nivel 1 tomará en cuenta la petición que se realiza en la Historia de Usuario 3. | |
| **Dependencias:** HU-02, HU-03 | |

**Tabla 19**

*Historia de Usuario 10.*

| **Historia de Usuario** | |
|---|---|
| **Número:** HU-10 | **Nombre:** Gestión del Nivel 2 |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** XL |
| **Descripción:**<br>Cómo desarrollador quiero consultar el Nivel 2 con la cantidad de datos especificada del API-REST y del envoltorio GraphQL de Twitch. Además, quiero obtener el tiempo de respuesta para realizar su respectiva comparación. | |
| **Criterios de aceptación:**<br>• Mostrar el tiempo de respuesta al consumir.<br>• El tiempo debe mostrarse en milisegundos, segundos y minutos.<br>• El nivel 2 tomará en cuenta la petición que se realiza en la Historia de Usuario 4 con los datos obtenidos de la Historia de Usuario 3. | |
| **Dependencias:** HU-02, HU-03, HU-04 | |

**Tabla 20**

*Historia de Usuario 11.*

| **Historia de Usuario** | |
|---|---|
| **Número:** HU-11 | **Nombre:** Gestión del Nivel 3. |
| **Usuario:** Desarrollador | |
| **Prioridad en negocio:** Alta | **Estimación:** XL |
| **Descripción:**<br>Cómo desarrollador quiero consultar el Nivel 3 con la cantidad de datos especificada del API-REST y del envoltorio GraphQL de Twitch. Además, quiero obtener el tiempo de respuesta para realizar su respectiva comparación. | |
| **Criterios de aceptación:**<br>• Mostrar el tiempo de respuesta al consumir.<br>• El tiempo debe mostrarse en milisegundos, segundos y minutos.<br>• El nivel 3 tomará en cuenta la petición que se realiza en la Historia de Usuario 5 con los datos obtenidos de la Historia de Usuario 3 y 4 respectivamente. | |
| **Dependencias:** HU-02, HU-03, HU-04, HU-05 | |

## 2.2. Diseño

El desarrollo está compuesto por Sprints. Se define el diseño del proyecto, el cual es denominado como Sprint 0 en donde se establece la arquitectura tecnológica. Se muestra en la Tabla 21.

**Tabla 21**

*Detalle Sprint 0.*

| Sprint | Fecha inicio | Fecha fin | Duración |
|---|---|---|---|
| Sprint 0 | 24-nov 2022 | 05-dic 2022 | 20 |

**Planificación del Sprint 0**

Reunión para la planificación del proyecto.

**Fecha:** 24/11/2022

**Objetivo:** Definir la arquitectura tecnológica del proyecto.

En la Tabla 22 se observa la planificación del presente Sprint.

**Tabla 22**

*Planificación Sprint 0.*

| Fase de desarrollo | Tarea | Estimación de tiempo |
|---|---|---|
| Análisis | Seleccionar la API pública para el envoltorio. | 3 |
| Análisis | Escoger las funcionalidades y definir los niveles de consulta. | 3 |
| Diseño | Determinar la arquitectura tecnológica del proyecto. | 5 |
| Planificación | Detallar las tareas a realizar en el Sprint actual. | 4 |
| Revisión | Revisar los resultados del desarrollo del Sprint | 3 |
| Retrospectiva | Analizar los resultados del Sprint. | 2 |
| | **TOTAL HORAS** | **20** |

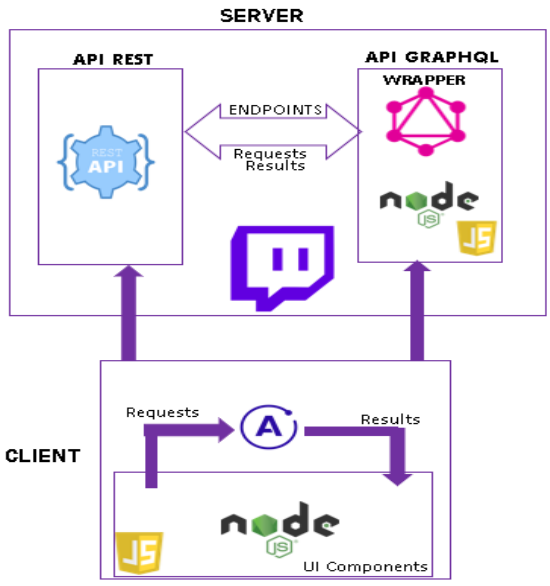
**Revisión del Sprint 0**

### 2.2.1. Arquitectura tecnológica

La arquitectura se compone de dos proyectos, el servidor y el cliente. El servidor que es el envoltorio GraphQL y de parte del cliente es el que consume los dos servicios para realizar la comparación con respecto al tiempo de respuesta de la petición. El servidor y el cliente están hechos con Node.js con la implementación de tecnologías necesarias para su funcionamiento. En la Figura 17 se puede observar la arquitectura.

**Figura 17**

*Arquitectura tecnológica.*




**Niveles**

Las funcionalidades para usar se muestran en la Tabla 1, sección 1.2.1.4, cada funcionalidad se define como un nivel de consumo.

- **Nivel 1:** en este nivel, la funcionalidad a usar es “Transmisiones en Vivo”, la cual obtiene una lista de los canales que están transmitiendo en tiempo real.
- **Nivel 2:** como funcionalidad principal para este nivel, se selecciona el servicio “Video por Juego”. Para mostrar los videos que tiene un juego en específico, se utiliza el Nivel 1 para extraer los IDs de un juego. Así, consumir esta funcionalidad y listar los datos.
- **Nivel 3:** Para este nivel, la funcionalidad es “Clips por Usuario”. Se necesita del Nivel 2 para obtener los datos ya que muestra los IDs de los usuarios. De esta manera se consume el servicio, listando los clips que tiene un usuario en específico.

## 2.3. Desarrollo

El proyecto se lo hizo por partes, las cuales son los Sprints, cada uno tiene una duración de 30 horas con excepción del Sprint 3 y 7, los cuales tienen una duración de 45 horas. En la Tabla 23 se muestra el detalle de los Sprints.

**Tabla 23**

*Detalle general de los Sprints.*

| Sprint | Fecha inicio | Fecha fin | Duración |
|---|---|---|---|
| Sprint 1 | 06-dic 2022 | 03-ene 2023 | 40 |
| Sprint 2 | 04-ene 2023 | 24-ene 2023 | 40 |
| Sprint 3 | 25-ene 2023 | 14-feb 2023 | 40 |
| Sprint 4 | 15-feb 2023 | 02-mar 2023 | 30 |
| Sprint 5 | 06-mar 2023 | 21-mar 2023 | 30 |

### 2.3.1. Sprint 1

**Planificación del Sprint 1**

Se establece las tareas para la investigación de la API actual de Twitch con el fin de generar información que contenga datos sobre el uso de los servicios de esta, que sirva como recurso para el desarrollo del producto.

**a) Reunión de planificación**

**Fecha:** 06 de diciembre de 2022.

**Objetivo:** Planificación y creación del Sprint Backlog.

**b) Sprint backlog**

En la Tabla 24 se muestra el desarrollo del Sprint 1 Backlog.

**Tabla 24**

*Sprint 1 Backlog.*

| Historia de usuario | Fase de desarrollo | Tarea | Estimación de tiempo |
|---|---|---|---|
| OTROS | Desarrollo | Crear el proyecto Node.js, establecer las librerías necesarias para el servidor y GraphQL y componer la estructura de archivos y carpetas para el proyecto. | 3 |
| | Desarrollo | Llevar a cabo la creación del servidor. | 3 |
| | Pruebas | Comprobar el funcionamiento del servidor. | 1 |
| HU-02 | Desarrollo | Crear un type Token con los campos similares a los de la API-REST de Twitch en el archivo schema. | 2 |
| | Desarrollo | Crear el type Query con la operación getToken, el cual se usará en los resolutores. | 2 |
| | Desarrollo | Poner en marcha los resolutores para las operaciones establecidas en el type Query y extraer el token. | 3 |
| | Pruebas | Pruebas de concepto de la implementación. | 2 |
| HU-03 | Desarrollo | Crear un type LiveStreams con los campos similares a los de la API-REST de Twitch en el archivo schema. | 2 |
| | Desarrollo | Generar la operación en el type Query para su próximo uso en los resolutores. | 2 |
| | Desarrollo | Implementar la función para extraer los datos respectivos de las transmisiones en vivo en los resolutores. | 3 |
| | Pruebas | Pruebas de concepto de la implementación. | 2 |
| HU-04 | Desarrollo | Crear un type VideosByGame con los campos similares a los de la API-REST de Twitch en el archivo schema. | 2 |
| | Desarrollo | Generar la operación en el type Query para su próximo uso en los resolutores. | 2 |
| | Desarrollo | Implementar la función para extraer los datos respectivos de los videos que tiene un juego en específico en los resolutores. | 3 |
| | Pruebas | Pruebas de concepto de la implementación. | 2 |
| Eventos | Planificación | Detallar las tareas a realizar en el Sprint actual. | 3 |
| | Revisión | Revisar los resultados del desarrollo del Sprint. | 2 |
| | Retrospectiva | Analizar los resultados del Sprint. | 1 |
| | | **TOTAL HORAS** | **40** |

**Revisión del Sprint 1**

La finalización del primer Sprint estableció la estructura para el desarrollo del envoltorio.

**a) Reunión de revisión**

**Fecha:** 03 de enero de 2023.

**Resultado:** Revisión del desarrollo del crecimiento del producto.

**b) Incremento del producto entregable**

**URL base de la API-REST Twitch:** https://api.twitch.tv/helix/.

**Autenticación**

La API de Twitch utiliza tokens de acceso de OAuth 2.0 para tener entrada a los recursos. Usa dos tipos de tokens de acceso: tokens de acceso de usuario y tokens de acceso a la aplicación. El contenido de funcionalidades para cada API establece el tipo de token que se debe usar para permitir acceder al recurso. Algunas funcionalidades requieren el token de usuario y otros el de aplicación (Twitch Developers, 2022).

**RECURSO TOKEN**

Para obtener un token de aplicación se realiza una consulta llamada "*getToken*", en donde como salida se obtiene un tipo "*Token*".

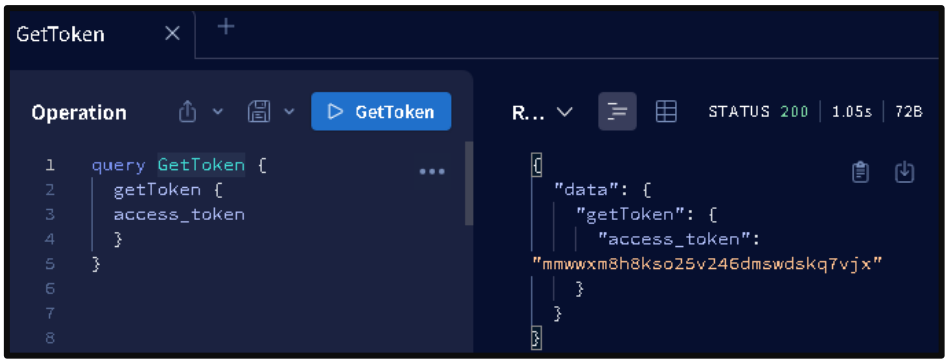
- **URL:** https://id.twitch.tv/oauth2/token.
- **Operación:**

| **POST** |
| --- |
| **Parámetros:**<br>**client_id (String):** ID del cliente registrado de la aplicación creada de Twitch.<br>**Client_secret (String):** El secreto del cliente registrado de la aplicación.<br>**Grant_type (String):** Debe establecerse en *client_credentials.* |

A continuación, en la Figura 18 se muestra un ejemplo del consumo del recurso desde Apollo Client.

**Figura 18**

*Ejemplo de estructura para obtener un token.*




**RECURSO TRANSMISIONES EN VIVO**

Para listar y obtener una lista de transmisiones en vivo, se especifica parámetros no obligatorios para un listado personalizado como *first* o *after*; se llama a la consulta "*getLiveStreams*" , como resultado se obtiene un tipo "*LiveStreams*".

- **URL:** https://api.twitch.tv/helix/streams.
- **Operación:**

| **GET** |
|---|
| **Parámetros:**<br>**first (Int):** Es el número máximo de elementos a devolver por página en la respuesta. Mínimo 1 y máximo 100. Por defecto 20. Opcional.<br>**After (String):** Es el cursor utilizado para obtener la siguiente página de resultados. El objeto de paginación en la respuesta obtiene el valor del cursor. En caso de requerir más del límite de datos.<br><br>**Cabeceras:**<br>**Authorization:** token de acceso. Por ejemplo "*Bearer token*"<br>**Client-Id:** ID del cliente registrado de la aplicación creada de Twitch. |

En la Figura 19 se muestra un ejemplo del consumo del recurso desde Apollo Client agregando la opción de *first* para extraer un límite de datos y se establece solo los datos que se requieren.

**Figura 19**

*Ejemplo de estructura para obtener transmisiones en vivo.*


**RECURSO VIDEOS POR JUEGO**

Para listar y obtener una lista de videos que tiene un juego, se agrega el *game_id* del juego en específico como parámetro obligatorio y parámetros no obligatorios para un listado personalizado como *first* o *after*; se llama a la consulta "*videosByGame*" , como resultado se obtiene un tipo "*VideosByGame*".

- **URL:** https://api.twitch.tv/helix/videos.
- **Operación:**

| **GET** |
|---|
| **Parámetros:**<br>**game_id:** id del juego requerido.<br>**first (Int):** Es el número máximo de elementos a devolver por página en la respuesta. Mínimo 1 y máximo 100. Por defecto 20. Opcional.<br>**After (String):** Es el cursor utilizado para obtener la siguiente página de resultados. El objeto de paginación en la respuesta obtiene el valor del cursor. En caso de requerir más del límite de datos.<br><br>**Cabeceras:**<br>**Authorization:** token de acceso. Por ejemplo "*Bearer token*"<br>**Client-Id:** ID del cliente registrado de la aplicación creada de Twitch. |

En la Figura 20 se muestra un ejemplo del consumo del recurso desde Apollo Client. En este caso se ingresa el id de un juego como parámetro obligatorio y un parámetro opcional como el límite de datos que es 2 y se establece solo los datos que se requieren.

**Figura 20**

*Ejemplo de estructura para obtener los videos que tiene un juego.*


**Retrospectiva del Sprint 1**

**a) Reunión de retrospectiva**

**Fecha:** 03 de enero de 2023.

**Objetivos:** Análisis de errores, mejoras y aciertos.

**b) Resultado de retrospectiva**

**Tabla 25**

*Retrospectiva Sprint 1.*

| | **Retrospectiva** |
|---|---|
| **Aciertos (¿Qué salió bien del Sprint?)** | Facilidad para realizar pruebas de concepto.<br>Obtención del token para el acceso a los recursos. |
| **Errores (¿Qué no salió bien del Sprint?)** | Subestimar el tiempo de desarrollo.<br>Actualizaciones de librerías.<br>Falla en la obtención de datos. |
| **Mejoras (¿Qué mejoras se implementarían?)** | Control de tiempos en base al desarrollo del software.<br>Investigación sobre herramientas y tecnologías a usar para mejor compresión.<br>Creación de métodos para la obtención de datos requeridos. |

### 2.3.2. Sprint 2

**Planificación del Sprint 2**

Se establece las tareas para realizar el desarrollo del envoltorio haciendo uso de las ultimas funcionalidades a usarse en el proyecto.

