UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Ingeniería en Sistemas Computacionales

DESARROLLO DE UNA API – GRAPHQL PARA ANALIZAR LA EFICIENCIA DEL CONSUMO DE DATOS ENTRE BASE DE DATOS RELACIONALES Y NO RELACIONALES UTILIZANDO LAS MÉTRICAS DE LA NORMA ISO/IEC 25023.

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

Autor:

Luis Andrés Quinche Morán

Director:

MSc. José Antonio Quiña Mera

Ibarra – Ecuador 2021

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:	1003866173		
APELLIDOS Y NOMBRES:	Quinche Moran Luis Andrés		
DIRECCIÓN:	Otavalo, Calle Quito y Atahualpa		
EMAIL:	laquinchem@utn.edu.ec		
TELÉFONO FIJO:	2922618	**TELÉFONO MÓVIL:**	0994501275

DATOS DE LA OBRA	
TÍTULO:	Desarrollo de una api – graphql para analizar la eficiencia del consumo de datos entre base de datos relacionales y no relacionales utilizando las métricas de la norma ISO/IEC 25023
AUTOR (ES):	Luis Andrés Quinche Moran.
FECHA: DD/MM/AAAA	08/10/2021
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	**INGENIERIA EN SISTEMAS COMPUTACIONALES.**
ASESOR /DIRECTOR:	**Msc. Antonio Quiña.**

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 la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 10 días del mes de agosto de 2021

EL AUTOR:

(Firma).....
Nombre: .....Luis Andrés Quinche Moran

CERTIFICACIÓN DIRECTOR

En mi calidad de tutor de Trabajo de Grado presentado por el egresado LUIS ANDRES QUINCHE MORAN para obtener Título de Ingeniería en Sistemas Computacionales cuyo tema es: Desarrollo de una API – GraphQL para analizar la eficiencia del consumo de datos entre base de datos relacionales y no relacionales utilizando las métricas de la norma ISO/IEC 25023. Considero que el presente trabajo reúne los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del tribunal examinador que se designe.

En la ciudad de Ibarra, a los 10 días del mes de agosto del 2021

JOSE ANTONIO QUIÑA MERA

Firmado digitalmente por JOSE ANTONIO QUIÑA MERA
Fecha: 2021.08.10 16:53:36 -05'00'

Msc. Antonio Quiña

DIRECTOR DE TRABAJO DE GRADO

DEDICATORIA

Quiero dedicar esta tesis a:

A mis padres Luis Enrique Quinche “papi kike” y Rosa Elena Morán “mami Rosi”, por su amor, trabajo y apoyo incondicional en todo este proceso de mi formación académica. Son una bendición en mi vida, agradezco a Dios por darme unos padres como ustedes, espero que con esta parte de mi éxito pueda devolverles un poquito de lo que me han dado. Son los mejores padres.

A mi hermano Freddy y su esposa Blanquita, a mi hermana Ligia y su esposo Christian, a mi hermana Miriam y su esposo David, por estar siempre presentes y sufrir conmigo en esta carrera de la vida, han sido como mis padres, siempre dispuestos a ayudarme en todo lo que necesite, motivándome día a día para que no me rinda. Sus consejos me han servido de mucho.

A mis sobrinos Eliam, Joel, Koral, Sahily, Liam y Sydney que a pesar de ser unos niños, siempre estaban pendientes de mí, pues así como quieren lo mejor para mí yo también quiero lo mejor para ustedes, siempre estaré ahí cuando me necesiten.

A toda mi familia que me siempre me ha brindado todo su apoyo y amor, lo cual me ha servido de inspiración para poder culminar con éxito mis estudios y mi trabajo de grado.

AGRADECIMIENTO

Agradezco primeramente a Dios, por ser mi guía, por darme la salud, la sabiduría y la oportunidad de estudiar una carrera que me gusta.

A mis padres por nunca dejar de creer en mí, en mis capacidades, por ser esa fuente de motivación, por siempre regalarme una sonrisa pese a las circunstancias que se presentaron, fueron quienes me enseñaron a no rendirme, no me alcanzara la vida para demostrarles cuan agradecido estoy con ustedes. Todo lo que hice fue por ustedes.

A mi hermano Freddy y su esposa Blanquita, a mi hermana Ligia y su esposo Christian, a mi hermana Miriam y su esposo David, quienes han sido como mis padres pendientes de todo lo que necesite, motivándome día a día, dispuestos ayudarme, a guiarme y a corregirme si es necesario, por eso y mucho más, gracias.

A mi grupo de amigos; Alejandro, Carlos, Leo, Christian, Edison, Andrés, Jayli, Kevin, Paul, Robert y Sebastián, con quienes luchamos por aprobar cada semestre, compartimos de momentos difíciles y disfrutamos de los tiempos libres. Gracias por su ayuda.

A mi asesor de tesis MSc. Antonio Quiña gracias a su guía en este complicado proceso hizo posible la realización y culminación de este proyecto.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

GraphQL es un lenguaje de consulta que esta implementado en el lado del servidor y está siendo utilizado por las principales empresas de software, este aparece como una alternativa ante API – Rest. GraphQL no ha tenido un estudio amplio de la eficiencia del consumo de datos y de su comportamiento frente a bases de datos relacionales y no relacionales. Es por ello por lo que se construyó una API – GraphQL para realizar un análisis del consumo de datos entre base de datos relacionales y no relacionales utilizando las métricas de la eficiencia en el tiempo de respuesta dada por la norma ISO/IEC 25023.

Para empezar con la construcción de la API – GraphQL, se realizó una investigación de las bases de datos relacionales y no relacionales que han sido utilizadas en los dos últimos años (2018-2020). Se seleccionó tres motores de bases de datos de cada una. Por último, el desarrollo de este producto se lo hizo siguiendo la metodología ágil Scrum que posee ciclos interactivos llamados Sprints.

Después del desarrollo se realizó la comparativa de los tiempos de respuesta de las bases de datos seleccionadas que fueron consumidas desde la API – GraphQL. Esto se lo realizo mediante un experimento controlado que agrega validez a esta investigación.

El experimento consistió en realizar consultas que estaban definidas en tres casos de uso, se obtuvo los resultados mediante la fórmula del tiempo de respuesta dada en la norma ISO/IEC 25023, se comparó los resultados y la base de datos que obtenía el menor tiempo de entre todas las bases de datos(relacionales y no relacionales) era la ganadora. En este experimento se obtuvo como resultado que la base de datos más eficiente es la base de datos no relacional MongoDB y la que se adapta de mejor manera a esta nueva e interesante tecnología llamada GraphQL.

Palabras clave: GraphQL, API-GraphQL, base de datos, relacional, no relacional, experimento controlado, MongoDB.

ABSTRACT

GraphQL is a query language that is implemented on the server side and is being used by the main software companies, it's appears as an alternative to API – Rest. GraphQL has not had a wide research of the efficiency of data consumption and it's behavior against relational and non-relational databases. That it is why an API - GraphQL was built to perform an analysis of data consumption between relational and non-relational databases using the response time efficiency metrics given by the ISO / IEC 25023 standard.

To begin with the construction the API – GraphQL, an investigation was made of the relational and non-relational databases that have been used in the last two years (2018-2020). Three database engines were selected from each. Finally, the development of this product was done following the agile Scrum methodology that has interactive cycles called Sprints.

After development, the response times of the selected databases that were consumed from the API - GraphQL were compared. This was done through a controlled experiment that adds validity to this research.

The experiment consisted of execute queries that were defined in three use cases, the results were obtained using the response time formula given in the ISO / IEC 25023 standard, the results were compared and the database that obtained the shortest time of all the databases (relational and non-relational) was the winner. In this experiment it was obtained as a result that the most efficient database is the non-relational MongoDB database and the one that best adapts to this new and interesting technology called GraphQL.

Keywords: GraphQL, API-GraphQL, database, relational, non-relational, controlled experiment, MongoDB.

INTRODUCCIÓN

Problema

Antecedentes:

Las bases de datos son un conjunto de datos estructurados o información que están almacenadas en un sistema informático, por lo que un programa informático puede utilizar un lenguaje de búsqueda para recuperar esta información (Čerešňák & Kvet, 2019). Una de estas opciones es API – GraphQL que aparece como una alternativa ante API – Rest.

GraphQL es un nuevo concepto en la construcción de APIs. Este es un lenguaje de consulta desarrollado por Facebook e implementado en el lado del servidor, originalmente estaba destinado exclusivamente a su aplicación. La causa fue la remodelación de las aplicaciones móviles de Facebook para iOS y Android, las cuales mostraban un rendimiento cada vez más bajo debido a un aumento de la complejidad al momento de consumir los datos. Aunque es un lenguaje de consulta, GraphQL no está conectado directamente con la base de datos. En otras palabras, GraphQL no se limita a bases de datos SQL y NOSQL (Hartina et al., 2018).

Situación Actual:

GraphQL es un lenguaje de consulta que puede ser utilizado en diferentes códigos de programación, gracias a la gran compatibilidad y capacidad multisistema es posible conectar información y comunicar resultados desde diferentes entornos.