**a) Reunión de planificación**

**Fecha:** 04 de enero de 2023.

**Objetivo:** Planificación y creación del Sprint Backlog.

**b) Sprint backlog**

En la Tabla 26 se muestra el desarrollo del Sprint 2 Backlog.

**Tabla 26**

*Sprint 2 Backlog.*

| Historia de usuario | Fase de desarrollo | Tarea | Estimación de tiempo |
|---|---|---|---|
| HU-05 | Desarrollo | Crear un type ClipsByUseId con los campos similares a los de la API-REST de Twitch en el archivo schema. | 2 |
| | Desarrollo | Generar la operación en el type Query para su próximo uso en los resolutores llamada getClipsByUserId. | 2 |
| | Desarrollo | Implementar la función para extraer los datos respectivos de los clips que tiene un usuario en los resolutores. | 4 |
| | Pruebas | Pruebas de concepto de la implementación. | 2 |
| HU-06 | Desarrollo | Crear un type ChannelInformation con los campos similares a los de la API-REST de Twitch en el archivo schema. | 2 |
| | Desarrollo | Generar la operación en el type Query para su próximo uso en los resolutores llamada getChannelInformation. | 2 |
| | Desarrollo | Implementar la función para extraer los datos respectivos de la información del canal de un usuario en específico en los resolutores. | 3 |
| | Pruebas | Pruebas de concepto de la implementación. | 2 |
| HU-07 | Desarrollo | Crear un type InformationGameId con los campos similares a los de la API-REST de Twitch en el archivo schema. | 2 |
| | Desarrollo | Generar la operación en el type Query para su próximo uso en los resolutores llamada getInformationGameById. | 2 |

| | | | |
|---|---|---|---|
| | Desarrollo | Implementar la función para extraer los datos respectivos de la información de un juego en específico en los resolutores. | 3 |
| | Pruebas | Pruebas de concepto de la implementación. | 2 |
| OTROS | Desarrollo | Control de errores de la API. | 3 |
| | Pruebas | Pruebas del control de errores. | 2 |
| Eventos | Planificación | Detallar las tareas a realizar en el Sprint actual. | 3 |
| | Revisión | Revisar los resultados del desarrollo del Sprint. | 2 |
| | Retrospectiva | Analizar los resultados del Sprint. | 2 |
| | | **TOTAL HORAS** | **40** |

**Revisión del Sprint 2**

La finalización del segundo Sprint estableció el envoltorio de los últimos niveles a usar en el proyecto.

**a) Reunión de revisión**

**Fecha:** 23 de enero de 2023.

**Resultado:** Revisión del desarrollo del crecimiento del producto.

**b) Incremento del producto entregable**

**RECURSO CLIPS POR USUARIO**

Para obtener una lista de clips que tiene un usuario, se agrega el *broadcaster_id*, el cual es el id de un usuario en específico como parámetro obligatorio y parámetros no obligatorios para un listado personalizado como *first* o *after*; se llama a la consulta "*getClipsByUserId*" , como resultado se obtiene un tipo "*ClipsByUseId*".

- **URL:** https://api.twitch.tv/helix/clips.
- **Operación:**

| **GET** |
|---|
| **Parámetros:**<br>**broadcaster_id:** id del usuario requerido.<br>**first (Int):** Es el número máximo de elementos a devolver por página en la respuesta. Mínimo 1 y máximo 100. Por defecto 20. Opcional.<br>**After (String):** Es el cursor utilizado para obtener la siguiente página de resultados. El objeto de paginación en la respuesta obtiene el valor del cursor. En caso de requerir más del límite de datos. |

**Cabeceras:**

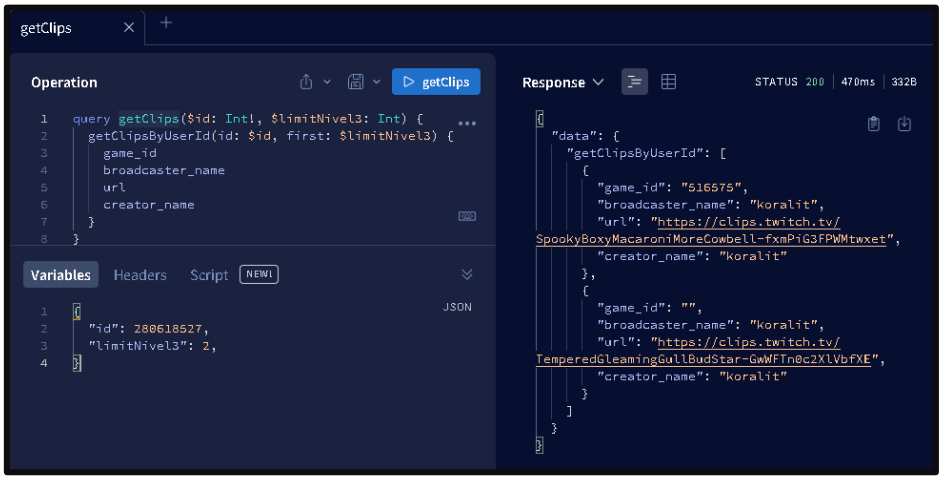
**Authorization:** token de acceso. Por ejemplo “*Bearer token*”

**Client-Id:** ID del cliente registrado de la aplicación creada de Twitch.

En la Figura 21 se muestra un ejemplo del consumo del recurso desde Apollo Client. En este caso se ingresa el id de un usuario como parámetro obligatorio y un parámetro opcional como el límite de datos que es 2 y se establece solo los datos que se requieren.

**Figura 21**

*Ejemplo de estructura para obtener los clips que tiene un usuario.*




**RECURSO INFORMACION DEL CANAL DE UN USUARIO**

Para extraer los datos que tiene el canal de un usuario, se agrega el *broadcaster_id* del usuario en específico como parámetro obligatorio, en este caso no se agrega parámetros opcionales como *first* o *after* ya que es solo un objeto de datos; se llama a la función “*getChannelInformation*” , como resultado se obtiene un tipo “*ChannelInformation*”.

- **URL:** https://api.twitch.tv/helix/channels.
- **Operación:**

| GET |
| --- |
| **Parámetros:**<br>**broadcaster_id:** id del usuario requerido.<br><br>**Cabeceras:**<br>**Authorization:** token de acceso. Por ejemplo "*Bearer token*"<br>**Client-Id:** ID del cliente registrado de la aplicación creada de Twitch. |

En la Figura 23 se muestra un ejemplo del consumo del recurso desde Apollo Client. En este caso se ingresa el id de un usuario como parámetro obligatorio y se establece solo los datos que se requieren.

**Figura 22**

*Ejemplo de estructura para obtener la información del canal de un usuario.*


**RECURSO INFORMACION DE UN JUEGO**

Para extraer los datos que tiene un juego, se agrega el *id* de un juego en específico como parámetro obligatorio, , en este caso tampoco se agrega parámetros opcionales como *first* o *after* ya que es solo un objeto de datos; se llama a la función "*getInformationGameById*" , como resultado se obtiene un tipo "*InformationGameId*".

- **URL:** https://api.twitch.tv/helix/games.
- **Operación:**

| GET |
| --- |
| **Parámetros:**<br>**id:** id del juego requerido.<br><br>**Cabeceras:**<br>**Authorization:** token de acceso. Por ejemplo “*Bearer token*”<br>**Client-Id:** ID del cliente registrado de la aplicación creada de Twitch. |

En la Figura 23 se muestra un ejemplo del consumo del recurso desde Apollo Client. En este caso se ingresa el id de un juego como parámetro obligatorio y se establece solo los datos que se requieren.

**Figura 23**

*Ejemplo de estructura para obtener la información de un juego.*


**Retrospectiva del Sprint 2**

a) **Reunión de retrospectiva**

**Fecha:** 23 de enero de 2023.

**Objetivos:** Análisis de errores, mejoras y aciertos.

b) **Resultado de retrospectiva**

**Tabla 27**

*Retrospectiva Sprint 2.*

| | Retrospectiva |
|---|---|
| **Aciertos (¿Qué salió bien del Sprint?)** | Facilidad para realizar el esquema de cada funcionalidad.<br>Facilidad para hacer las pruebas de cada funcionalidad. |
| **Errores (¿Qué no salió bien del Sprint?)** | Subestimar el tiempo de desarrollo.<br>No obtención de datos.<br>Algunos usuarios no tienen la cantidad de datos requeridos. |
| **Mejoras (¿Qué mejoras se implementarían?)** | Control de tiempos en base al desarrollo del software.<br>Implementar condiciones en caso de que no se obtengan los datos requeridos.<br>Investigación de errores y hacer una depuración del programa. |

### 2.3.3. Sprint 3

**Planificación del Sprint 3**

Se establece las tareas para realizar la selección del número de caso en el programa y gestionar el Nivel 1.

**a) Reunión de planificación**

**Fecha:** 25 de enero de 2023.

**Objetivo:** Planificación y creación del Sprint Backlog.

**b) Sprint backlog**

En la Tabla 28 se muestra el desarrollo del Sprint 3 Backlog.

**Tabla 28**

*Sprint 3 Backlog.*

| Historia de usuario | Fase de desarrollo | Tarea | Estimación de tiempo |
|---|---|---|---|
| HU-08 | Desarrollo | Establecer la cantidad de datos al consumir una funcionalidad. | 3 |

| | | | |
|---|---|---|---|
| | Desarrollo | Implementar un menú donde se escoja la opción de límite de datos para aplicar en cada nivel. | 2 |
| | Análisis | Investigar sobre el uso del cache de Apollo Client para el desarrollo del proyecto. | 2 |
| | Desarrollo | Disponer de un menú donde se escoja una opción de consumo. Cuatro opciones: consumo de la API-REST, API-ERST con cache, envoltorio GraphQL y el envoltorio GraphQL con cache. | 3 |
| | Desarrollo | Mostrar un menú donde se seleccione el nivel a consumir. | 2 |
| | Pruebas | Pruebas de concepto de la implementación. | 2 |
| HU-09 | Desarrollo | Crear un método para el consumo de la funcionalidad de la API-REST LiveStreams. | 3 |
| | Desarrollo | Crear un método para el consumo de la funcionalidad de la API-REST LiveStreams con cache. | 2 |
| | Desarrollo | Crear un método para el consumo de la funcionalidad LiveStreams del envoltorio GraphQL. | 3 |
| | Desarrollo | Crear un método para el consumo de la funcionalidad LiveStreams del envoltorio GraphQL con cache. | 2 |
| | Desarrollo | Mostrar el tiempo de respuesta de los distintos tipos de consumo de LiveStreams. | 2 |
| | Pruebas | Pruebas de concepto de la implementación. | 3 |
| OTROS | Desarrollo | Control de errores de consumo | 2 |
| | Pruebas | Pruebas del control de errores. | 2 |
| Eventos | Planificación | Detallar las tareas a realizar en el Sprint actual. | 3 |
| | Revisión | Revisar los resultados del desarrollo del Sprint. | 2 |
| | Retrospectiva | Analizar los resultados del Sprint. | 2 |
| | | **TOTAL HORAS** | **40** |

**Revisión del Sprint 3**

La finalización del tercer Sprint estableció el desarrollo por parte del cliente de los Menús y el consumo de la API-REST y el envoltorio, haciendo uso del cache de Apollo Client del Nivel1.

**a) Reunión de revisión**

**Fecha:** 14 de febrero de 2023.

**Resultado:** Revisión del desarrollo del crecimiento del producto.

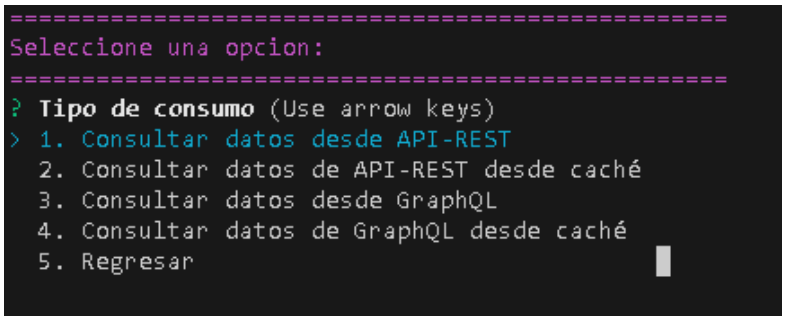
**b) Incremento del producto entregable**

**MENU DE SELECCIÓN DE CONSUMO**

En la Figura 24 se muestra un menú donde aparecen cuatro opciones, las cuales define el tipo de consumo que se quiere hacer con el objetivo de comparar las herramientas.

**Figura 24**

*Ejemplo del menú de selección de consumo*.

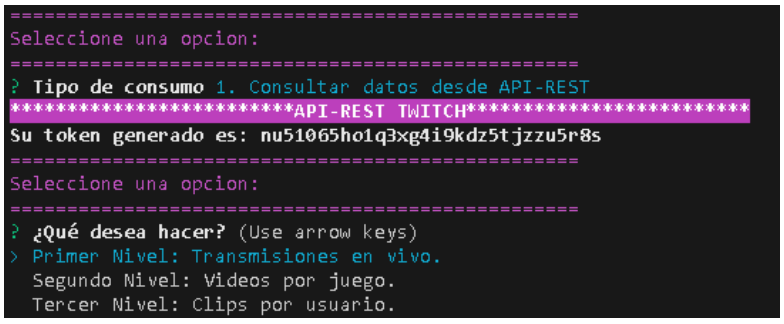



**MENU DE SELECCIÓN DE NIVEL**

En la Figura 25 se observa un menú donde están los niveles de consumo y un título con fondo sobre el tipo de consumo seleccionado.

**Figura 25**

*Ejemplo del menú de selección de funcionalidad a consumir.*

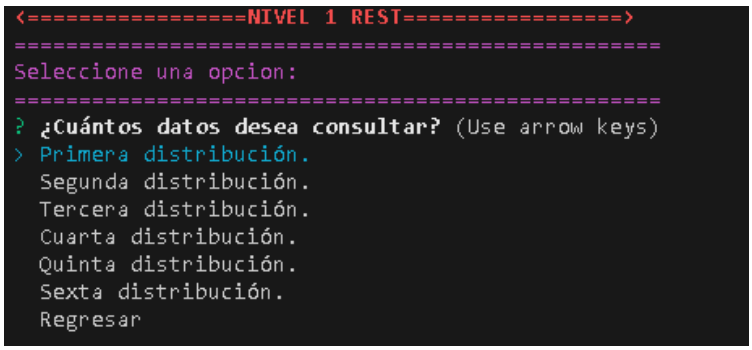



**MENU DE LIMITE DE DATOS PARA CADA NIVEL**

Para poder realizar las pruebas de cada nivel, se realizó un menú donde se muestra 6 opciones de selección como en la Figura 26, cada una contiene la cantidad de datos límite que se definió en la distribución. Además, se muestra el nivel que se va a consumir.

**Figura 26**

*Ejemplo del menú de selección de distribución.*




**GESTION DEL NIVEL 1**

Para el consumo de las funcionalidades del API de Twitch, se emplea la biblioteca *apollo-link-rest.* Con Apollo Link Rest se puede consumir los puntos finales y hacer que Apollo Client administre todos los datos. Es adecuado para hacer una integración completa y luego migrar a una experiencia GraphQL impulsada por back-end (Apollo, 2022). Se crea un modelo que será la consulta de las funcionalidades y tener un código más ordenado. En la figura 28 se muestra un ejemplo del modelo para realizar las consultas.

**Figura 27**

*Ejemplo del modelo para consulta de la funcionalidad Transmisiones en Vivo.*

```
const queryLiveStreams = gql`
  query getLiveStreams($limitNivel1: Int, $cursor: String) {
    liveStreams(first: $limitNivel1, after: $cursor)
      @rest(
        type: "liveStreams"
        path: "streams?first={args.first}{args.after}"
      ) {
      data
      pagination
    }
  }
`;
```

Después de desarrollar el consumo para la API-REST, se utiliza la misma biblioteca para emplear el cache de Apollo Client en las consultas. Se emplea el método de consumo de los modelos de la API-REST para realizar los procedimientos para consumir el envoltorio sin cache y con cache. En el caso de GraphQL, se define la estructura para obtener los datos. Finalmente se muestra el tiempo de respuesta.

**Retrospectiva del Sprint 3**

a) **Reunión de retrospectiva**

**Fecha:** 14 de febrero de 2023.

**Objetivos:** Análisis de errores, mejoras y aciertos.

b) **Resultado de retrospectiva**

**Tabla 29**

*Retrospectiva Sprint 3.*

| | **Retrospectiva** |
|---|---|
| **Aciertos (¿Qué salió bien del Sprint?)** | Desarrollo de los Menús de selección. |
| **Errores (¿Qué no salió bien del Sprint?)** | Poca información sobre las bibliotecas que se usarán en el envoltorio. |
| **Mejoras (¿Qué mejoras se implementarían?)** | Investigación sobre las herramientas que son necesarias para ejecutar el proyecto.<br>Revisión de ejemplos. |

### 2.3.4. Sprint 4

**Planificación del Sprint 4**

Se establece las tareas para realizar la gestión del nivel 2.

a) **Reunión de planificación**

**Fecha:** 15 de febrero de 2023.

**Objetivo:** Planificación y creación del Sprint Backlog.

b) **Sprint backlog**

En la Tabla 30 se muestra el desarrollo del Sprint 4 Backlog.

**Tabla 30**

*Sprint 4 Backlog.*

| **Historia de usuario** | **Fase de desarrollo** | **Tarea** | **Estimación de tiempo** |
|---|---|---|---|
| HU-10 | Desarrollo | Crear un método para el consumo de la funcionalidad de la API-REST VideosByGame. | 4 |

| | | | |
|---|---|---|---|
| | Desarrollo | Crear un método para el consumo de la funcionalidad de la API-REST VideosByGame con cache. | 3 |
| | Desarrollo | Crear un método para el consumo de la funcionalidad VideosByGame del envoltorio GraphQL. | 4 |
| | Desarrollo | Crear un método para el consumo de la funcionalidad VideosByGame del envoltorio GraphQL con cache. | 3 |
| | Desarrollo | Mostrar el tiempo de respuesta de los distintos tipos de consumo de VideosByGame. | 2 |
| | Pruebas | Pruebas de concepto de la implementación. | 3 |
| OTROS | Desarrollo | Control de errores de consumo | 3 |
| | Pruebas | Pruebas del control de errores. | 2 |
| Eventos | Planificación | Detallar las tareas a realizar en el Sprint actual. | 3 |
| | Revisión | Revisar los resultados del desarrollo del Sprint. | 2 |
| | Retrospectiva | Analizar los resultados del Sprint. | 1 |
| | | **TOTAL HORAS** | **30** |

**Revisión del Sprint 4**

La finalización del cuarto Sprint estableció el desarrollo por parte del cliente para el consumo de la API-REST y el envoltorio, haciendo uso del cache de Apollo Client del Nivel 2

**a) Reunión de revisión**

**Fecha:** 02 de marzo de 2023.

**Resultado:** Revisión del desarrollo del crecimiento del producto.

**b) Incremento del producto entregable**

**GESTION DEL NIVEL 2**

Se crea un modelo que será la consulta de las funcionalidades. Luego se hace uso de esa consulta para obtener los datos respectivos a la funcionalidad. En la figura 28 se muestra un ejemplo del modelo para realizar las consultas.

**Figura 28**

*Ejemplo del modelo para consulta de la funcionalidad Videos por Juego.*

```
const queryVideosByGame = gql`
  query getVideosByGame($id: Int, $limitNivel2: Int, $cursor: String) {
    videosByGame(id: $id, first: $limitNivel2, after: $cursor)
    @rest(
        type: "videosByGame"
        path: "videos?game_id={args.id}&first={args.first}{args.after}"
      ) {
      data
      pagination
    }
  }
`;
```

Para este caso, se utilizó el mismo proceso que la gestión de Nivel 1, pero con una variante, donde se extraían datos o el id para los videos por juego, los cuales al usarlos, contenga la misma o mayor cantidad de datos que la que se define en limitNivel2 porque al consumir los VideosByGame con un id en específico, no tiene la cantidad de datos deseada o tiene datos vacíos. Finalmente se muestra el tiempo de respuesta.

**Retrospectiva del Sprint 4**

a) **Reunión de retrospectiva**

**Fecha:** 02 de marzo de 2023.

**Objetivos:** Análisis de errores, mejoras y aciertos.

b) **Resultado de retrospectiva**

**Tabla 31**

*Retrospectiva Sprint 4.*

| | **Retrospectiva** |
|---|---|
| **Aciertos (¿Qué salió bien del Sprint?)** | Desarrollo de los métodos respectivos a la API-REST y el envoltorio GrahpQL del Nivel 2. |
| **Errores (¿Qué no salió bien del Sprint?)** | Errores de falta de datos o datos vacíos.<br>Errores de tiempo de espera del consumo de una API como ETIMEDOUT o ECONNRESET. |
| **Mejoras (¿Qué mejoras se implementarían?)** | Investigación de la documentación de Twitch sobre la API.<br>Investigar sobre el manejo de errores. |

### 2.3.5. Sprint 5

**Planificación del Sprint 5**

Se establece las tareas para realizar la gestión del nivel 3.

**a) Reunión de planificación**

**Fecha:** 06 de marzo de 2023.

**Objetivo:** Planificación y creación del Sprint Backlog.

**b) Sprint backlog**

En la Tabla 32 se muestra el desarrollo del Sprint 5 Backlog.

**Tabla 32**

*Sprint 5 Backlog.*

| Historia de usuario | Fase de desarrollo | Tarea | Estimación de tiempo |
|---|---|---|---|
| HU-11 | Desarrollo | Crear un método para el consumo de la funcionalidad de la API-REST ClipsByUser. | 4 |
| | Desarrollo | Crear un método para el consumo de la funcionalidad de la API-REST ClipsByUser con cache. | 3 |
| | Desarrollo | Crear un método para el consumo de la funcionalidad ClipsByUser del envoltorio GraphQL. | 4 |
| | Desarrollo | Crear un método para el consumo de la funcionalidad ClipsByUser del envoltorio GraphQL con cache. | 3 |
| | Desarrollo | Mostrar el tiempo de respuesta de los distintos tipos de consumo de ClipsByUser. | 2 |
| | Pruebas | Pruebas de concepto de la implementación. | 3 |
| OTROS | Desarrollo | Control de errores de consumo | 3 |
| | Pruebas | Pruebas del control de errores. | 2 |
| Eventos | Planificación | Detallar las tareas a realizar en el Sprint actual. | 3 |
| | Revisión | Revisar los resultados del desarrollo del Sprint. | 2 |
| | Retrospectiva | Analizar los resultados del Sprint. | 1 |
| | | **TOTAL HORAS** | **30** |

**Revisión del Sprint 5**

La finalización del quinto Sprint estableció el desarrollo por parte del cliente para el consumo de la API-REST y el envoltorio, haciendo uso del cache de Apollo Client del Nivel 3.

**a) Reunión de revisión**

**Fecha:** 21 de marzo de 2023.

**Resultado:** Revisión del desarrollo del crecimiento del producto.

**b) Incremento del producto entregable**

**GESTION DEL NIVEL 3**

Primeramente se crea el modelo respectivo y luego se hace uso de esa consulta para obtener los datos. En la Figura 29 se muestra un ejemplo del modelo para realizar las consultas.

**Figura 29**

*Ejemplo del modelo para consulta de la funcionalidad Clips por Usuario.*

```
const queryClipsByUser = gql`
  query getClipsByUser($id: Int!, $limitNivel3: Int, $cursor: String) {
    clipsUser(id: $id, limitNivel3: $limitNivel3, after: $cursor)
      @rest(
        type: "clipsUser"
        path: "clips?broadcaster_id={args.id}&first={args.limitNivel3}{args.after}"
      ) {
      data
      pagination
    }
  }
`;
```

Se lo hizo con el proceso que la gestión de Nivel 2, en este caso la configuración es para obtener los *broadcaster_id* de los usuarios que tengan más o igual cantidad de datos que se indica en el límite de la funcionalidad. Para finalizar, se muestra el tiempo de respuesta.

**Retrospectiva del Sprint 5**

**a) Reunión de retrospectiva**

**Fecha:** 21 de marzo de 2023.

**Objetivos:** Análisis de errores, mejoras y aciertos.

**b) Resultado de retrospectiva**

**Tabla 33**

*Retrospectiva Sprint 5.*

| | **Retrospectiva** |
|---|---|
| **Aciertos (¿Qué salió bien del Sprint?)** | Desarrollo de los métodos del Nivel 3. |
| **Errores (¿Qué no salió bien del Sprint?)** | Errores de falta de datos o datos vacíos.<br>Errores de consumo de la API como ETIMEDOUT o ECONNRESET. |
| **Mejoras (¿Qué mejoras se implementarían?)** | Investigar sobre el manejo de errores. |

## 2.4. Pruebas de aceptación

Al terminar el desarrollo del proyecto, se necesita realizar la verificación a través de pruebas de aceptación. Se muestra en la Tabla 34.

**Tabla 34**

*Pruebas de aceptación.*

| **ID-HU** | **Descripción** | **Función** | **Aceptación** | |
|---|---|---|---|---|
| | | | **SI** | **NO** |
| HU-01 | Definición de estructura del software | Modelo de las herramientas a usar para el desarrollo del proyecto. | X | |
| | | Modelo del consumo de las principales funcionalidades de la API de Twitch. | X | |
| HU-02 | Gestión del recurso del token de aplicación | Obtener credenciales | X | |
| | | Asignar las credenciales como parámetros de la API | X | |
| | | Consumir la funcionalidad del token | X | |
| HU-03 | Gestión del recurso de transmisiones en vivo | Mostrar token generado | X | |
| | | Ingresar límite de datos como parámetro | X | |

| | | | |
|---|---|---|---|
| | | Consumir la funcionalidad | X |
| | | Listar transmisiones en vivo | X |
| | | Ingresar límite de datos como parámetro | X |
| HU-04 | Gestión del recurso de videos que tiene un juego | Ingresar el id de juego como parámetro | X |
| | | Consumir la funcionalidad | X |
| | | Listar los videos que tiene un juego | X |
| | | Ingresar límite de datos como parámetro | X |
| HU-05 | Gestión del recurso de clips que tiene un usuario | Ingresar el id de un usuario como parámetro | X |
| | | Consumir la funcionalidad | X |
| | | Listar los clips de un usuario | X |
| | | Ingresar el id de un usuario como parámetro | X |
| HU-06 | Gestión del recurso de información de un canal | Consumir la funcionalidad | X |
| | | Mostrar la información del canal del usuario | X |
| | | Ingresar el id de un juego como parámetro | X |
| HU-07 | Gestión del recurso de información de un juego | Consumir la funcionalidad | X |
| | | Mostrar la información del juego | X |
| | | Opciones para generar token | X |
| | | Opciones para seleccionar tipo de consumo | X |
| HU-08 | Gestión de Menús de selección | Opciones para seleccionar el nivel de consumo | X |
| | | Opciones para seleccionar el límite de datos para cada nivel (Distribución) | X |
| HU-09 | Gestión del Nivel 1 | Consumir el primer nivel con un límite de datos seleccionado | X |
| | | Mostrar tiempo de consumo | X |

| | | | |
|---|---|---|---|
| HU-10 | Gestión del Nivel 2 | Consumir el segundo nivel con un límite de datos seleccionado | X |
| | | Mostrar tiempo de consumo | X |
| HU-11 | Gestión del Nivel 3 | Consumir el tercer nivel con un límite de datos seleccionado | X |
| | | Mostrar tiempo de consumo | X |

# CAPÍTULO 3

## Validación de resultados

La validación del estudio se hizo mediante un experimento basado en la guía de Wohlin et al., (2012) "*Experimentation in Software engineering*". El experimento consiste en comparar la eficiencia del consumo de datos, tomando en cuenta el tiempo de respuesta. Para este capítulo se plantea la pregunta de investigación ***PI*****:** ¿En qué ambientes se presentan las condiciones ideales para implementar con éxito enfoques que envuelvan REST y GraphQL? Para medir el tiempo se usó la norma ISO/IEC 25023 con respecto a la característica *Eficiencia de rendimiento*, de la calidad de un producto de software.

### 3.1. Entorno del experimento

#### 3.1.1. Objetivo del experimento

Comparar la eficiencia de rendimiento de los servicios de la API-REST y el envoltorio GraphQL de Twitch en un entorno "localhost", con relación a la calidad del producto de software.

#### 3.1.2. Factores y tratamiento

El factor para indagar es la arquitectura del software, específicamente el servicio de las APIs. Los tratamientos que se emplearon fueron:

- Arquitectura REST para el desarrollo de la API.
- Arquitectura GraphQL para el desarrollo de un envoltorio de la API-REST.

#### 3.1.3. Variable

Para la variable independiente se definió a la "Arquitectura del software" que representa el factor de manipulación de los servicios REST y GraphQL. Por otro lado, se determinó como variable dependiente a la "Eficiencia de rendimiento". En la Tabla 35 se muestra de forma más detallada las variables.

**Tabla 35**

*Variables para experimento.*

| | |
|---|---|
| **Variable Independiente** | Arquitectura REST de la API de Twitch para el desarrollo. |
| | Arquitectura GraphQL para el desarrollo de un envoltorio de la API-REST de Twitch. |

| | |
|---|---|
| **Variable Dependiente** | Calidad del producto de software en función a la característica de Eficiencia de Rendimiento. |

La norma ISO/IEC 25023 define las métricas de la eficiencia del rendimiento en la calidad del software, en este caso, se toma en cuenta el tiempo medio de respuesta.