Últimamente, en la comunidad científica y tecnológica se está hablando y adoptando el lenguaje de consulta GraphQL para el desarrollo de APIs Web. Aunque es un lenguaje de consulta, GraphQL no está vinculado a ninguna base de datos o motor de almacenamiento específico, sino que está garantizado por el código y datos existentes. La posición de GraphQL está en el lado del servidor ejecutando la consulta usando el sistema de tipos especificado para los datos (Hartina et al., 2018).

Actualmente no existe un estudio entre la comparativa de la eficiencia del consumo de datos entre base de datos relacionales y no relacionales desde una API – GraphQL, este trabajo intenta realizar un aporte inédito en el ámbito de desarrollo de Arquitecturas Orientadas a microservicios (Roksela et al., 2020).

Prospectiva:

El consumo de datos desde una API – GraphQL cada vez es más común en estos últimos años, grandes empresas han optado por utilizar esta herramienta de consulta, debido al gran poder que tiene.

Se propuso realizar una investigación de la comparativa del consumo de datos desde tres bases de datos no relacionales y tres bases de datos relacionales desde una API – GraphQL, mediante experimentos utilizando casos de uso de consumo de datos en las bases de datos mencionadas. Este análisis aportara valor a la comunidad desarrolladora para que puedan optar utilizar esta herramienta de consulta e integrarlos en sus proyectos, además de distinguir la base de datos que más se ajusta a esta herramienta.

Planteamiento del Problema:

Con el aumento de la tasa de adopción de GraphQL en todas las organizaciones, es cada vez más importante conocer cómo pueden funcionar las API basadas en GraphQL. Esto debe verificarse entre las diferentes condiciones de ejecución que proporciona la misma herramienta, pero no existe un estudio amplio de la eficiencia del consumo de datos desde una API – GraphQL.

Para poder definir el diagrama de Planteamiento de Problema se utilizó el instrumento de investigación de identificación y clasificación de problemas (Matriz Vester).

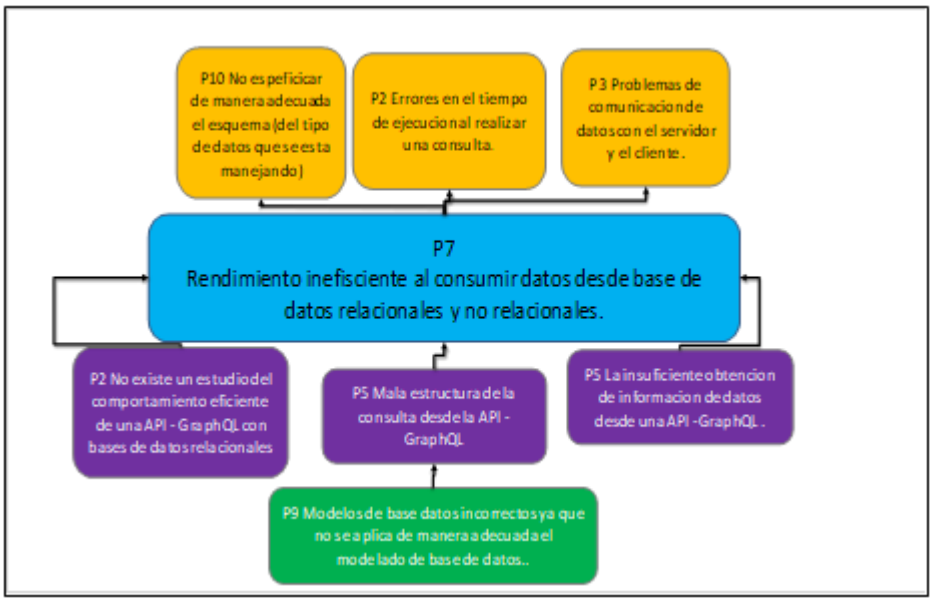


Fig. 1. Árbol de Problemas

Objetivos

Objetivo General:

Desarrollar una API – GraphQL para realizar el análisis de la eficiencia del consumo de datos entre base de datos relacionales y no relacionales utilizando las métricas de la norma ISO/IEC 25023.

Objetivos Específicos:

- Establecer un marco teórico para el desarrollo del estudio.
- Desarrollar un API – GraphQL para el manejo de consumo de datos en base de datos relacionales y no relacionales.
- Realizar la comparativa de la eficiencia del consumo de datos entre bases de datos relacionales y no relacionales mediante el API – GraphQL desarrollado a través de experimentos utilizando la métrica de la Eficiencia en el desempeño - (Tiempo de respuesta) de la norma ISO/IEC 25023.
- Analizar los resultados obtenidos en la investigación planteada.

Alcance

Mediante el presente estudio que se realizará en los siguientes seis meses, se construirá una API – GRAPHQL utilizando el lenguaje JavaScript con el framework Node.js ya que es un lenguaje de scripts de alto nivel incorporado en los navegadores que permite implementar interactividad en páginas web y apps, debido a que es muy usado al realizar este tipo de servicio web y de esta manera poder consumir los datos desde tres bases de datos relacionales (las más significativas) y tres bases de datos no relacionales (las más significativas) para realizar una comparativa de la eficiencia del consumo de datos entre estas bases de datos mediante experimentos de laboratorio, por último, se analizara los datos obtenidos.

El experimento se lo realizara utilizando casos de uso (consulta de datos), que consultaran a las bases de datos del estudio. Para esto se realizará una investigación de las tres bases de datos relacionales que han sido importantes en los últimos dos años y tres bases de datos no relacionales que han sido importantes en los últimos dos años para la comunidad científica y tecnológica. Finalmente se realizará un análisis de los resultados obtenidos.

Para la evaluación de la eficiencia del consumo de datos se utilizará las métricas de la norma ISO/IEC 25023 y así visualizar de mejor manera la productividad de la API - GraphQL**.**

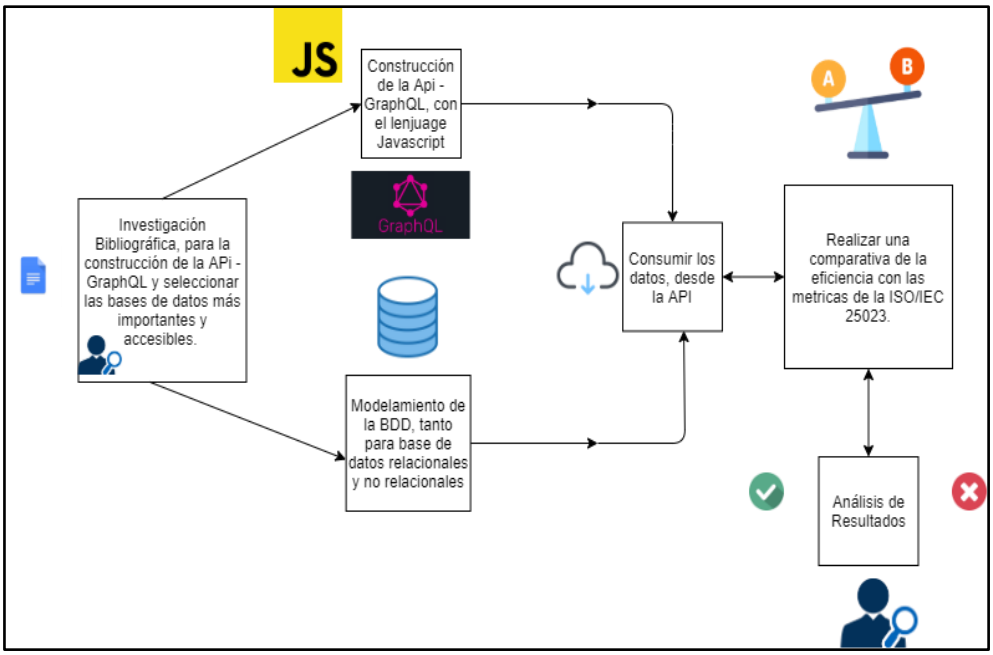


Fig. 2. Diagrama del Alcance.

Justificación

Este estudio está enfocado en la comparativa de la eficiencia del consumo de datos desde una API – GraphQL entre bases de datos relacionales y no relacionales, esto puede contribuir como una herramienta a la comunidad de desarrolladores y puedan distinguir con que base de datos una API – GraphQL puede trabajar de manera más eficiente, y puedan integrarla en sus aplicaciones o proyectos.

El enfoque del siguiente estudio hacia los objetivos de desarrollo sostenible (ODS) son los siguientes:

Objetivo 4) Educación de calidad: La investigación que se realizará en este trabajo podrá contribuir con la formación técnica además de obtener conocimiento acerca del contenido de esta investigación y contribuirá a la comunidad científica y tecnológica. (Programa de las Naciones Unidas para el Desarrollo, 2018).

Objetivo 9) Industria, innovación e infraestructura: La innovación y el progreso tecnológico ayuden a generar un crecimiento económico y laboral (Programa de las Naciones Unidas para el Desarrollo, 2018).

Justificación Tecnológica.

Debido a que GraphQL puede ser utilizado en diferentes códigos de programación, gracias a la compatibilidad y capacidad multisistema (Hartina et al., 2018), se utilizará el lenguaje de programación JavaScript que es muy amigable a la hora de intercambiar datos entre

aplicaciones web y el servidor. Para medir la eficiencia de una API – GraphQL se utilizará las métricas de la norma ISO/IEC 25023.