**Tiempo medio de respuesta**

Esta métrica se refiere al tiempo que se demora en completar un proceso asíncrono. La función que se aplica para este es:

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

Donde, $\boldsymbol{A_I}$ es el tiempo de inicio del trabajo i; $\boldsymbol{B_I}$ es el tiempo final del trabajo i; y $\boldsymbol{n}$ es el número de mediciones.

### 3.1.4. Hipótesis

En la presente sección se define la pregunta de investigación que se deriva de la ***PI***.

- ***PI₁:*** ¿Cómo afecta la implementación de enfoques híbridos REST/GraphQL a la eficiencia del rendimiento de las APIs?

Luego de detallar la pregunta de investigación, se generan las hipótesis para el experimento con relación a la ***PI₁***.

- ***H₀:*** *(Hipótesis nula)*: No hay diferencias significativas en la eficiencia del rendimiento, entre la arquitectura REST y el envoltorio GraphQL. En otras palabras, no afecta el desempeño de la aplicación, en este caso el tiempo de respuesta.

- ***H₁:*** *(Hipótesis alternativa 1)*: Las arquitecturas REST son mejores en términos de eficiencia de rendimiento, en comparación con el uso de GraphQL. Es decir, se espera que esta arquitectura proporcione una calidad superior en comparación con el uso de GraphQL.

- ***$H_2$:*** *(Hipótesis alternativa 2)*: Los servicios desarrollados con GraphQL son mucho más eficientes en términos de rendimiento, en comparación con los servicios REST. Esto implica que el uso de GraphQL como envoltorio mostrará un mejor trabajo de un sistema.

### 3.1.5. Diseño

El diseño del experimento tiene como fin disponer los términos necesarios para comparar la eficiencia del API-REST y el envoltorio GraphQL mediante la elaboración de un laboratorio computacional. Para este proceso, se dispone de cuatro trabajos experimentales. La primera que consumirá datos el API-REST, la segunda del API-REST desde caché, la tercera del envoltorio GraphQL y la cuarta del envoltorio GraphQL con caché.

Se determina tres casos de uso de diferentes niveles de complejidad. En cada uno de estos escenarios se realizan tres repeticiones con diferentes cantidades de datos para verificar la eficiencia de las APIs. Se toma en cuenta las métricas establecidas en la sección 3.1.3, que captura el tiempo de respuesta por cada caso de uso. A continuación, se expone el diseño del experimento en la Tabla 36.

**Tabla 36**

*Diseño del experimento.*

| Caso de uso | Repeticiones | REST-REST Caché | GraphQL – GraphQL Caché |
|---|---|---|---|
| Caso de uso 1 | 3 | Registros: 1, 50, 250, 3150, 32800, 100000 | Registros: 1, 50, 250, 3150, 32800, 100000 |
| Caso de uso 2 | 3 | Registros: 1, 36, 256, 3136, 32604 | Registros: 1, 36, 256, 3136, 32604 |
| Caso de uso 3 | 3 | Registros: 1, 27, 175, 2375, 20088 | Registros: 1, 27, 175, 2375, 20088 |

### 3.1.6. Tareas experimentales

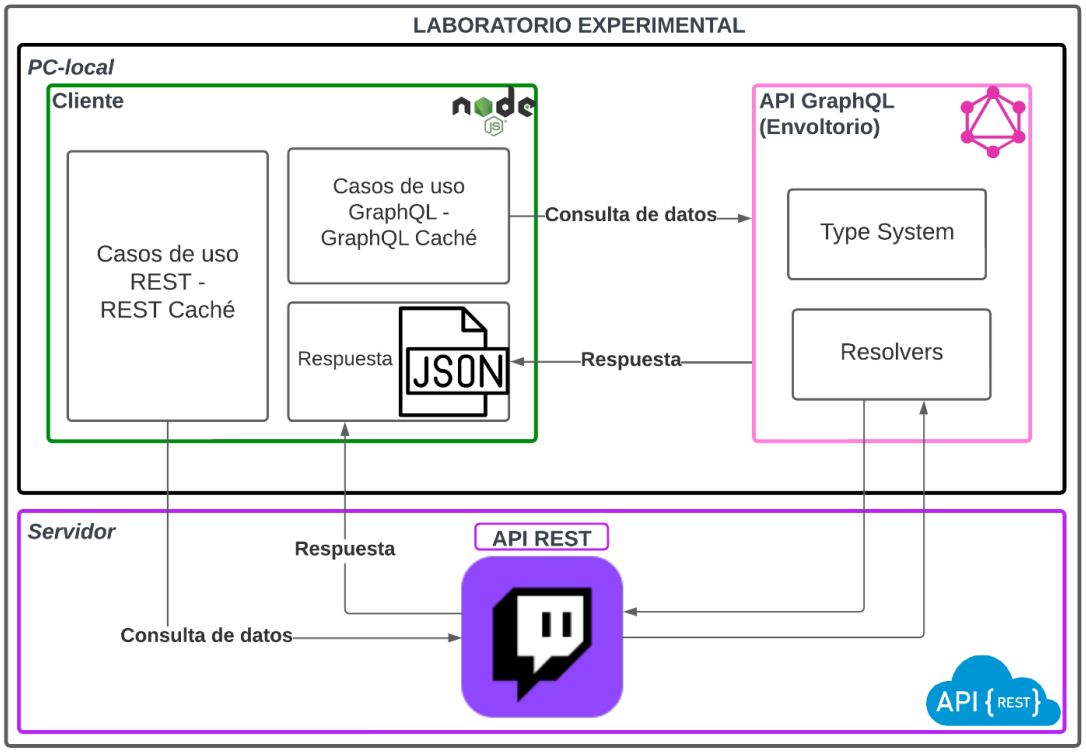
En esta sección, como se muestra en la Figura 30 se elaboró un laboratorio experimental con cuatro tareas experimentales acorde al diseño y tratamientos establecidos para el experimento.

- Tarea experimental 1: Consulta a la API-REST de Twitch.
- Tarea experimental 2: Consulta a la API-REST de Twitch usando caché.

- Tarea experimental 3: Consulta al envoltorio GraphQL.
- Tarea experimental 4: Consulta al envoltorio GraphQL usando caché.

**Figura 30**

*Arquitectura de laboratorio experimental.*




### 3.1.7. Instrumentación

En esta sección, se enumeran los componentes del laboratorio computacional, que incluyen la instrumentación como infraestructura, tecnología y bibliotecas utilizadas.

**Especificaciones de la PC-local**, cuenta con las siguientes características:

- Sistema Operativo: Windows 10 Pro 64-bit
- Procesador: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
- Memoria RAM: 16,0 GB

**Ambiente de desarrollo**, está conformada del uso de las siguientes tecnologías y aplicaciones:

- Entorno de desarrollo integrado IDE: Visual Studio Code v1.79
- Lenguaje de programación: JavaScript
- Entorno de tiempo de ejecución de JavaScript: Node.js v18.15

- Librerías npm para el servidor GraphQL: @apollo/client v3.7.9, apollo-client v2.6.10, apollo-link-rest v0.9, apollo-link-http v1.5.17, apollo-server v3.11.1, graphql v15.8, node-storage v0.0.9
- Librerías npm para el cliente: @apollo/client v3.7.9, apollo-link-rest v0.9, graphql-tag v2.12.6, inquirer v8.0.0, node-cache v5.1.2, node-fetch v2.6.8, node-storage v0.0.9
- Aplicación cliente para el consumo del API-REST: Postman v10.14.2
- Aplicación cliente para el consumo del envoltorio GraphQL: Apollo Client

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

- Microsoft Excel 365
- IBM SPSS v29.0.1

### 3.1.8. Recolección de datos

En la Tabla 36 se observa la estructura de datos del archivo Excel para la recolección de datos, los cuales se obtuvieron al ejecutar el experimento diseñado en la Tabla 37.

**Tabla 37**

*Estructura de recolección de datos.*

| Variable | Descripción |
|---|---|
| Nro. | Número de muestras |
| Arquitectura | Envoltorio GraphQL o REST (Con caché y sin caché) |
| Repeticiones | Número de repeticiones (1 a 3 veces) |
| Caso de uso | Caso de uso ejecutado |
| Nivel | Nivel de complejidad de la consulta |
| Nro. Registros | Cantidad de registros consultados en el caso de uso |
| Tiempo | Tiempo de respuesta en milisegundos de la ejecución de cada caso de uso |

### 3.1.9. Análisis

El análisis del experimento se llevó a cabo utilizando la herramienta IBM SPSS Statistics. Para evaluar la normalización de los datos, se toma en cuenta medidas como la desviación estándar, el coeficiente de variación, el valor mínimo o el valor máximo. Se usará un modelo lineal mixto para probar las hipótesis.

## 3.2. Ejecución del experimento

La ejecución del experimento se llevó a cabo en julio de 2023, donde se usó el entorno experimental que se presentó en la sección 3.1.

### 3.2.1. Muestra

Como muestra se usó los casos de uso la gestión de niveles desarrollado anteriormente, con el objetivo de extraer los datos necesarios para llevar a cabo el experimento, el cual se lo hizo con relación al diseño elaborado en la sección 3.1.5.

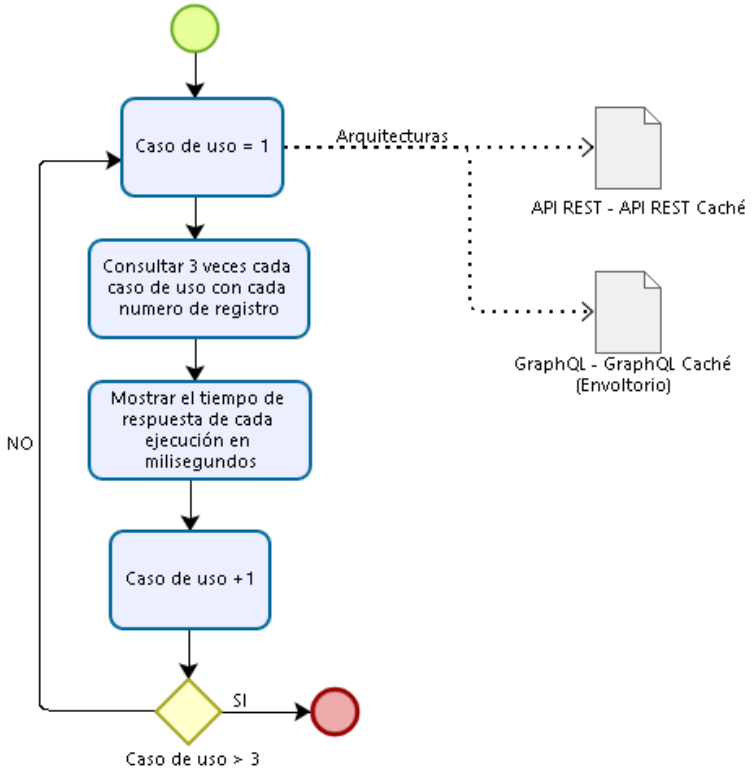
### 3.2.2. Preparación

Primero, se comprobó la correcta conexión a internet. Se verificó la autenticación y conexión a la API de Twitch, también el correcto funcionamiento de las tareas experimentales expuestas en la sección 3.1.6 a través de la ejecución de varias pruebas del envoltorio GraphQL y el servicio API-REST de Twitch. Luego de confirmar y probar el funcionamiento del entorno experimental, se procedió a llevar a cabo la ejecución del experimento.

La ejecución del experimento se lo hizo de modo iterativo por cada caso de uso, en otras palabras, se corrió la parte del cliente que procede a consumir un caso de uso 3 veces por cada número de registro y arquitectura. De esta manera, la cantidad de ejecuciones por cada caso son 36. Obteniendo un total de 204 registros. En la Figura 31 se observa el proceso de ejecución del experimento.

**Figura 31**

*Proceso de ejecución del experimento.*




### 3.2.3. Recolección de datos

Una vez concluido el experimento de acuerdo con la representación mostrada en la Figura 31, se procedió a recopilar los resultados relacionados con los tiempos de respuesta, los cuales se registraron en un archivo de Microsoft Excel 365 para su posterior análisis. La recopilación de datos se llevó a cabo utilizando el mismo enfoque en los diversos casos de uso analizados. Posteriormente a cada ejecución del programa cliente, se obtuvieron los tiempos de respuesta correspondientes, los cuales se imprimieron en la consola y se recopilaron en el mencionado libro de Excel. En la Tabla 38 se presenta una representación de los datos recopilados específicamente para el caso de uso 1 donde se calcula el promedio del tiempo por el número de repeticiones de ejecución para mostrar una tabla flexible.

**Tabla 38**

*Datos Caso de uso 01.*

| Caso de uso (CU) | Nivel | Repeticiones | Nro. Registros | Tiempo (ms) - Arquitectura | | | |
|---|---|---|---|---|---|---|---|
| | | | | REST | REST Caché | GraphQL | GraphQL Caché |
| CU-01 | 1 | 3 | 1 | 648,27 | 0,68 | 807,05 | 0,88 |
| | 1 | 3 | 50 | 1.087,70 | 2,12 | 395,43 | 0,54 |
| | 1 | 3 | 250 | 3.707,48 | 3,20 | 1.941,01 | 0,83 |
| | 1 | 3 | 3150 | 59.974,46 | 22,28 | 18.150,18 | 1,01 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 3 | 32800 | 687.307,43 | 254,23 | 149.446,77 | 1,06 |
| | 1 | 3 | 100000 | 843.265,33 | 1.800,58 | 818.106,46 | 12.368,22 |

Para el Caso de uso 02 y 03, los datos se muestran en la Tabla 39 y 40 respectivamente. Así mismo, se muestra el promedio del tiempo por las tres repeticiones por cada número de registro.

**Tabla 39**

*Datos Caso de uso 02.*

| Caso de uso (CU) | Nivel | Repeticiones | Nro. Registros | Tiempo (ms) - Arquitectura | | | |
|---|---|---|---|---|---|---|---|
| | | | | REST | REST Caché | GraphQL | GraphQL Caché |
| | 2 | 3 | 1 | 953,18 | 1,42 | 933,16 | 0,80 |
| | 2 | 3 | 36 | 3.491,69 | 2,42 | 946,09 | 0,88 |
| CU-02 | 2 | 3 | 256 | 7.739,53 | 5,45 | 962,71 | 0,97 |
| | 2 | 3 | 3136 | 51.041,00 | 32,18 | 2.540,74 | 0,38 |
| | 2 | 3 | 32604 | 322.373,64 | 91,87 | 9.923,69 | 0,44 |

**Tabla 40**

*Datos Caso de uso 03.*

| Caso de uso (CU) | Nivel | Repeticiones | Nro. Registros | Tiempo (ms) - Arquitectura | | | |
|---|---|---|---|---|---|---|---|
| | | | | REST | REST Caché | GraphQL | GraphQL Caché |
| | 3 | 3 | 1 | 1.244,71 | 2,75 | 998,06 | 0,71 |
| | 3 | 3 | 27 | 5.747,24 | 5,16 | 1.353,87 | 1,03 |
| CU-03 | 3 | 3 | 175 | 20.312,01 | 15,48 | 1.420,63 | 0,50 |
| | 3 | 3 | 2375 | 118.509,49 | 57,72 | 3.149,60 | 0,39 |
| | 3 | 3 | 20088 | 614.627,10 | 116,13 | 7.910,59 | 1,16 |

## 3.3. Resultados

### 3.3.1. Análisis de resultados

En este apartado, se examina el impacto de desarrollo de las arquitecturas REST y GraphQL en la calidad del software. Además, se llevó a cabo un análisis estadístico para evaluar la eficiencia del tiempo de respuesta de dichas arquitecturas. Este proceso está basado en los resultados obtenidos durante la ejecución del experimento detallado en la sección 3.2.

### 3.3.2. Análisis de eficiencia

Seguidamente se hace un análisis de la eficiencia del rendimiento por cada uno de los casos de uso con las diferentes arquitecturas. Para este análisis el tiempo que se obtenía en milisegundos, se lo transformó a segundos para identificar de mejor manera la diferencia de la eficiencia.

**Caso de uso 1**

Para este caso, el cual es la obtención de transmisiones en vivo (Nivel 1), se realizó el cálculo del promedio del tiempo de respuesta de cada arquitectura. Se obtuvo lo siguientes resultados: el tiempo promedio que presenta la Arquitectura GraphQL fue 164,80 segundos y en REST fue 265,99 segundos. Por otro lado, el resultado de la Arquitectura GraphQL usando caché fue 2,06 segundos y 0,34 segundos en REST con caché. En la Tabla 41 se muestra el porcentaje de eficiencia.

**Tabla 41**

*Porcentaje de eficiencia del Caso de uso 01.*

| **Arquitectura** | **GraphQL** | **REST** | **GraphQL Caché** | **REST Caché** |
|---|---|---|---|---|
| **Tiempo promedio** | 164,808 | 265,998 | 2,062 | 0,347 |
| **Eficiencia** | **38%** | | **83%** | |

Así mismo tanto en la Figura 32 como en la Figura 33 se grafica los promedios de cada arquitectura para una mejor interpretación y los datos estadísticos.

**Figura 32**

*Promedios del tiempo de arquitecturas sin caché del Caso de uso 01.*

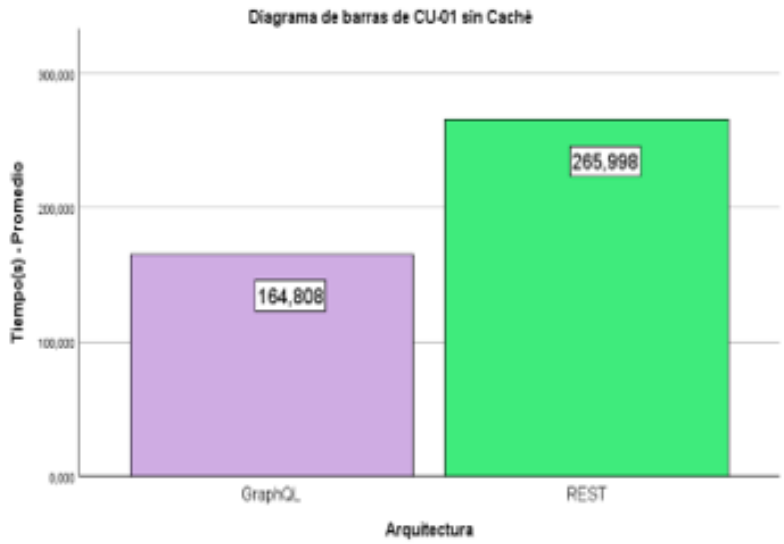
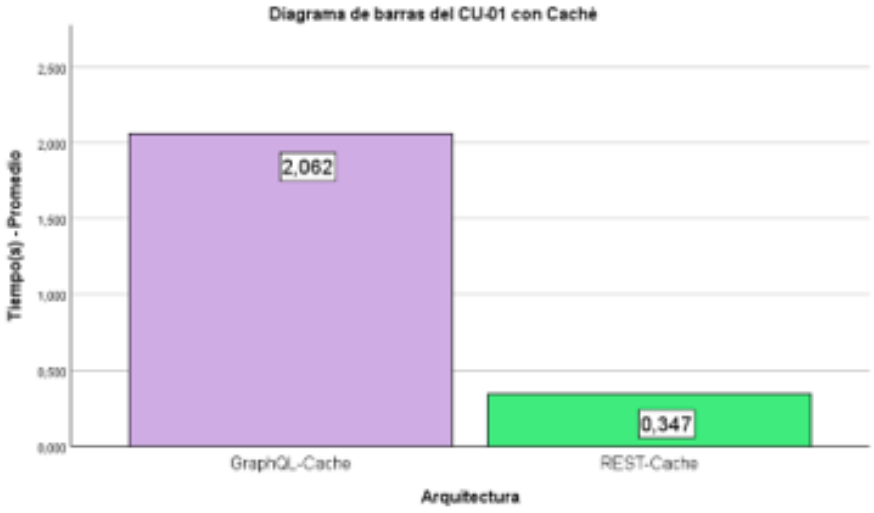



**Figura 33**

*Promedios del tiempo de arquitecturas con caché del Caso de uso 01.*




Se puede observar que sin el uso del caché, el tiempo promedio de GraphQL es un 38% más eficiente que el de REST, pero al usar caché es lo contrario, el tiempo de REST es mucho menor. Por esta razón se determina que con caché, REST 83% más eficiente que GraphQL.

**Caso de uso 2**

En el caso de uso 02, se obtienen los videos que tiene un juego (Nivel 2). Al realizar el cálculo del promedio del tiempo se obtuvo los siguientes resultados: en las arquitecturas sin caché, GraphQL es igual a 3,061 segundos y REST 77,120 segundos y usando caché, el resultado de GraphQL es igual a 0,000 segundos y 0,027 segundos para REST. El porcentaje de eficiencia se lo muestra en la Tabla 42.

**Tabla 42**

*Porcentaje de eficiencia del Caso de uso 02.*

| Arquitectura | GraphQL | REST | GraphQL Caché | REST Caché |
|---|---|---|---|---|
| **Tiempo promedio** | 3,061 | 77,120 | 0,000 | 0,027 |
| **Eficiencia** | **96%** | | **97%** | |

De la misma manera, se realiza un diagrama de barras para mostrar los promedios de las diferentes arquitecturas como se ve en la Figura 34 y Figura 35.

**Figura 34**

*Promedios del tiempo de arquitecturas sin caché del Caso de uso 02.*

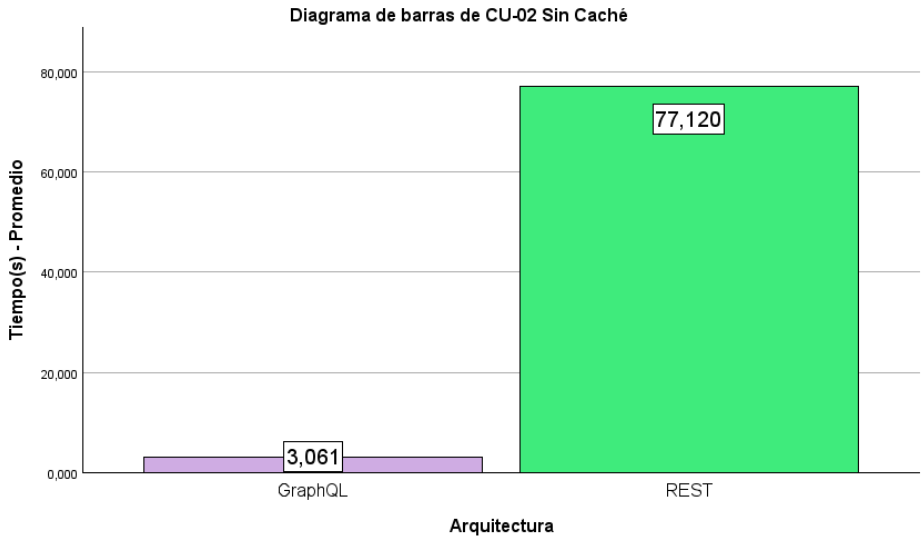
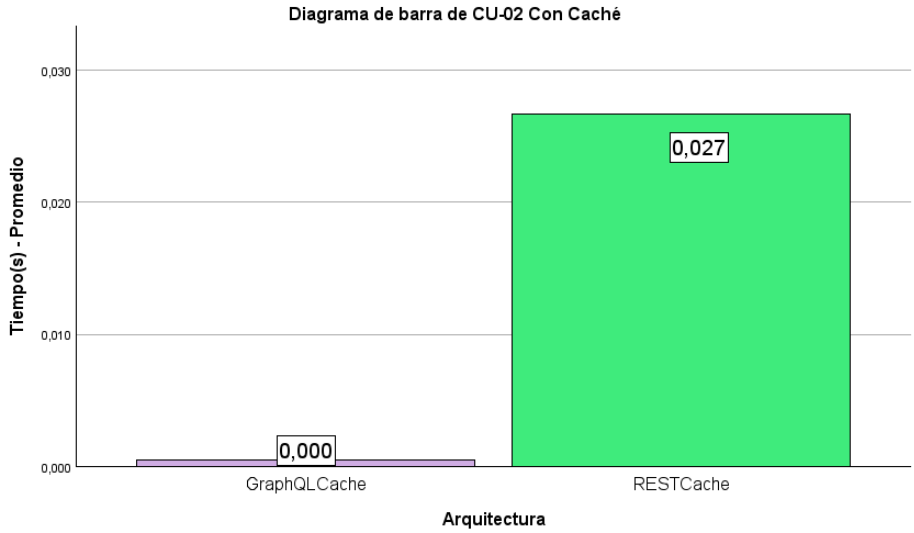



**Figura 35**

P*romedios del tiempo de arquitecturas con caché del Caso de uso 02.*




Al obtener estos resultados en los diagramas, se establece GraphQL es mucho más eficiente que REST tanto al usar caché como al no usarlo por la razón de que el porcentaje de eficiencia supera el 95% en los dos casos.

**Caso de uso 3**

Para este caso, se obtienen los clips que tiene cada usuario (Nivel 3). Los resultados del promedio del tiempo de las arquitecturas sin caché fueron los siguientes: GraphQL es igual a 2,967 segundos y REST 152,088 segundos y el resultado al usar cache es 0,001 segundos en GraphQL y 0,039 segundos para REST. En la Tabla 43 se muestra el porcentaje de eficiencia.

**Tabla 43**

*Porcentaje de eficiencia del Caso de uso 03.*

| **Arquitectura** | **GraphQL** | **REST** | **GraphQL Caché** | **REST Caché** |
|---|---|---|---|---|
| **Tiempo promedio** | 2,967 | 152,088 | 0,001 | 0,039 |
| **Eficiencia** | **98%** | | **98%** | |

Así mismo, se muestran los promedios de las diferentes arquitecturas en la Figura 36 y Figura 37.

**Figura 36**

*Promedios del tiempo de arquitecturas sin caché del Caso de uso 03.*

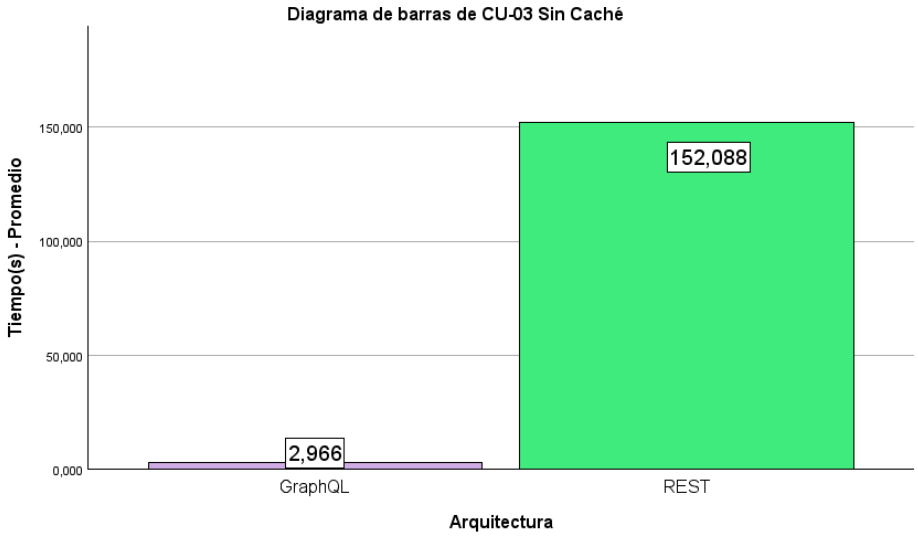
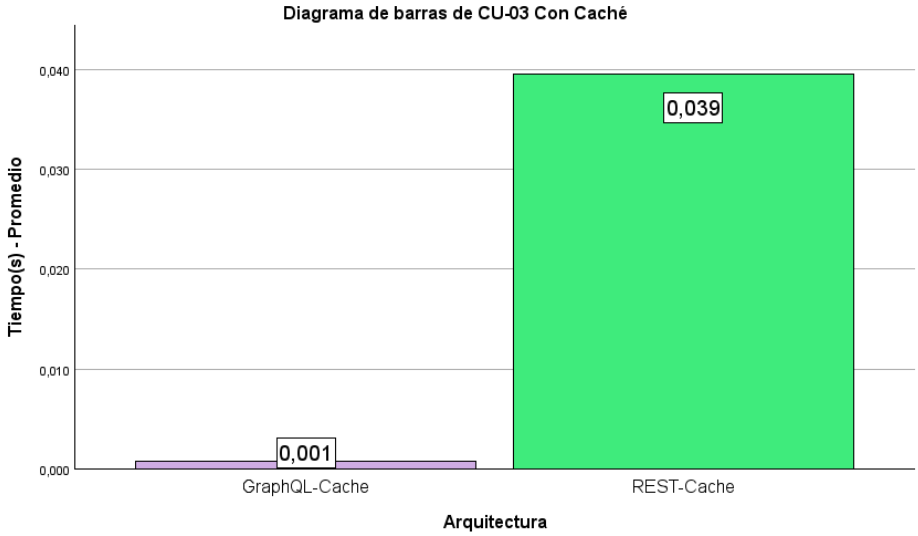



**Figura 37**

*Promedios del tiempo de arquitecturas con caché del Caso de uso 03.*




**Análisis agrupado**

En la Figura 38 y Figura 39 se expone el comportamiento de los tres casos de uso con las arquitecturas sin caché y con caché. En la primera figura se puede ver que el tiempo de respuesta de GraphQL sin el uso del caché es menor en todos los casos de uso. En la siguiente figura, donde se usa caché, el resultado es que el tiempo de GraphQL es igual de menor en la mayoría de los casos, a excepción del caso de uso 01,donde el tiempo es mayor.