Justificación Teórica.

Se realizará una comparativa de la eficiencia de consumo de datos desde una API – GraphQL entre tres bases de datos relacionales y tres no relacionales para medir la eficiencia del consumo de datos, esta investigación contribuirá con evidencia empírica que fortalecerá la base de conocimiento acerca de GraphQL ante la comunidad científica y tecnológica.

Justificación Metodológica.

Mediante la utilización de la experimentación como método empírico de la ingeniería de software, se realizará la comparativa de la eficiencia del consumo de datos desde una API – GraphQL.

CAPÍTULO 1

Marco Teórico.

1.1. Arquitectura Orientada a Microservicios.

1.1.1. Introducción

La arquitectura orientada a microservicios es un estilo arquitectónico muy popular y adoptado por servicios de Internet de fama mundial como Netflix, Amazon y eBay (Li, 2018). Este tipo de arquitectura ha evolucionado y ganado una popularidad significativa en los últimos años, ofreciendo varios beneficios en comparación con las arquitecturas existentes en la Ingeniería de Software actualmente (Munaf et al., 2019). Los microservicios son un modelo de arquitectura que nos ayudan a diseñar aplicaciones y cada función se denomina servicio, este se puede diseñar e implementar de forma independiente. Esto permite que los servicios funcionen independientemente sin afectar a los demás (Ret Hat, 2019), aprovechando la idea de un diseño de software modular, escalable, resistente y reutilizable (Petrasch, 2017).

Para que las arquitecturas de microservicios funcionen en la práctica, se debe obtener información dentro y fuera de estos servicios y encontrar la manera de hacer que el intercambio de información sea seguro. Existen estándares que proporcionan la base para tal intercambio de datos. Los formatos de datos más populares en la computación en la nube son la notación de objetos JavaScript (JSON) y XML (Sill, 2016).

La comunicación del servicio web se realiza a través de componentes como la API de recursos HTTP. La interfaz de programación de aplicaciones o API es un fragmento de código que permite que puedan interactuar dos componentes de software. Las aplicaciones de software recientes se están distribuyendo en varios servidores que interactúan con las aplicaciones de back-end a través de interfaces estandarizadas. Las API REST o RESTful son las APIs más utilizadas en los últimos años pero GraphQL aparece como alternativa del API REST (Isha et al., 2018).

En la Fig. 3 se ha ilustrado la estructura de una arquitectura orientada a microservicios.

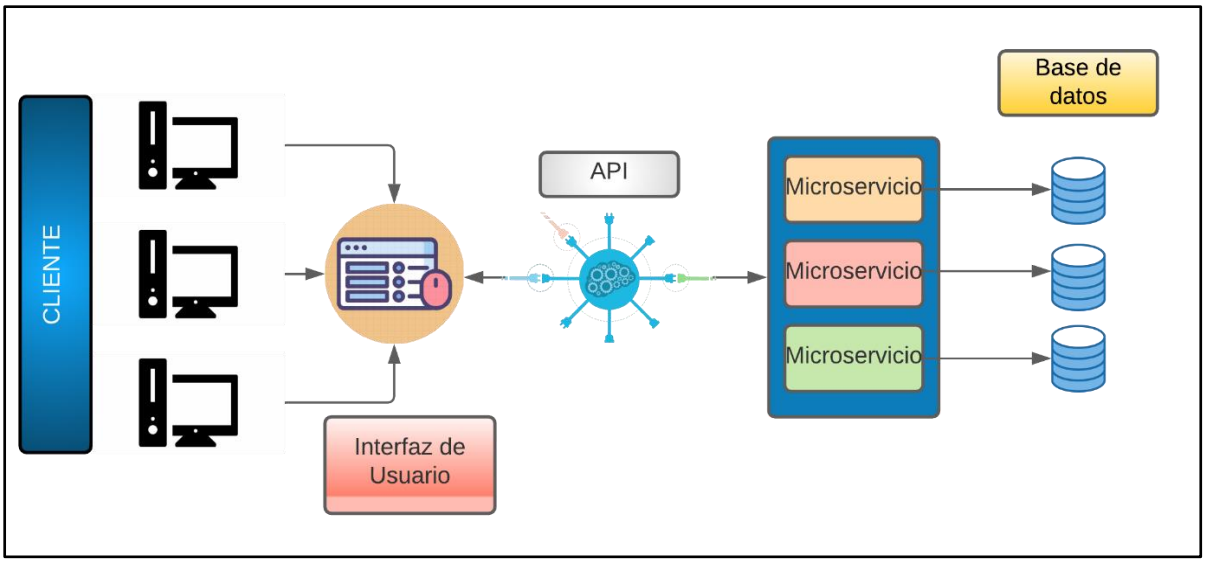


Fig. 3. Estructura de la arquitectura orientada a microservicios.

1.1.2. API (Interfaz de Programación de Aplicaciones).

El API ayuda a conectar dos componentes de software (back-end y front-end) existen algunos modelos de APIs que se detallaran a continuación:

- **API SOAP.**

Tanto (REST) y (SOAP) el Protocolo simple de acceso a objetos, son los dos principales servicios web más utilizados en los últimos años, SOAP es un protocolo subyacente de REST, pero ambos servicios se utilizan para gestionar la comunicación en la World Wide Web (www), esta comunicación se realiza mediante XML (Soni & Ranga, 2019).

- **API REST.**

En los últimos años la rápida y dominante API-REST (Transferencia de estado representacional) se ha establecido como un modo para realizar sistemas distribuidos y se supone que ganarán aún más importancia en el entorno de la computación en la nube, internet de las cosas y microservicios (Haupt et al., 2017).

- **API GraphQL.**

GraphQL es un lenguaje de consulta diseñada por Facebook, en la sección 1.2 se detallará a más profundidad sobre la definición de GraphQL.

1.1.3. Diferencias entre Arquitectura Orientada a Microservicios y Arquitectura Monolítica.

La arquitectura de los microservicios está ganando campo en la comunidad académica y la industria tecnológica, grandes empresas están migrando a las arquitectura orientadas a microservicios, ya que una de sus principales ventajas es que pueden depender de la complejidad de la tecnología que se utiliza, lo que significa que cada servicio en un sistema puede utilizar una tecnología diferente a los otros servicios para lograr los objetivos y el rendimiento deseado, otra de las ventajas más comunes de los microservicios es que si un componente del sistema falla, no afectara a todo el sistema, lo cual es grandioso a la hora de estar creando aplicaciones con este tipo de arquitectura. Por otro lado, la arquitectura monolítica como su nombre lo dice es una arquitectura desarrollada en una sola pieza, claro que sus aplicaciones se pueden crear a partir de decenas o cientos de servicios diferentes que están estrechamente acoplados en una base de código monolítica (en una sola pieza). Esto puede crear muchas dificultades para los equipos de desarrollo que trabajan en el mismo entorno. Es por ello por lo que están prefiriendo la arquitectura de microservicios sobre la arquitectura monolítica, para permitir que sus equipos de desarrollo se coordinen entre sí fácilmente (Martinek, 2019).

1.2. GraphQL.

1.2.1. Introducción.

GraphQL es un nuevo concepto en la construcción de APIs, fue creado en el 2015 por Facebook e implementado en el lado del servidor (Hartina et al., 2018) este lenguaje representa una alternativa a las APIs populares basadas en REST para resolver ciertas limitaciones que presentaba en algunos casos (Brito et al., 2019).

El lenguaje está ganando impulso y ahora lo utilizan las principales empresas de software, como Facebook y GitHub (Brito & Valente, 2020).
Aunque es un lenguaje de consulta, GraphQL no está conectado directamente con la base de datos. En otras palabras, GraphQL no se limita a bases de datos SQL y NOSQL (Hartina et al., 2018).

1.2.2. Lenguaje de consultas.

Como se mencionó anteriormente, GraphQL es un lenguaje de consulta para crear APIs, a pesar de que es un lenguaje de consulta no está vinculado a ninguna base de datos o motor de almacenamiento especifico, GraphQL es solo un traductor de lenguaje de consulta, por lo que no depende del lenguaje de programación del lado del servidor y de ninguna base de datos, este lenguaje de consultas ejecuta las consultas del lado del servidor y devuelve solo

los datos definidos por un sistema de tipos en el servicio web correspondiente (Hartina et al., 2018).

1.2.3. Estructura de consultas y mutaciones.

GraphQL es un lenguaje de consulta de datos para implementar servicios basados en web centrados en abstracciones de alto nivel, como esquemas, tipos, consultas y mutaciones, al usar GraphQL, los clientes pueden definir exactamente los datos que requieren los proveedores de servicios (Brito & Valente, 2020).

- **Campos.**

Las consultas se definen mediante un tipo, llamado Query, aunque a veces no es necesario colocar la palabra reservada Query para realizar la consulta, esta es la forma más simple, GraphQL trata de solicitar campos específicos en objetos (GraphQL, 2020a).

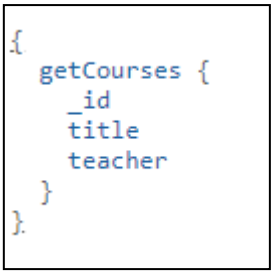


Fig. 4. Campos de una consulta GraphQL.

- **Argumentos.**

Los campos de la consulta pueden verse como "funciones" (implementadas en el back-end) que devuelven valores; y como "funciones", pueden aceptar argumentos para alterar su comportamiento o los mismos valores devueltos (Vazquez-Ingelmo et al., 2017)**.**

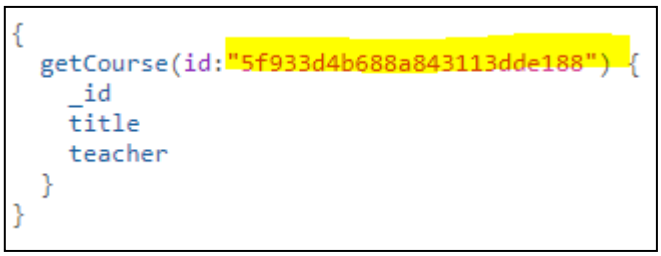


Fig. 5. Argumentos en una consulta GraphQL.

"id" es un argumento que le dice al backend qué curso en particular está solicitando el cliente. Por supuesto, debe haber mecanismos implementados en el backend para filtrar

los objetos de datos por un identificador y, en consecuencia, hacer disponible el argumento "id" (Vazquez-Ingelmo et al., 2017).

1.2.4. Esquema y sistema de tipos.

La base de una API-GraphQL es el Schema. El Schema es un documento que describe detalladamente todos los tipos, datos y métodos de consulta que tendrá la API.

Para ilustrar de mejor manera los siguientes conceptos se ha creado este esquema Fig. 6. de un sistema simple de una escuela que contiene: Estudiantes y Cursos.

```
1   type Course{
2       _id: ID!
3       title: String!
4       teacher: String!
5       description: String!
6       topic: String
7       student:[Student]
8   }
9   type Student {
10      _id: ID!
11      name: String!
12      email: String!
13      avatar: String
14  }
15  type Query{
16      "Devuelve todos los cursos"
17      getCourses:[Course]
18      "Devuelve un solo curso, el que ud defina"
19      getCourse(id:ID!):Course
20  }
21  type Mutation {
22     "Crear un curso"
23         createCourse (input: Course!): Course
24     "Editar un curso"
25         editCourse (id:ID!, input: Course!): Course
26     "Eliminar un Curso"
27        deleteCourse (id:ID!): Course
28  }
```

Fig. 6.Esquema de una Api - GraphQL.

A continuación, se expondrá algunos elementos básicos del Schema de una API - GraphQL

- **Campos y tipos de objetos.**

Una API-GraphQL se define mediante un esquema, Brito (2019) describe al esquema como un gráfico múltiple, el cual contiene nodos que son muy similares a los objetos en JavaScript. Los tipos de objetos (nodos) se definen utilizando la palabra clave ***type*** estos contienen un campo o una lista de campos, cada campo tiene un nombre y también un tipo de dato (Brito et al., 2019).

Por ejemplo, en la Fig.6 (línea 2) se puede observar el tipo de objeto llamado ***Course*** que contiene seis campos: ***id, title, teacher, description, topic, student***. Los campos ***id***, ***title,***

teacher y description son cadenas no nulas ya que el signo (!) después del tipo descarta los valores nulos, mientras que el campo llamado **t*opic*** puede aceptar valores nulos. El campo llamado ***student*** (línea 7) recibe otro objeto en el esquema llamado ***Student.***

- **Tipos consulta y mutación (type query, type mutation).**

Type Query.

Los esquemas incluyen un tipo predefinido llamado *Query* (consulta), que representa el punto de entrada de las API GraphQL. Un objeto *Query* muestra los tipos de objeto que pueden consultar los clientes y los argumentos que se deben proporcionar (Brito et al., 2019).

Por ejemplo, en la Fig. 6 (línea 17) muestra una consulta que va a devolver un objeto ***Course***, el cual se refiere a todos los cursos existentes, mientras que la consulta (línea 19) acepta una cadena no nula como argumento y devuelve un objeto ***Course*** que tiene esta cadena como id.

Type Mutation.

Los esquemas incluyen un tipo predefinido llamado ***Mutation*** y se utiliza para insertar nuevos objetos en la base de datos del servidor, además de modificar o eliminar los existentes.(Brito et al., 2019) En el ejemplo de la Fig. 6 se define tres mutaciones:

- La mutación para crear un nuevo curso (línea 23) es el ***createCourse***, que pide un objeto curso con los tipos de datos que se declararon anteriormente en el ***type Course.***
- La mutación para editar un curso (línea 25) es el ***editCourse,*** que pide 2 parámetros; uno es el ID del curso que será editado y el siguiente es el objeto ***Course*** que se ha editado.
- La mutación para eliminar un curso (línea 27) es el ***deleteCourse***, que pide solo un parámetro que es el ID.

1.2.5. Ejecución.

GraphQL ejecuta una consulta que devuelve un resultado reflejando la forma de la consulta solicitada, normalmente como JSON (GraphQL, 2020a)**.**

- **Resolvers.**

Para responder a las consultas, el desarrollador de un servidor GraphQL debe implementar una función llamada ***resolver*** para cada consulta declarada en el tipo de

consulta. Estas funciones se llaman cada vez que el motor del servidor GraphQL necesita recuperar un tipo de objeto especificado en una consulta (Brito et al., 2019).

```
Query: {
  human(obj, args, context, info) {
    return context.db.loadHumanByID(args.id).then(
      userData => new Human(userData)
    )
  }
}
```

Fig. 7. Ejemplo de un resolver.
Fuente: (GraphQL, 2020a)

Existen ***resolvers*** asincrónicos que permiten cargar datos desde una base de datos, esto devuelve una Promesa. Las promesas se utilizan para trabajar con valores asincrónicos. Cuando la base de datos regresa un dato, podemos construir y devolver un nuevo objeto.(GraphQL, 2020a). La siguiente imagen muestra un resolver asincrónico que está cargando datos desde a una base de datos NoSQL (MongoDB).

```
getCourses: async()=>{

    let courses = []
    try {
      db = await connectDB()
      courses = await db.collection('course').find().toArray()
      return courses
      }catch(error){
      console.error(error)
    }
},
```

Fig. 8. Ejemplo de un resolver asincrónico en GraphQL.

- **Result.**

El resultado de la consulta se presenta en la Fig. 9. Como se puede ver, el resultado es un objeto JSON que se asemeja a la estructura de la consulta (Brito & Valente, 2020).

Fig. 9. Ejemplo del resultado de una consulta en GraphQL.

1.3. Bases de datos relacionales y no relacionales.

1.3.1. Introducción.

La base de datos es un conjunto de datos estructurados o información almacenada en un sistema informático, por lo tanto, un programa informático puede utilizar un lenguaje de consulta para recuperar esta información. El programa informático utilizado para la gestión de datos y las consultas se llama SRBD (Sistema de gestión de bases de datos). Los científicos informáticos pueden clasificar el sistema de gestión de bases de datos de acuerdo con los modelos de base de datos que admiten. Las primeras bases de datos mencionadas relacionales se volvieron dominantes en la década de 1980. Admiten el uso de filas y columnas en una serie de tablas. Usan SQL para escribir y consultar los datos. Las bases de datos no relacionales se han vuelto populares en la década de 2000, llamadas “NoSQL” porque usan un lenguaje de consulta diferente a SQL (Čerešňák & Kvet, 2019).

1.3.2. Bases de datos relacionales SQL.

Las bases de datos relacionales son aquellas que están escritas en lenguaje SQL (Structure Query Language). El modelo relacional está bien adaptado a la programación cliente-servidor, hoy es una tecnología predominante para almacenar datos estructurados en aplicaciones web y comerciales (Now et al., 2020).

En la década de 1970, cuando la base de datos relacional entró en escena, los esquemas de datos sobre los que trabajar eran razonablemente elementales y simples en los que los elementos de datos debían organizarse como un conjunto de tablas descritas

formalmente con filas y columnas. Pero con la necesidad de almacenar variedad de datos (no estructurados) (Now et al., 2020).

En la Fig. 10 se puede observar la estructura del diseño de una Base de Datos SQL.

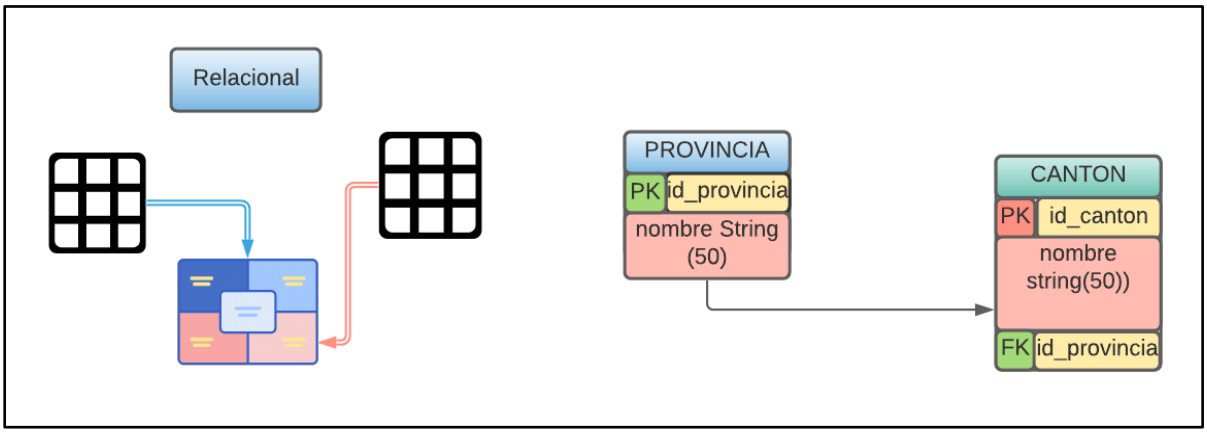


Fig. 10. Estructura de un diseño de Base de Datos SQL.