**Figura 38**

*Valor promedio de eficiencia entre GraphQL y REST sin caché.*

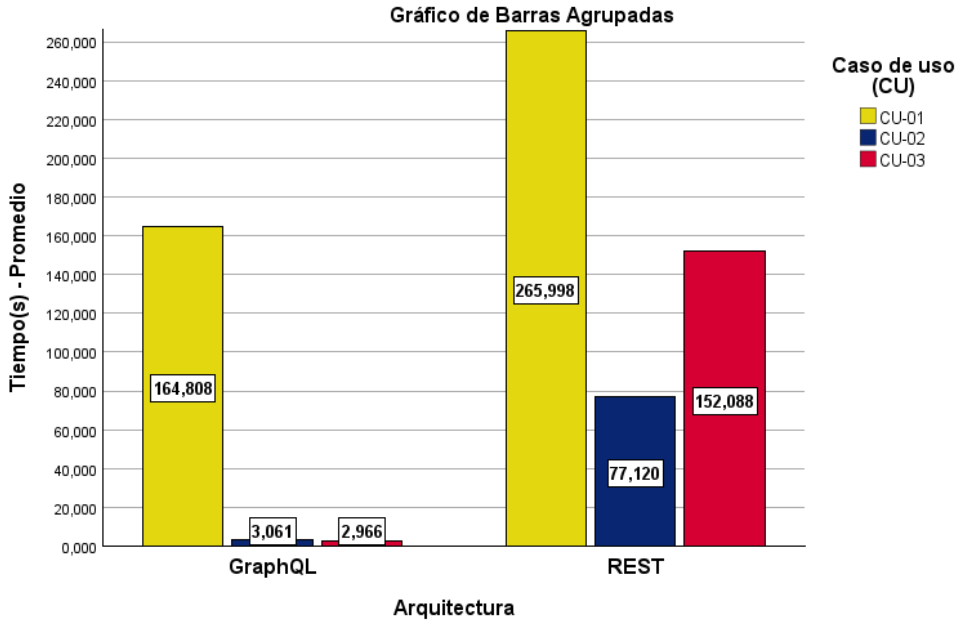
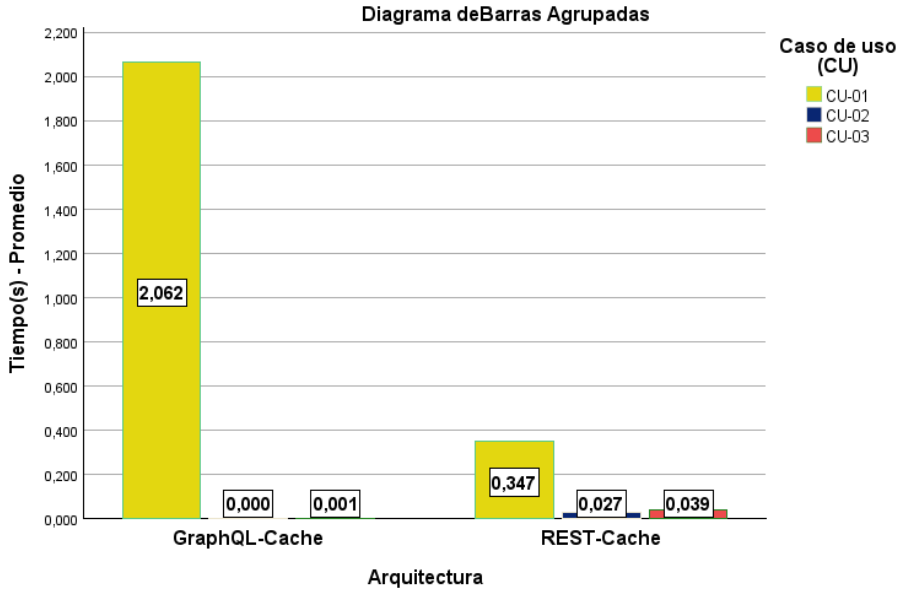



**Figura 39**

*Valor promedio de eficiencia entre GraphQL y REST con caché.*




Posteriormente, en la Figura 40 se presentan los resultados estadísticos globales por cada arquitectura, donde la media de GraphQL fue igual a 63,69. Para el caso de GraphQL con caché la media fue 0,77. Para REST sin caché la media fue igual a 171,38. Finalmente en REST con caché los resultados de la media y error estándar fueron 0,15 y 0,06 respectivamente; el intervalo de confianza fue 0,02 y 0,28.

**Figura 40**

*Resultado estadísticos descriptivos entre las arquitecturas.*

**Descriptivos**

| | Arquitectura | | | Estadístico | Error estándar |
|---|---|---|---|---|---|
| Tiempo(s) | GraphQL | Media | | 63,68656 | 30,106972 |
| | | 95% de intervalo de confianza para la media | Límite inferior | 3,11915 | |
| | | | Límite superior | 124,25398 | |
| | | Media recortada al 5% | | 20,84813 | |
| | | Mediana | | 1,76050 | |
| | | Varianza | | 43508,629 | |
| | | Desv. estándar | | 208,587221 | |
| | | Mínimo | | ,297 | |
| | | Máximo | | 1046,770 | |
| | | Rango | | 1046,473 | |
| | | Rango intercuartil | | 7,861 | |
| | | Asimetría | | 3,987 | ,343 |
| | | Curtosis | | 15,727 | ,674 |
| | GraphQL Caché | Media | | ,77365 | ,437751 |
| | | 95% de intervalo de confianza para la media | Límite inferior | -,10700 | |
| | | | Límite superior | 1,65429 | |
| | | Media recortada al 5% | | ,15644 | |
| | | Mediana | | ,00100 | |
| | | Varianza | | 9,198 | |
| | | Desv. estándar | | 3,032829 | |
| | | Mínimo | | ,000 | |
| | | Máximo | | 13,202 | |
| | | Rango | | 13,202 | |
| | | Rango intercuartil | | ,001 | |
| | | Asimetría | | 3,763 | ,343 |
| | | Curtosis | | 12,775 | ,674 |
| | REST | Media | | 171,37683 | 40,665576 |
| | | 95% de intervalo de confianza para la media | Límite inferior | 89,56825 | |
| | | | Límite superior | 253,18542 | |
| | | Media recortada al 5% | | 141,53912 | |
| | | Mediana | | 13,53750 | |
| | | Varianza | | 79377,077 | |
| | | Desv. estándar | | 281,739378 | |
| | | Mínimo | | ,544 | |
| | | Máximo | | 901,940 | |
| | | Rango | | 901,396 | |
| | | Rango intercuartil | | 262,458 | |
| | | Asimetría | | 1,579 | ,343 |
| | | Curtosis | | 1,048 | ,674 |
| | REST Caché | Media | | ,15083 | ,064502 |
| | | 95% de intervalo de confianza para la media | Límite inferior | ,02107 | |
| | | | Límite superior | ,28059 | |
| | | Media recortada al 5% | | ,06280 | |
| | | Mediana | | ,00950 | |
| | | Varianza | | ,200 | |
| | | Desv. estándar | | ,446880 | |
| | | Mínimo | | ,001 | |
| | | Máximo | | 2,335 | |
| | | Rango | | 2,334 | |
| | | Rango intercuartil | | ,064 | |
| | | Asimetría | | 3,897 | ,343 |
| | | Curtosis | | 15,108 | ,674 |

Con los resultados presentes se determinó que el tiempo medio de respuesta de GraphQL es más eficiencia que REST sin el uso de caché. Al usar caché, REST resulta tener un mejor resultado que GraphQL.

### 3.3.3. Análisis de impactos

Después de realizar el análisis estadístico de eficiencia, se obtuvieron los promedios globales de tiempo de respuesta para las arquitecturas con y sin caché. Para la arquitectura REST con caché, el promedio global de tiempo de respuesta fue de 0.151, mientras que para GraphQL con caché fue de 0.774. Esto indica que al utilizar caché, REST es un 81% más eficiente que GraphQL.

En cuanto a las arquitecturas sin caché, el promedio global de tiempo de respuesta fue de 171.177 para REST y 63.387 para GraphQL. En este caso, se determinó que GraphQL es un 63% más eficiente que REST.

Los resultados de los promedios globales y la eficiencia de cada arquitectura se resumen en la Tabla 44.

**Tabla 44**

*Eficiencia entre las arquitecturas.*

| **Arquitectura** | **GraphQL** | **REST** | **GraphQL Caché** | **REST Caché** |
|---|---|---|---|---|
| **Tiempo promedio global** | 63,687 | 171,377 | 0,774 | 0,151 |
| **Eficiencia** | **63%** | | **81%** | |

Finalmente, los resultados se validaron a través de estadística descriptiva. Así, determinando la base para la aceptación de una de las hipótesis realizadas en el experimento. En este sentido se aprueba la Hipótesis Alternativa 2 (***$H_2$***), donde se menciona que los servicios desarrollados con GraphQL son mucho más eficientes en términos de rendimiento, en comparación con los servicios REST. Eso sucede en la mayoría de los casos con cada arquitectura. Dado que en el Caso de uso 01, solo en la arquitectura REST con caché el tiempo es menor, resulta ser más eficiente.

## Conclusiones

Se ha logrado establecer un marco teórico sólido a través de una profunda revisión de literatura. Este marco teórico, brinda los conceptos necesarios para el desarrollo del trabajo de investigación y para la comprensión del contexto de las teorías que tienen relación con el tema de estudio. También, es fundamental para dirigir el proceso de desarrollo del envoltorio de la API-REST de Twitch y la evaluación de la eficiencia en comparación con la arquitectura REST.

En base a los conceptos planteados se desarrolló con éxito el envoltorio GraphQL de la API-REST de Twitch que demuestra la viabilidad de envolver los servicios REST. Este enfoque permitió mejorar el rendimiento del servicio. Se usaron las tecnologías descritas en la planificación como Node.js para la elaboración del servidor y para las pruebas de concepto (Cliente); Apollo Client para consumir y manejar datos de proyectos GraphQL tanto con caché como sin caché.

Para evaluar la eficiencia y el rendimiento de GraphQL frente a REST se implementó un laboratorio experimental que se describe en el capítulo 3 de este trabajo, en donde se hizo una comparación de la eficiencia de rendimiento de la arquitectura GraphQL y REST, tomando en cuenta el tiempo de respuesta, ingresando un límite de cantidad de datos por consulta. Para lo cual, se respondió la pregunta de investigación ***PI*****:** ¿En qué ambientes se presentan las condiciones ideales para implementar con éxito enfoques que envuelvan REST y GraphQL? Con el objetivo de contestar la pregunta, se efectuó tres casos en todas las arquitecturas, donde se consumió tres veces por cada número de registro y se midió la eficiencia del rendimiento mediante la métrica "*tiempo medio de respuesta*" de la ISO/IEC 25023.

Finalmente, luego de llevar a cabo la ejecución del experimento y realizar el análisis de la eficiencia, se confirmó que el tiempo de respuesta de GraphQL es menor en la mayoría de los casos, a excepción del Caso de uso 01 usando caché. Por esta razón, se concluye que la arquitectura GraphQL ha demostrado ser mucho más eficiente en el desarrollo de servicios en comparación con REST. Sin embargo, es importante destacar la excepción donde REST supera ligeramente al rendimiento de GraphQL, lo que indica que el uso del caché en esta arquitectura puede ofrecer ventajas de eficiencia en escenarios específicos.

## Recomendaciones

Realiza una investigación profunda de la información relacionada con tu tema de investigación. Buscando de fuentes confiables como informes de casos relevantes, artículos científicos o libros con respecto al tema investigado. Esto ayudará a tener un enfoque actualizado y amplio sobre los datos necesarios para realizar el trabajo eficiente.

Si el objetivo es usar APIs o un servicio de alguna aplicación externa, se recomienda leer y revisar profundamente la documentación como las formas de autenticación, los recursos y funciones disponibles, límite de solicitudes, límite de datos o paginación. En caso de no tener mucho conocimiento sobre la API, existe la posibilidad de que se den errores a la hora de realizar el envoltorio o cualquier proyecto que requiera de este servicio.

Aunque el lenguaje de consultas GraphQL se creó en un ambiente de JavaScript, se recomienda usar diferentes herramientas y una diferente API-REST con el objetivo de realizar un trabajo de investigación y un estudio comparativo. Además, tomar en cuenta otros aspectos como el tamaño.

Evaluar cuidadosamente el uso del caché en proyectos que usen REST. A pesar de que GraphQL resultó ser mejor en la mayoría de los casos realizados, se mostró que REST superó ligeramente en uno de ellos. Puede ser necesario evaluar las necesidades del proyecto y comprobar si la aplicación del caché es necesario y puede mejorar el rendimiento en casos específicos.

Tener en cuenta las diferentes guías o metodologías para realizar experimentos tecnológicos y aplicarlos al proyecto, según las necesidades y objetivos que se requieran. Así mismo, estudiar la rama de estadística profundamente para poder validar los resultados de la investigación e interpretarlos de mejor manera.

## Bibliografía


Alfredo Guillen-Drija, C., & Quintero, R. (2018). *GraphQL vs REST: una comparación desde la perspectiva de eficiencia de desempeño*. https://doi.org/10.13140/RG.2.2.25221.19680

Amat, A. P. (2021). *Análisis de la influencia de Twitter en el interés y consumo de videojuegos en Twitch*.

Angulo Vásquez, J. L. (2020). *Evaluación De La Eficiencia De Las Tecnologías Graphql Y Rest En La Implementación De Servicios Web Consumidos Por Aplicaciones Android*. 141.

Antolín-Prieto, Rebeca, Reyes-Menendez, Ana, & Ruiz-Lacaci, N. (2021). *Explorando los factores que afectan al comportamiento de los consumidores en plataformas de live streaming*. https://doi.org/10.48082/espacios-a21v42n14p03

Apollo. (2022). *Introduction to Apollo Client - Apollo GraphQL Docs*. https://www.apollographql.com/docs/react/

Arias, J., & Durango, C. (2018). *Propuesta de un método para desarrollar Sistemas de Información Geográfica a partir de la metodología de desarrollo ágil - SCRUM.* https://ojs.tdea.edu.co/index.php/cuadernoactiva/article/view/490/661

Aziz, M. N., Sapta, I. M., & Rochimah, S. (2018). Security Characteristic Evaluation Based on ISO/IEC 25023 Quality Model, Case Study: Laboratory Management Information System. *2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar, EECCIS 2018*, 332–336. https://doi.org/10.1109/EECCIS.2018.8692982

Baham, C. (2019). Teaching tip: Implementing scrum wholesale in the classroom. *Journal of Information Systems Education*, *30*(3), 141–159.

Banks, A., & Porcello, E. (2018). *Learning GraphQL*. O'Reilly Media.