1.3.3. Base de datos no relacionales NoSQL.

Con el paso de los años la cantidad de datos trasmitidos por internet aumentan exponencialmente a una velocidad impresionante. Una gran parte de estos datos son manejados por sistemas de administración de bases de datos relacionales (RDBMS). Sin embargo, manipular un volumen tan grande de datos que deben entregarse a través de internet para llegar intactos a su destino, es un proceso que para las Bases de datos relacionales requieren mucho tiempo y, por lo tanto, se desarrolló una alternativa para esto llamada base de datos NoSQL ("No SQL", "No relacional" o "No solo SQL") (Now et al., 2020).

En los últimos años, han surgido varias tecnologías de bases de datos que permiten agrupar los datos de forma más natural y lógica. Una de las formas más populares de almacenar datos es una base de datos orientada a documentos, básicamente empleada para almacenar datos de gran magnitud. Se considera como un documento ya que la forma en la que se almacena los datos es en formato JSON que tienen un par de clave-valor, además de administrar y recuperar datos semiestructurados. Los documentos hacen que el procesamiento sea más rápido y más fácil para distribuir los datos a través de múltiples servidores(Now et al., 2020).

La Fig. 11 es un ejemplo de la estructura del diseño de la Base de Datos NoSQL.

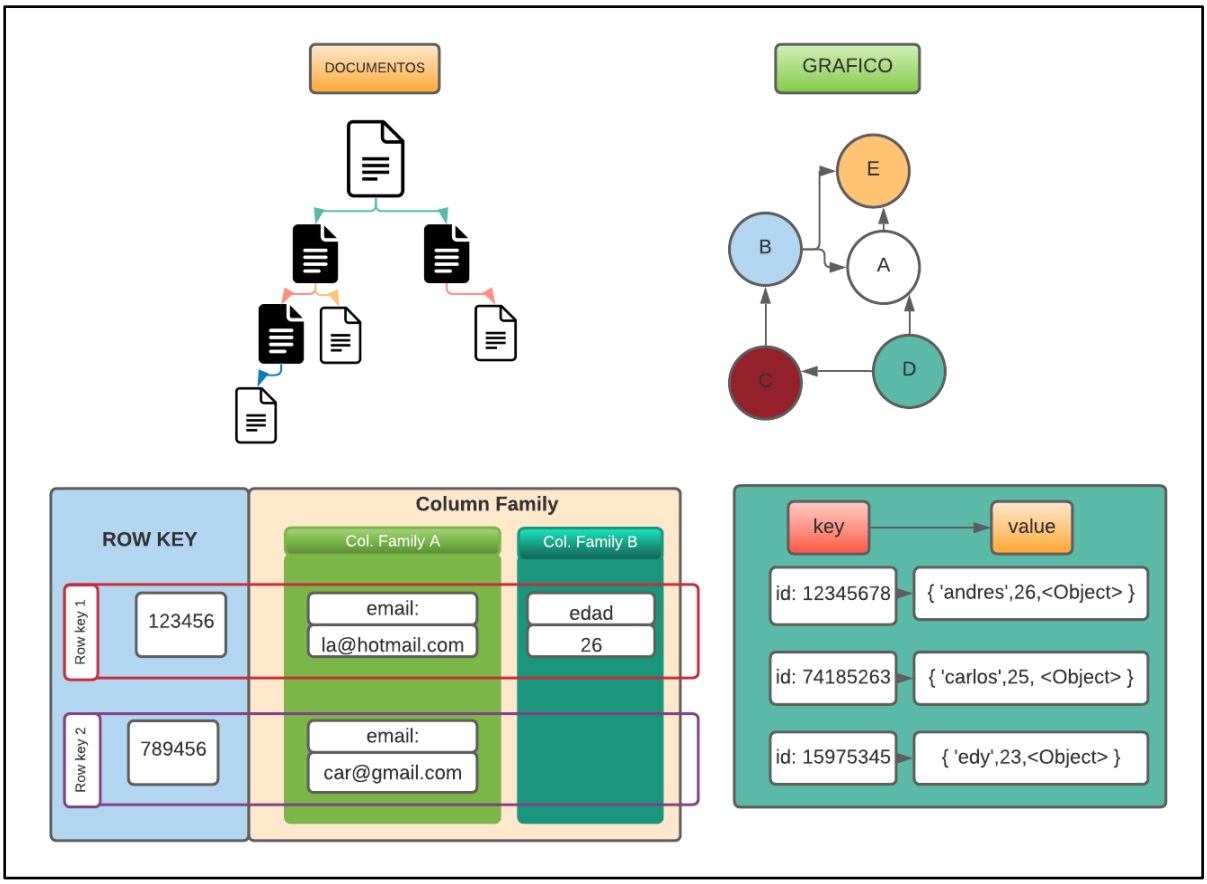


Fig. 11. Estructura del diseño de Base de Datos NoSQL.

1.3.4. SQL VS NoSQL.

Ambos motores de bases datos son utilizados por la comunidad científica y tecnológica, sin embargo, la diferencia más significativa entre estas dos opciones es que la base de datos SQL es relacional y contiene claves externas, mientras que la base de datos NoSQL no define las relaciones.

La TABLA 1. muestra las diferentes características de base de datos SQL y NoSQL.

TABLA 1. DIFERENCIAS ENTRE BASE DE DATOS SQL Y NOSQL.
Fuente: (Čerešňák & Kvet, 2019).

Propiedades	SQL	NoSQL
La forma de almacenar los datos.	Tablas.	Documentos, clave - valor.
Organización de datos.	Un esquema predefinido	Un esquema dinámico
Escalabilidad (aumento del rendimiento).	Vertical (RAM más grande, procesador más fuerte)	Horizontal (más servidores, instancias)
Lenguaje de consulta.	SQL estandarizado	Lenguaje de consulta propio
Intercambio de datos.	Llaves extranjeras	Documentos anidados

Seguridad.	Transacciones, consistencia, aislamiento	No existe

Como se puede observar en la TABLA 1, SQL parece superar a NoSQL, ya que NoSQL no ofrece ninguna característica de seguridad, pero lee, escribe y recupera datos, cumple su función como base de datos. Por lo tanto, se utiliza para aplicaciones BigData, donde se esperan datos en terabytes. NoSQL también resuelve el problema, cuando no conocemos los esquemas de la base de datos. Por el contrario, debemos abordar la coherencia de los datos en el lado de la aplicación, y esa es la desventaja con respecto a SQL (Čerešňák & Kvet, 2019).

1.3.5. Bases de datos más utilizadas en los últimos años.

En esta sección se muestra una investigación realizada en cuatro distintas plataformas, las cuales se ha considerado para obtener información valiosa acerca de las bases de datos (relacionales y no relacionales) que son populares entre la comunidad de desarrolladores.

- **Estudio en la plataforma Platzi.**

Platzi es una plataforma de educación en línea latinoamericana más importante en los últimos años que ha ganado una popularidad significativa ya que tienen muchos cursos que ayudan a crecer profesionalmente en el ámbito de la tecnología, marketing, diseño, etc. Se ha escogido esta plataforma ya que sus profesores son profesionales que trabajan en las principales empresas tecnológicas como Microsoft, IBM, AWS entre otras. Platzi menciona en uno de sus foros acerca de las bases de datos más utilizadas, en las que presenta tres relacionales y dos no relacionales.

A continuación, se detalla cuáles son las bases de datos más utilizadas en la comunidad de desarrolladores y el por qué son escogidas:

Dentro de los motores de bases de datos relaciones se encuentran tres distintas opciones:

- **MySQL:** Es el más utilizado, por el stack LAMP (Linux, Apache, MySQL y PHP) es la más acogida ya que con eso trabaja la mayor parte de internet.
- **PostgreSQL:** Según Platzi esta base de datos ofrece mejor rendimiento, tiene ciertas características como almacenamiento de archivos y objetos en formato JSON.
- **SQL Server:** Esta base de datos es muy utilizada dentro del software corporativo y bancario

(Platzi, 2020).

En cuanto a los motores de bases de datos no relacionales, las más utilizadas son:

- **Redis:** Es una base de datos no relacional y utilizada para hacer caché, la mayoría de los frameworks de desarrollo web ofrecen soporte y compatibilidad con esta base de datos.
- **MongoDB:** Esta base de datos no relacional está orientada a documentos y los datos se almacenan en estructuras de tipo BSON (Binary JSON).