Benavidez, J., & García, J. (2019). *Aquitectura REST para la plataforma UAO-IoT*. 1–9. https://doi.org/.1037//0033-2909.I26.1.78

Brito, G., Mombach, T., & Valente, M. T. (2019). Migrating to GraphQL: A Practical Assessment. *SANER 2019 - Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering*, 140–150. https://doi.org/10.1109/SANER.2019.8667986

Butt, S. A., Khalid, A., Ercan, T., Ariza-Colpas, P. P., Melisa, A. C., Piñeres-Espitia, G., De-La-Hoz-Franco, E., Melo, M. A. P., & Ortega, R. M. (2022). A software-based cost estimation technique in scrum using a developer's expertise. *Advances in Engineering Software*, *171*. https://doi.org/10.1016/J.ADVENGSOFT.2022.103159

Caillamara Encalada, D. A., & Lalangui Campoverde, J. L. (2023). *Sistema de desvinculación laboral y permisos del departamento de Talento Humano de la Universidad Nacional Chimborazo aplicando tecnología .Net Core*. http://dspace.unach.edu.ec/handle/51000/10627

Chaves-Fraga, D., Priyatna, F., Alobaid, A., & Corcho, O. (2020a). Exploiting Declarative Mapping Rules for Generating GraphQL Servers with Morph-GraphQL. *International Journal of Software Engineering and Knowledge Engineering*, *30*(6), 785–803. https://doi.org/10.1142/S0218194020400070

Chaves-Fraga, D., Priyatna, F., Alobaid, A., & Corcho, O. (2020b). Exploiting Declarative Mapping Rules for Generating GraphQL Servers with Morph-GraphQL. *International Journal of Software Engineering and Knowledge Engineering*, *30*(6), 785–803. https://doi.org/10.1142/S0218194020400070

Dallas Teste. (2017). *La nueva API de Twitch | Blog de contracción*. https://blog.twitch.tv/en/2017/08/31/the-new-twitch-api-be3fb2b078e6/

David, S. R., Angel, V., Miguel, B., Natalia, C., & Reinaldo, C. (2010). *Evaluación de la calidad de la Información extraída por wrappers, de un sitio web*. 185–188.

Delgadillo Hinojosa, R. G. (2019). *COMO DESARROLLAR APLICACIONES WEB CON API REST Y GRAPHQL*. http://ddigital.umss.edu.bo:8080/jspui/handle/123456789/14423

Eddie. (2020). *Experiencing connection failures/timeouts - API - Twitch Developer Forums*. https://discuss.dev.twitch.tv/t/experiencing-connection-failures-timeouts/28004

Eguíluz Pérez, J. (2019). *Introducción a JavaScript*.

Engineering, C. S. (2020). *Master in Web Engineering Master ' s final thesis*. *July*.

Fang, Y., Huang, C., Zeng, M., Zhao, Z., & Huang, C. (2022). JStrong: Malicious JavaScript detection based on code semantic representation and graph neural network. *Computers & Security*, *118*, 102715. https://doi.org/10.1016/J.COSE.2022.102715

Fernando, L. (2019). *JavaScript - Aprende a programar en el lenguaje de la web*. https://books.google.com.ec/books?hl=es&lr=&id=SqikDwAAQBAJ&oi=fnd&pg=PA4&dq=lenguaje+de+programación+javascript&ots=pzahSW1jAE&sig=L5euRdrf60AeMMD3Z9ukMEXz8x0&redir_esc=y#v=onepage&q=lenguaje de programación javascript&f=false

Flores, Á. (2016). *Desarrollo de un entorno virtual colaborativo aplicado a la enseñanza del diseño web, en la espoch extensión morona santiago para mejorar el aprendizaje significativo*.

Foundation, G. (2022). *Introduction to GraphQL | GraphQL*. https://graphql.org/learn/

García, D., & Hidalgo, J. (2021). *Análisis comparativo de Jamstack vs NODE.JS en el desarrollo de páginas y aplicaciones web*. 7–8.

Ghebremicael, E. S. (2017). *Transformation of REST API to GraphQL for OpenTOSCA*.

Glasbergen, B., Abebe, M., Daudjee, K., Foggo, S., & Pacaci, A. (2018). Apollo: Learning query correlations for predictive caching in geo-distributed systems. *Advances in Database Technology - EDBT*, *2018-March*, 253–264. https://doi.org/10.5441/002/edbt.2018.23

Gómez, D. (2018). *Desarrollo del sistema de requisiciones para la empresa hidroelectrica abanico s.a. aplicando el entorno de programacion node.js*.

Gómez, O. S., Ucán, J. P., & Gómez, G. E. (2018). Aplicación del proceso de experimentación a la Ingeniería de Software. *Abstraction & Application*, *8*, 26–37.

Grado, T. F. I. N. D. E., Almeria, U. D. E., Gil, M. T., De, I., & Graphql, A. P. I. (2021). *Implementación de una API GraphQL Director / es :*

Guanabara, E., Ltda, K., Guanabara, E., & Ltda, K. (2018). *Aprendizaje npm*.

Guerrero, G., Guevara, A., Quiña-Mera, J. A., Guevara-Vega, C. P., & García-Santillán, I. (2022). Software Project Management Integrating CMMI-DEV and SCRUM. *Communications in Computer and Information Science*, *1535 CCIS*, 538–551. https://doi.org/10.1007/978-3-031-03884-6_39

Gutiérrez, M. R. (2022). *La distribución de contenidos en la era del live streaming : análisis de la creación, producción y evolución en el consumo de Twitch en 2020 - 2021*. https://rephip.unr.edu.ar/handle/2133/23876

Haro, E., Guarda, T., Zambrano Peñaherrera, A. O., & Ninahualpa Quiña, G. (2019). *Desarrollo backend para aplicaciones web, Servicios Web Restful: Node.js vs Spring Boot*.

Haselmann, T., Thies, G., & Vossen, G. (2010). Looking into a REST-based API for database-as-a-service systems. *Proceedings - 12th IEEE International Conference on Commerce and Enterprise Computing, CEC 2010*, 17–24. https://doi.org/10.1109/CEC.2010.11

Hernández, A., Domínguez, J., Cruz, A., Hernández Paez, A., Alejandro Domínguez Falcón, J., Andrés, A., & Cruz, P. (2018). *Citación:-NC-SA 4.0 license Software architecture for the development of videogames on the game engine Unity 3D*. *14*, 54–64.

Hernández, L. M. A., Romero, V. A. P., González, S. A. S., & Rodríguez, J. A. V. (2021). Arquitectura REST para el desarrollo de aplicaciones web empresariales. *Revista Electrónica Sobre Tecnología, Educación y Sociedad*, *8*(15).

Hietala, J., Ala-Laurinaho, R., Autiosalo, J., & Laaki, H. (2020). GraphQL Interface for OPC UA. *Proceedings - 2020 IEEE Conference on Industrial Cyberphysical Systems, ICPS 2020*, 149–155. https://doi.org/10.1109/ICPS48405.2020.9274754

Hron, M., & Obwegeser, N. (2022). Why and how is Scrum being adapted in practice: A systematic review. *Journal of Systems and Software*, *183*, 111110. https://doi.org/10.1016/J.JSS.2021.111110

Iso 25000. (2018). *ISO/IEC 2502n – División de Medición de Calidad*. https://iso25000.com/index.php/normas-iso-25000/8-iso-iec-2502n

Karabey Aksakalli, I., Çelik, T., Can, A. B., & Teknerdoğan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, *180*, 111014. https://doi.org/10.1016/J.JSS.2021.111014

Koishybayev, I., & Kapravelos, A. (2020). Mininode: Reducing the attack surface of node.js applications. *RAID 2020 Proceedings - 23rd International Symposium on Research in Attacks, Intrusions and Defenses*, 121–134.

Lixandru, B., Buchmann, R. A., & Ghiran, A. (2022). *Towards a Modeling Method for Managing Node . js Projects and Dependencies*.

Llamuca-Quinaloa, J., Vera-Vincent, Y., Tapia-Cerda, V., Llamuca-Quinaloa, J., Vera-Vincent, Y., & Tapia-Cerda, V. (2021). Análisis comparativo para medir la eficiencia de desempeño entre una aplicación web tradicional y una aplicación web progresiva. *TecnoLógicas*, *24*(51), 164–185. https://doi.org/10.22430/22565337.1892

Maida, E. G., & Pacienzia, J. (2015). *Metodologías de desarrollo de software*. https://repositorio.uca.edu.ar/handle/123456789/522

Malinowski, E., Zimányi, E., Joseph, S. K., Warehouse, D., Inmon, B., Analytical, O., Olap, P., Gatziu, S., Vavouras, A., Nilsson, A. A., & Merkle, D. (2019). GraphQL. *Data Vault 2.0*, *January 1999*, 1–15. https://doi.org/10.1007/978-3-322-94873-1

Mavroudeas, G., Baudart, G., Cha, A., Hirzel, M., Laredo, J. A., Magdon-Ismail, M., Mandel, L., & Wittern, E. (2021). Learning GraphQL Query Cost. *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*, 1146–1150. https://doi.org/10.1109/ASE51524.2021.9678513

Meng, M., Steinhardt, S., & Schubert, A. (2018). Application programming interface documentation: What do software developers want? *Journal of Technical Writing and Communication*, *48*(3), 295–330. https://doi.org/10.1177/0047281617721853

Mera, A. Q. (2022). *Gobierno de arquitecturas híbridas rest y graphql basadas en acuerdos de nivel de servicio*. 3–16.

Milena Velásquez Restrepo, S., David Vahos-Montoya, J., Ester Gómez-Adasme, M., Alexandra Pino -Martínez, A., Juliett Restrepo-Zapata, E., & Londoño-Marín, S. (2019). Una revisión comparativa de la literatura acerca de metodologías tradicionales y modernas de desarrollo de software. *Revista CINTEX*, *24*(2), 13–23. https://doi.org/10.33131/24222208.334

Molina Montero, B., Vite Cevallos, H., & Dávila Cuesta, J. (2018). Metologías ágiles frente a las tradicionales en el proceso de desarrollo de software. *Espirales Revista Multidisciplinaria de Investigación*, *June*, 114–121. https://doi.org/10.31876/re.v2i17.269

Morales Inga, S., & Morales Tristán, O. (2020). ¿Por qué hay pocas mujeres científicas? Una revisión de literatura sobre la brecha de género en carreras STEM. *ADResearch ESIC International Journal of Communication Research*, *22*(22), 118–133. https://doi.org/10.7263/ADRESIC-022-06

Naciones Unidas. (2018). La Agenda 2030 y los Objetivos de Desarrollo Sostenible Una oportunidad para América Latina y el Caribe Gracias por su interés en esta publicación de la CEPAL. In *Publicación de las Naciones Unidas*. https://repositorio.cepal.org/bitstream/handle/11362/40155/24/S1801141_es.pdf

Novella, T., & Holubová, I. (2017). User-friendly and extensible web data extraction. *Information Systems Development: Advances in Methods, Tools and Management - Proceedings of the 26th International Conference on Information Systems Development, ISD 2017*.

Oliveira, T., Satoh, K., & Novais, P. (2019). *Evaluation of Open Source Software for Testing Performance of Web Applications*. *1*, 567–576. https://doi.org/10.1007/978-3-030-16184-2

Organización Internacional del Trabajo. (2017). Manual de referencia Sindical sobre la Agenda 2030 para el Desarrollo Sostenible. In *Organización Internacional del Trabajo* (Vol. 1). http://www.ilo.org/wcmsp5/groups/public/@ed_dialogue/@actrav/documents/publication/wcms_569914.pdf

Perdomo, W., & Zapata, C. M. (2021). Software quality measures and their relationship with the states of the software system alpha. *Ingeniare*, *29*(2), 346–363. https://doi.org/10.4067/S0718-33052021000200346

Porter, J., Menascé, D. A., & Gomaa, H. (2021). A decentralized approach for discovering runtime software architectural models of distributed software systems. *Information and Software Technology*, *131*, 106476. https://doi.org/10.1016/J.INFSOF.2020.106476

Pradanita, W. R., Ni'Mah, A. T., Rochimah, S., & Adiputra, F. (2019). Assessment of Academic Information System Quality from Two Perspectives: Product Quality and Quality in Use. *Proceedings of 2019 International Conference on Information and Communication Technology and Systems, ICTS 2019*, 135–140. https://doi.org/10.1109/ICTS.2019.8850933

Priyatna, F., Chaves-Fraga, D., Alobaid, A., & Corcho, O. (2019). Morph-GraphQL: GraphQL servers generation from R2RML mappings (SESE). *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, *2019-July*, 291–296. https://doi.org/10.18293/SEKE2019-055

Quiña-Mera, A., Fernandez, P., García, J. M., & Ruiz-Cortés, A. (2023). GraphQL: A Systematic Mapping Study. *ACM Computing Surveys*, *55*(10). https://doi.org/10.1145/3561818

Quiña-Mera, A., Fernández-Montes, P., García, J. M., Bastidas, E., & Ruiz-Cortés, A. (2022). Quality in Use Evaluation of a GraphQL Implementation. *Lecture Notes in Networks and Systems*, *405 LNNS*, 15–27. https://doi.org/10.1007/978-3-030-96043-8_2/COVER

Quiña-Mera, A., García, J. M., Fernández, P., Vega-Molina, P., & Ruiz-Cortés, A. (2022). GraphQL or REST for Mobile Applications? *Communications in Computer and Information Science*, *1675 CCIS*, 16–30. https://doi.org/10.1007/978-3-031-20319-0_2/COVER

Research, C., Rios, N., Mendonça, M., Seaman, C., & Oliveira Spínola, R. (2019). *Technical Debt Causes and Effects in Agile Projects Causes and Effects of the Presence of Technical Debt in Agile Software Projects*.

Rodríguez, K. (2020). Desarrollo De Un Envoltorio Del Api-Rest De Mendeley Con Graphql. *Universidad Técnica Del Norte*, 1–27. http://repositorio.utn.edu.ec/handle/123456789/10292

Román, Á. G. (2018). *Integrando fuentes heterogéneas de datos en Web con GraphQL*. 79.

Schuh, F. (2018). Graphene Documentation (Release). *White Paper*.

Schwaber, K., & Sutherland, J. (2020). La Guía Definitiva de Scrum: Las Reglas del Juego. *Scrum.Org and ScrumInc*, 16. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf

SNP, S. N. de P. (2021). Plan-de-Creación-de-Oportunidades-2021-2025-Aprobado. In *Plan de Creación de Oportunidades 2021-2025* (pp. 43-48-85–90).

Soni, A., & Ranga, V. (2019). API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering*, *8*(9 Special Issue), 664–671. https://doi.org/10.35940/ijitee.I1107.0789S19

Torres, J. (2018). *MOMMIV: Model for the decomposition of a monolithic architecture towards a microservices architecture under the principle of information hiding BIG DATA MACHINE LEARNING CYBERSECURITY View project CEDIA-CEPRA View project Victor Velepucha Escuela Politécn*.