(Platzi, 2020).

- **Estudio en la plataforma Jetbrains.**

Según la compañía de desarrollo de software **JetBrains** (2020) en una encuesta realizada en el 2019 a sus usuarios y obtuvo los siguientes resultados en cuanto a las bases de datos más utilizadas (JerBrains, 2020).

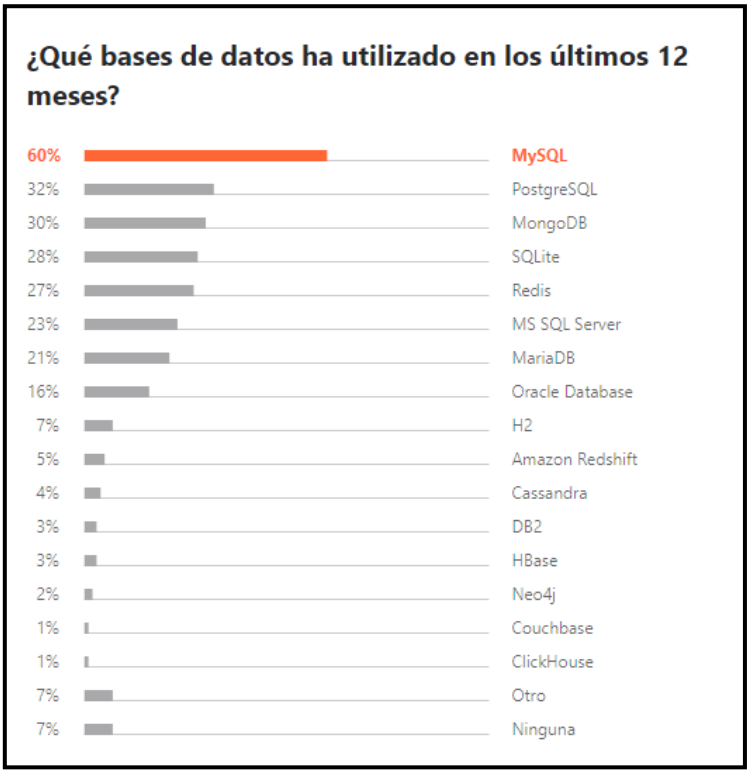


Fig. 12. Raking de las base de datos más utilizadas según JetBrains.

Fuente: (JerBrains, 2020).

En esta investigación se puede apreciar que MYSQL supera a las demás Bases de Datos.

- **Estudio en la plataforma Explore Group.**

Explore Group es uno de los especialistas en reclutamiento digital y de tecnología de más rápido crecimiento en el Reino Unido, este muestra otros resultados basados en la encuesta anual para desarrolladores de Stack Overflow 2019. En su investigación se pudo observar cuáles son las bases de datos; desde las más populares hasta las más temidas, además cuentan con una información efectiva de cómo ha cambiado el mercado de bases de datos durante el último año (Explore Group, 2019).

Plataformas de bases de datos más populares				♥ Amor		💔 Pavor		★ Desear	
	2019	2018	%Cambio	2019	2018	2019	2018	2019	2018
MySQL	52%	59%	-7%	54%	49%	46%	51%	8%	8%
PostgreSQL	36%	33%	3%	70%	62%	30%	38%	14%	11%
MS SQL Server	34%	42%	-8%	58%	52%	43%	48%	3%	4%
SQLite	30%	20%	10%	56%	48%	45%	52%	7%	53%
MongoDB	26%	26%	0%	60%	55%	41%	45%	18%	45%
Redis	20%	19%	1%	71%	sesenta y cinco%	29%	36%	11%	35%
MariaDB	17%	14%	3%	59%	53%	41%	47%	4%	47%
Oráculo	dieciséis%	11%	5%	38%	37%	62%	63%	3%	63%
Casandra	dieciséis%	14%	2%	63%	60%	67%	40%	11%	40%
DynamoDB	12%			61%		39%		8%	
Firebase	7%			55%		45%		4%	
Elasticsearch	4%			47%		53%		6%	
Couchbase	2%			37%		63%		2%	

Fig. 13. Raking de las bases de datos más utilizadas según Explore Group
Fuente: (Explore Group, 2019).

- **DB-ENGINES.**

DB-Engines es una iniciativa para recopilar y presentar información sobre sistemas de gestión de bases de datos (DBMS). Además de las bases de datos relacionales, se enfatizan en los sistemas y conceptos del área creciente NoSQL. DB-Engines en su sitio web presentó un ranking de las bases de datos más utilizadas en los últimos años, esta lista se actualiza mensualmente (Db-Engines, 2020).

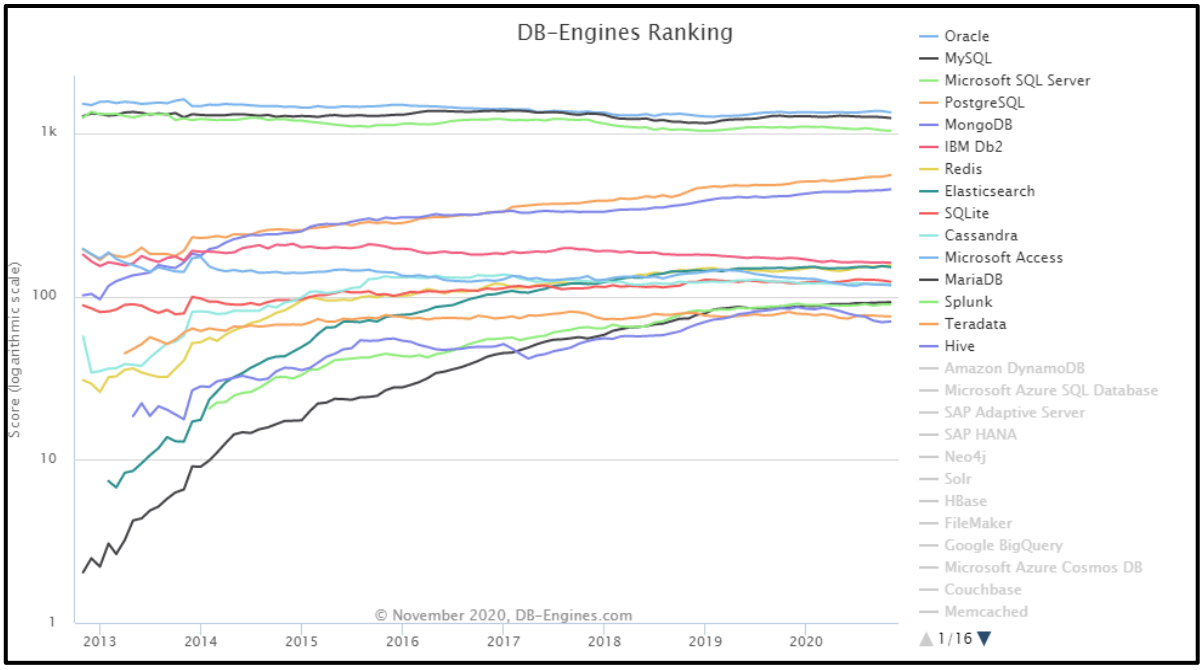


Fig. 14. Raking de las bases de datos más utilizadas según DB-engines.
Fuente: (Db-Engines, 2020)

Para cumplir con los objetivos de este estudio se ha seleccionado seis bases de datos entre relacionales y no relacionales a partir de la investigación realizada acerca del top de las bases de datos más usados, en la siguiente tabla se muestra la posición que ocupa cada una de ellas según la plataforma.

TABLA 2. POSICIÓN QUE OCUPA LAS BASES DE DATOS SQL Y NOSQL

Base de datos.	JetBrains	Explore Group	Db-Engines	Platzi	Modelo de Base de Datos.
MYSQL	1er	1er	2do	1er	Relacional
SQL	4ta	3er	3er	3er	Relacional
PostgreSQL	2do	2do	4to	2do	Relacional
MongoDB	3er	5to	5to	5to	Documentos
Redis	5to	6to	7mo	4to	Key-value
Cassandra	11vo	9no	10mo	-	Key-value

1.4. Experimentos de Ingeniería de Software.

1.4.1. Introducción.

La ingeniería de software es una ciencia de laboratorio por lo que la metodología de investigación de ingeniería de software es muy importante ya que agrega validez científica a los resultados obtenidos en el estudio realizado. Según Li Zhang (2018) menciona que

la elección de la estrategia y el método depende principalmente del enfoque y el objetivo de la investigación. Para este estudio se ha seleccionado la metodología de investigación empírica ya que son el instrumento principal por el cual la comunidad de investigación en ingeniería de software estudia y aprende de la práctica (Stol & Fitzgerald, 2015).

1.4.2. Investigación Empírica.

El enfoque de la investigación empírica ha crecido significativamente en los últimos años que coincide con el surgimiento de la ingeniería de software basada en evidencia (Stol & Fitzgerald, 2015).

El pilar de la investigación empírica es la observación de eventos y la captura de experiencias, por ello es fundamental planificar los pasos para realizar los estudios deseados (Li Zhang et al., 2018). Existen varios puntos de vista de los investigadores acerca de los tipos de investigación para los estudios empíricos entre ellos están; Modelo positivista, Modelo pos-positivista, Modelo interpretativo, Investigación exploratoria, Investigación explicativa, Modelo crítico, Modelo Integral y el que más ajusta para esta investigación es la siguiente:

- **Investigación explicativa**: Se ocupa principalmente de medir una relación o comparar dos o más grupos para identificar una relación de causa y efecto, a menudo se lleva a cabo mediante experimentos controlados detallados a profundidad en la (sección 1.4.5). Este estudio es de diseño fijo, por lo que los elementos se fijan antes de iniciar el estudio. La investigación de diseño fijo también se conoce como investigación cuantitativa, ya que se basa principalmente en datos cuantitativos (Li Zhang et al., 2018).

1.4.3. Métodos de la investigación empírica.

Existen varios criterios que son utilizados por los investigadores para definir el método de investigación para los estudios empíricos. A continuación, se detallan los criterios que más se ajustan a esta investigación:

- **Método inductivo:** Dicho razonamiento permitirá obtener conclusiones generales a partir de hechos particulares, es decir que las conclusiones dependen de los resultados obtenidos.
- **Observación:** La observación es un método que está basado en la recolección de datos cuantitativos que indican regularidades, en este caso los datos recopilados nos indicaran el tiempo de respuesta según la norma ISO/IEC 25023.

- **Medición:** Este es un método de recopilación de información de manera sistemática, válida, confiable e intencional.

1.4.4. Estrategias empíricas en la Ingeniería de software.

Existen varios tipos de estrategias que son utilizadas en la Ingeniería de Software, pero depende de los enfoques de la investigación, la justificación, los métodos, las herramientas y el propósito de la evaluación para dicha investigación. A continuación, se describen los métodos que más se ajusta a esta investigación:

- **Casos de estudio:** Esta metodología es la ideal para la investigación en Ingeniería de software ya que se realiza para investigar un fenómeno en su contexto de la vida real (Li Zhang et al., 2018). Según Wohlin (2012) durante la realización de un estudio de caso, se deben aplicar diversos procedimientos de recopilación de datos y perspectivas de análisis.

  La "unidad de análisis" puede ser algún aspecto de un proyecto de ingeniería de software, en esta investigación el aspecto a evaluar es el tiempo que se demora en devolver una consulta realizada a una API – GraphQL (Li Zhang et al., 2018).

  Los estudios de caso no solo se usan para evaluar cómo o por qué ocurren ciertos fenómenos, sino también para evaluar las diferencias entre dos proyectos o dos metodologías (Wohlin et al., 2012).
- **Experimentos:** Los experimentos se lanzan cuando se desea controlar la situación y manipular el comportamiento de forma directa, precisa y sistemática. Al igual que en los casos de estudio la investigación se ejecuta en el campo en un contexto de la vida real. Un ejemplo podría ser investigar el efecto de un método de inspección con respecto a las fallas encontradas en dos sistemas diferentes, utilizando dos lenguajes de programación diferentes. Entonces, los diferentes sistemas son el contexto para evaluar el método de inspección, por lo tanto, se necesitan objetos similares en el experimento (Wohlin et al., 2012).

1.4.5. Experimentos controlados.

En un experimento controlado el investigador tiene control sobre el estudio. Una de las ventajas de dicho experimento es que el estudio se puede planificar y diseñar de tal manera que se garantice una alta validez. Este tipo de experimentos se realizan para comparar varias técnicas, métodos, procedimientos de trabajo, etc. Por ejemplo sería posible ver un proyecto de desarrollo de software completo como un estudio de caso, pero un experimento típico no incluye todas las actividades de dicho proyecto si no, solo los aspectos antes mencionados (Wohlin et al., 2003).

- **Diseño:** Antes que se realice el experimento, se debe realizar una planificación. Este plan a menudo se denomina diseño del experimento. El objetivo de realizar un experimento es sacar conclusiones que sean válidas para una determinada población de interés (Wohlin et al., 2003). En ocasiones en este tipo de experimentos se declaran dos tipos de variables.
  - **Variables independientes:** Estas variables describen los tratamientos en el experimento (Wohlin et al., 2003). En esta investigación la variable independiente es la API – GraphQL que se conecta las diferentes bases de datos.
  - **Variables dependientes:** Estas variables se estudian para investigar si están influenciadas por las variables independientes (Wohlin et al., 2003). En esta investigación la variable dependiente es la eficiencia con respecto a el tiempo de respuesta de una consulta a la API – GraphQL.

El objetivo del experimento es determinar si las variables dependientes se ven afectadas por las variables independientes (Wohlin et al., 2003), es decir si el tiempo de respuesta se ve afectada por la API – GraphQL que se conecta a una base de datos determinada.

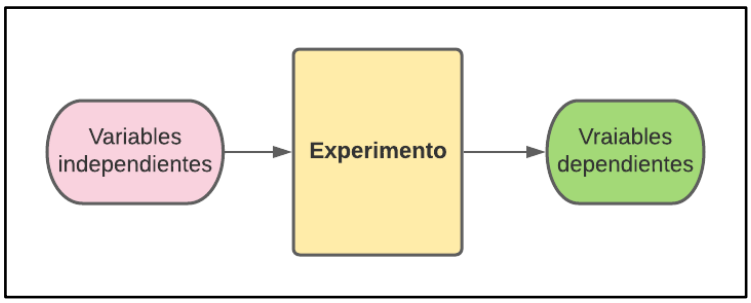


Fig. 15. Variables en un experimento
Fuente: (Wohlin et al., 2003)

- **Operación:** En esta fase se deben incluir tres partes importantes para cumplir con los objetivos del experimento.
  - **Comprometer a los participantes**: Es importante que cada participante esté comprometido con las tareas (Wohlin et al., 2003).
  - **Preparar la instrumentación:** Se debe preparar todo el material que se va a utilizar durante el experimento. La instrumentación debe desarrollarse de acuerdo con el diseño del experimento. (Wohlin et al., 2003).
  - **Ejecución:** Durante esta tarea, los participantes utilizan la instrumentación preparada para recibir instrucciones y registrar datos que se pueden utilizar más adelante en el análisis (Wohlin et al., 2003).

- **Análisis y Resultados:** Antes de realizar un análisis, es importante validar que los datos sean correctos y que las tareas se hayan completado correctamente. La primera parte del análisis consiste normalmente en aplicar estadísticas descriptivas. Esto con el objetivo de obtener una descripción general de los datos y parte de este análisis es identificar los valores que son diferentes al resto de los datos obtenidos. Por último, el objetivo aquí es decidir si existe un efecto del valor de las variables independientes sobre el valor de las variables dependientes (Wohlin et al., 2003).
    - **Interna:** La validez interna se ocupa de los factores que pueden afectar las variables dependientes sin el conocimiento del investigador (Wohlin et al., 2003).
    - **Externa:** La validez externa está relacionada con la capacidad de generalizar los resultados de los experimentos (Wohlin et al., 2003).
    - **Conclusión:** La validez de la conclusión se refiere a la posibilidad de sacar conclusiones correctas con respecto a la relación entre los tratamientos y el resultado de un experimento (Wohlin et al., 2003).
    - **Constructo:** La validez de constructo está relacionada con la relación entre los conceptos y teorías detrás del experimento, de lo que se mide y afecta (Wohlin et al., 2003).

Wholin menciona que una vez que se completa el análisis, el siguiente paso es sacar conclusiones y tomar acciones basadas en las conclusiones (Wohlin et al., 2003).

1.5. Herramientas tecnológicas.

1.5.1. JavaScript.

JavaScript es un lenguaje de programación que utilizan los desarrolladores y rápidamente se ha vuelto en un lenguaje dominante para crear aplicaciones web interactivas. JavaScript es un lenguaje usado cada vez más para aligerar la funcionalidad principal al lado del cliente e implementar aplicaciones en tiempo real en el lado del servidor a través de entornos como Node.js (Mesbah, 2015).

1.5.2. Node JS.

NodeJS es uno de los entornos de desarrollo más interesantes y que ha ganado popularidad en el espacio de JavaScript del lado del servidor, este se centra en el rendimiento y poco consumo de memoria (Tilkov & Vinoski, 2010).

1.5.3. GraphiQL

GraphiQL es el entorno de desarrollo integrado (IDE) de GraphQL, este proporciona soporte de primera clase para gráficos y cuenta con un explorador que aplica técnicas de optimización de consultas para ajustar el rendimiento ya permite construir consultas completas de forma interactiva (Jindal & Madden, 2014).

Fig. 16. Ejemplo de GraphiQL

1.5.4. Git y GitHub.

Git es un sistema de control de versiones distribuido que ayuda a los desarrolladores a tener un mejor control de sus proyectos. GitHub está diseñado para la colaboración de proyectos que se alojan en su portal.

1.5.5. ReactJS.

React (también llamado React.js o ReactJS) es una biblioteca de JavaScript de código abierto con un enfoque en la creación de interfaces de usuario (frontend) en páginas web. Lo mantienen Facebook, Instagram, otras empresas, así como una comunidad de desarrolladores individuales (Sousa, 2020).

1.5.6. Express GraphQL.

La forma más sencilla de ejecutar un servidor API GraphQL es utilizar Express, un marco de aplicación web popular para Node.js (GraphQL, 2020).