Twitch Developers. (2022). *Twitch API | Twitch Developers*. https://dev.twitch.tv/docs/api

Universidad Técnica del Norte. (2020). *DIRECCIÓN DE INVESTIGACIÓN – Universidad Técnica del Norte*. https://www.utn.edu.ec/direccion/#1638194878129-33ebc452-1320

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 01

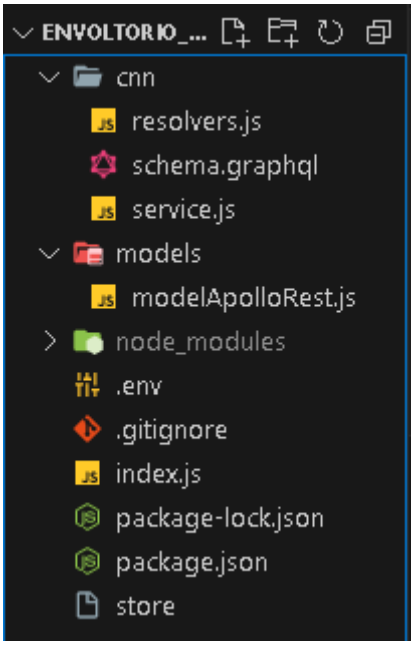
Código fuente y despliegue de los proyectos.

| Proyecto | Repositorio |
|---|---|
| Envoltorio GraphQL (Servicio) | https://github.com/aveliz12/envoltorio_graphql_twitch.git |
| Prueba de concepto (Cliente) | https://github.com/aveliz12/envoltorio_cliente_twitch.git |

### ANEXO 02

Se presenta la estructura del servicio del envoltorio GraphQL.

- Desarrollado con Node.js , donde se crea los archivos necesarios para el proyecto como los resolvers, el esquema graphql y el service los métodos de consumo de la API-REST. Además un archivo de modelo donde se codifica el modelo de consumo de cada funcionalidad donde se hace uso de la librería apollo-link-rest. Finalmente el archivo index de conexión para el uso de Apollo Client.




- El archivo index para la conexión.

```
const { ApolloServer } = require("apollo-server");
const { fileLoader, mergeTypes } = require("merge-graphql-schemas");
const typeDefs = mergeTypes(fileLoader(`${__dirname}/**/*.graphql`));
const resolvers = require("./cnn/resolvers");
require("dotenv").config({ path: "../variables.env" });

//server
const server = new ApolloServer({
  typeDefs,
  resolvers,
});

//run server

server.listen().then(({ url }) => {
  console.log(`🚀 Run server in the URL: ${url}`);
});
```

- El esquema GraphQL de los datos de la API-REST para el consumo de datos en Apollo Client.

```
type Token {
  access_token: String
  expires_in: Int
  token_type: String
}

type LiveStreams {
  id: String
  user_id: String
  user_login: String
  user_name: String
  game_id: String
  game_name: String
  type: String
  title: String
  viewer_count: Int
  started_at: String
  language: String
  thumbnail_url: String
  tag_ids: [String]
  tags: [String]
  is_mature: Boolean
  videosByGame(limitNivel2: Int): [VideosByGame]
}

```

- El archivo Resolvers donde se realiza los métodos para el consumo de datos.

```
const resolvers = {
  Query: {
    getToken: async () => {
      const params = new URLSearchParams();
      params.append("client_id", process.env.CLIENTID);
      params.append("client_secret", process.env.CLIENTSECRET);
      params.append("grant_type", process.env.GRANTTYPE);

      try {
        const response = await fetch("https://id.twitch.tv/oauth2/token", {
          method: "POST",
          body: params,
        });
        const data = await response.json();
        store.put("token", data.access_token);

        return data;
      } catch (error) {
        console.log(error);
      }
    },
    getCasosPruebasLiveStreams: async (_, { limitNivel1 }) => {
      try {
        const data = await getDataLiveStreams(limitNivel1);
        return data;
      } catch (error) {
        console.log(error);
      }
    }
  }
}
```

- Modelo de apollo-link-rest.

```
const { gql } = require("@apollo/client/core");

const queryLiveStreams = gql`
  query getLiveStreams($limitNivel1: Int, $cursor: String) {
    liveStreams(limitNivel1: $limitNivel1, after: $cursor)
      @rest(
        type: "liveStreams"
        path: "streams?first={args.limitNivel1}{args.after}"
      ) {
      data
      pagination
    }
  }
`;
```

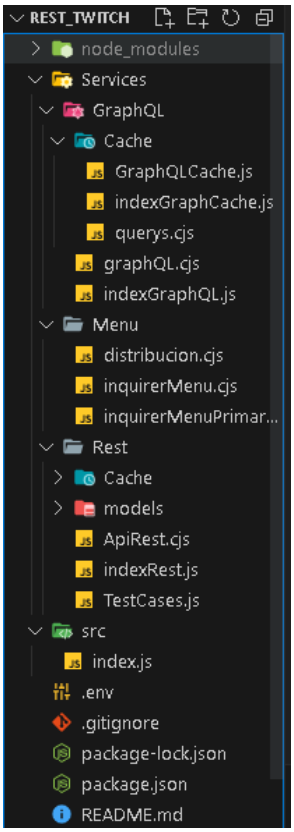
- El archivo service

```
const getDataVideos = async (id, first) => {
  try {
    let dataVideos = [];
    let cursor = null;
    while (first > 0) {
      const response = await getJsonTokenData(
        `videos?game_id=${id}&first=${first > 50 ? 50 : first}${
          cursor === null ? "" : `&after=${cursor}`
        }`
      );
      if (response.data.length > 0) {
        first = first - response.data.length;
        dataVideos = [...dataVideos, ...response.data];
        if (
          response.pagination.length > 0 ||
          response.pagination.cursor !== undefined
        ) {
          cursor = response.pagination.cursor;
        } else {
          break;
        }
      } else {
        break;
      }
    }

    return dataVideos;
  } catch (error) {
    console.log(error);
  }
};
```

**ANEXO 03**

Se muestra la aplicación cliente.

- Se crearon carpetas distintas para guardar archivos de cada arquitectura con y sin caché con el objetivo de consumir los datos. Para REST se usó el mismo modelo mostrado anteriormente. Además una carpeta donde se almacenan los archivos de Menú a usarse.




- GrapgQL sin caché.

```
const client = new ApolloClient({
  link: new HttpLink({ uri, fetch: buildAxiosFetch(instanceAxios) }),
  cache: new InMemoryCache(),
  defaultOptions: defaultOptions,
});

//LIVE STREAMS
const casoPrueba1GraphQL = async (limitNivel1) => {
  const t1 = performance.now();

  //CONSUMO
  const response = await client.query({
    query: DATA_CASOPRUEBA1,
    variables: {
      limitNivel1,
    },
  });
  let t2 = performance.now();

  const tiempo = getTime(t1, t2);

  console.log(
    `Datos del nivel 1 GraphQL: ${response.data.getCasosPruebasLiveStreams.length}.`
      .underline
  );
  return tiempo;
};
```

- GraphQL con caché.

```
const client = new ApolloClient({
  link: new HttpLink({ uri, fetch }),
  cache: new InMemoryCache(),
});

//LIVE STREAMS
export const casoPrueba1Cache = async (limitNivel1) => {
  const t1 = performance.now();

  //CONSUMO
  const response = await client.query({
    query: DATA_CASOPRUEBA1,
    variables: {
      limitNivel1,
    },
  });
  const dataLiveStreams = response.data.getCasosPruebasLiveStreams;
  const t2 = performance.now();

  const tiempo = getTime(t1, t2);
  console.log(
    `Datos del nivel 1 GraphQL con Cache: ${dataLiveStreams.length}.`.underline
  );
  return { data: dataLiveStreams, time: tiempo };
};
```

- REST sin caché.

```
//CASO DE PRUEBA 1: LIVE STREAMS
export const casoPrueba1 = async (first) => {
  let t1 = performance.now();
  console.log("INICIO CASO DE PRUEBA 1".red);
  const { data, requests } = await getLiveStreams(first);
  let t2 = performance.now();
  const tiempo = getTime(t1, t2);
  const numPeticiones = requests;

  return {
    data: data,
    time: tiempo,
    requests: numPeticiones,
  };
};
```

- REST con caché

```
//CASO DE PRUEBA 1 CACHE: LIVE STREAMS
const getCasoPrueba1RestCache = async (first) => {
  const t1 = performance.now();
  const { data, requests } = await getLiveStreamsCache(first);
  const t2 = performance.now();
  const tiempo = getTime(t1, t2);

  const numPeticiones = requests;
  return { data: data, time: tiempo, requests1: numPeticiones };
};
```

- Menús

```
import inquirer from "inquirer";
import "colors";

const menuOptions = [
  {
    type: "list",
    name: "opcion",
    message: "Tipo de consumo",
    choices: [
      {
        value: "1",
        name: "1. Consultar datos desde API-REST",
      },
      {
        value: "2",
        name: "2. Consultar datos de API-REST desde caché",
      },
      {
        value: "3",
        name: "3. Consultar datos desde GraphQL",
      },
      {
        value: "4",
        name: "4. Consultar datos de GraphQL desde caché",
      },
      {
        value: "0",
        name: "5. Regresar",
      },
    ],
  },
];
```

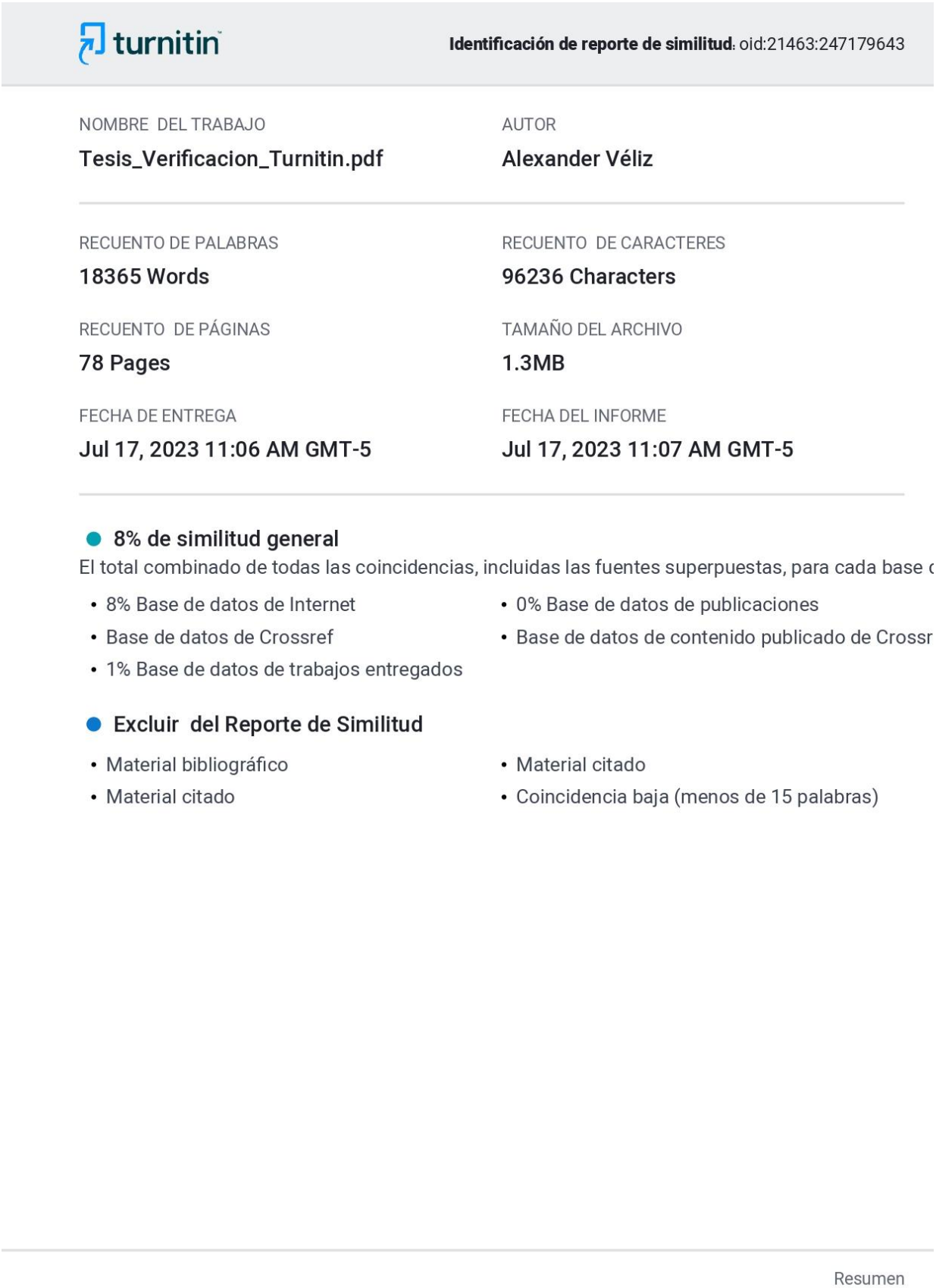
```
const inquirer = require("inquirer");

const menuOptions = [
  {
    type: "list",
    name: "opcion",
    message: "¿Qué desea hacer?",
    choices: [
      {
        value: "1",
        name: "Primer Nivel: Transmisiones en vivo.",
      },
      {
        value: "2",
        name: "Segundo Nivel: Videos por juego.",
      },
      {
        value: "3",
        name: "Tercer Nivel: Clips por usuario.",
      },
      {
        value: "4",
        name: "Cuarto Nivel: Información del canal de un usuario.",
      },
      {
        value: "5",
        name: "Quinto Nivel: Información de un juego.",
      },
      {
        value: "0",
        name: "Regresar",
      },
    ],
  },
```

- El index para correr el proyecto y ejecutar las funcionalidades.

```
const main = async () => {
  let opt = "";

  try {
    console.log("Desea generar un nuevo token? ");
    do {
      opt = await inquireMenuToken();

      switch (opt) {
        case "1":
          await getToken();
          token = store.get("token");

          console.log(`Su nuevo token es: ${token}`);

          await consumoTwtich();
          break;
        case "2":
          token = store.get("token");
          console.log(`Su token es: ${token}`);

          await consumoTwtich();
          break;
        default:
          break;
      }
    } while (opt !== "0");
  } catch (error) {
    console.log(error);
  }
};

main();
```

**ANEXO 04**

Verificación de plagio de Turnitin.



turnitin

**Identificación de reporte de similitud**: oid:21463:247179643

NOMBRE DEL TRABAJO
**Tesis_Verificacion_Turnitin.pdf**

AUTOR
**Alexander Véliz**

RECUENTO DE PALABRAS
**18365 Words**

RECUENTO DE CARACTERES
**96236 Characters**

RECUENTO DE PÁGINAS
**78 Pages**

TAMAÑO DEL ARCHIVO
**1.3MB**

FECHA DE ENTREGA
**Jul 17, 2023 11:06 AM GMT-5**

FECHA DEL INFORME
**Jul 17, 2023 11:07 AM GMT-5**

**8% de similitud general**

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

- 8% Base de datos de Internet
- 0% Base de datos de publicaciones
- Base de datos de Crossref
- Base de datos de contenido publicado de Crossr
- 1% Base de datos de trabajos entregados

**Excluir del Reporte de Similitud**

- Material bibliográfico
- Material citado
- Material citado
- Coincidencia baja (menos de 15 palabras)

Resumen