1.6. Métricas de la norma ISO/IEC 25023.

Esta norma internacional se utiliza para medir y evaluar la calidad del sistema o producto de software. La normativa expuesta en esta sección pertenece a un conjunto de Normas Internacionales de la serie SQuaRE (Requisitos y evaluación de la calidad del software y del sistema) ISO/IEC 2502n en la que constan las siguientes normas:

- **ISO/IEC 25020** – Guía y modelo de referencia de medición.
- **ISO/IEC 25021** – Elementos de medida de la calidad.
- **ISO/IEC 25022** – Medición de la calidad en el uso.
- **ISO/IEC 25023** – Medición de la calidad del sistema y producto de software.
- **ISO/IEC 25024** – Medición de la calidad de datos.

El siguiente cuadro es la representación de la relación entre las normas de la serie ISO/IEC 2502n - División de Medición de Calidad.

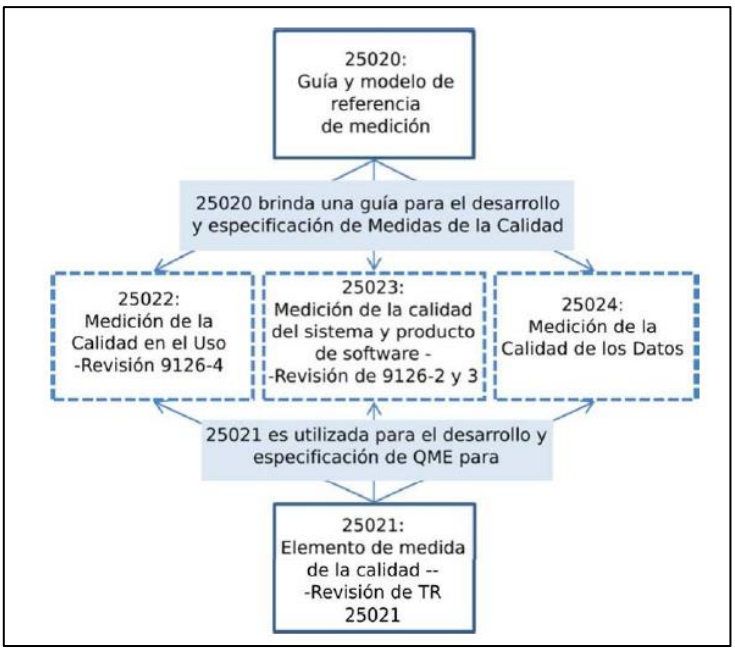


Fig. 17. Estructura de la División de la Medición de la Calidad
Fuente: (ISO 25023, 2020)

Esta norma incluye:

a) Un conjunto básico de medidas de la calidad para cada una de las características y subcaracterísticas.
b) Una explicación sobre cómo aplicar las medidas de la calidad de sistemas y productos de software.

1.6.1. Medidas de la calidad del sistema y producto de software.

La calidad de un sistema o producto de software se mide en el nivel que satisface las necesidades. Estas necesidades están representadas en la serie de Normas SQuaRE (Requisitos y evaluación de la calidad del software y del sistema) que organizan la calidad de un sistema o producto de software en características y subcaracterísticas (ISO 25023, 2020).

Las medidas de la calidad pueden utilizarse con diferentes técnicas de evaluación que pueden seleccionarse de acuerdo con la característica de la calidad, dependiendo de si se utiliza como medida interna o externa (ISO 25023, 2020).

Para esta investigación se seleccionó la medida del tiempo de respuesta que está dentro de las medidas de eficiencia del desempeño.

1.6.2. Medidas de eficiencia del desempeño.

Esta medida de la eficiencia del desempeño se utiliza para evaluar el desempeño referente a la cantidad de recursos utilizados bajo condiciones establecidas. Los recursos pueden incluir otros productos de software, como, por ejemplo; la configuración de software y hardware del sistema y los materiales (medios de almacenamiento) (ISO 25023, 2020).

- **Medidas de comportamiento del tiempo.**

Las medidas de comportamiento del tiempo se utilizan para evaluar si el tiempo de respuesta y el procesamiento del sistema cumplen con su función al realizar tareas específicas (ISO 25023, 2020).

La TABLA 3. muestra a detalle las medidas del comportamiento del tempo.

TABLA 3. MEDIDAS DE COMPORTAMIENTO DEL TIEMPO ISO/IEC 25023
FUENTE: (ISO 25023, 2020)**.**

ID	Nombre	Descripción	Función de medición
PTb-3-G	Tiempo de respuesta	¿Cuál es el tiempo empleado para completar trabajo o un proceso asíncrono?	$X = B - A$ A= Tiempo de envío de petición B = Tiempo en recibir la primera respuesta

CAPÍTULO 2

DESARROLLO

En esta fase se desarrolló un laboratorio experimental mediante dos aplicaciones (Backend y Frontend) usando el marco de trabajo para desarrollo ágil “SCRUM”. La primera aplicación tiene la función de brindar servicios de una API GraphQL; trabajando con bases de datos SQL y bases de datos NoSQL. La segunda aplicación es un cliente que consume y ejecuta los casos de uso desarrollados en el API GraphQL, establecidos en el diseño de experimento de software.

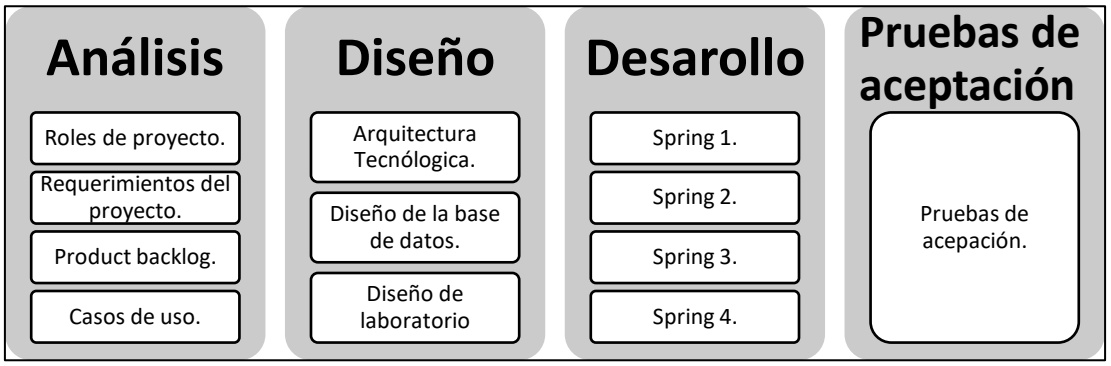


Fig. 18. Estructura de la fase de desarrollo del proyecto.

2.1. Análisis.

2.1.1. Roles de Proyecto.

Para el desarrollo de la API GraphQL, el cliente y la prueba de aceptación se ha realizado un grupo de trabajo según “SCRUM”, en la TABLA 4. se detalla a profundidad sobre los integrantes del proyecto y sus respectivos roles.

TABLA 4 ROLES DE PROYECTO.

Miembro	Descripción	Rol
Ing. Antonio Quiña	Director del presente Trabajo de Grado y Docente de la Carrera de Ingeniería en Sistemas Computacionales de la Universidad Técnica del Norte.	Propietario del Producto (Product Owner).
Luis Andrés Quinche Moran	Estudiante de la Carrera de Ingeniería en Sistemas Computacionales de la Universidad Técnica del Norte.	Jefe del Proyecto (Scrum Master).

Luis Andrés Quinche Moran	Estudiante de la Carrera de Ingeniería en Sistemas Computacionales de la Universidad Técnica del Norte.	Equipo de Desarrollo (Development Team).

2.1.2. Requisitos del proyecto.

- **Historias de Usuarios**

Las historias de usuario son el instrumento fundamental en la metodología de SCRUM, los cuales se levantaron por el Product Owner con requisitos para el desarrollo de la API GraphQL, el cliente y las pruebas de concepto.

Las siguientes TABLAS muestran a detalle todas las historias de usuario levantadas por el Product Owner.

TABLA 5 HISTORIA DE USUARIO 1

Historia de Usuario	**N:** H-U-1
Nombre: Configuración del entorno de desarrollo	**Usuario: -**
Dependencia: Ninguna	**Estimación:** 12 h
Descripción: Como desarrollador requiero configurar el entorno de trabajo que se va a utilizar para desarrollar el proyecto como la instalación de softwares necesarios.	

TABLA 6 HISTORIA DE USUARIO 2

Historia de Usuario	**N:** H-U-2
Nombre: Desarrollo del API GraphQL (SQL)	**Usuario:** Desarrollador Backend
Dependencia: H-U-1	**Estimación: 3**0 h
Descripción: Como desarrollador backend requiero un servicio API que me permita consumir datos de 3 base de datos SQL, el diseño de la base de datos será la que se ha creado anteriormente.	
Pruebas de aceptación: • La API permitirá listar los datos consultados. • La API permitirá consultar los datos por ID. • La API permitirá crear, eliminar y editar los datos. • La API GraphQL tendrá su documentación que servirá como guía para la correcta utilización.	

TABLA 7 HISTORIA DE USUARIO 3

Historia de Usuario	**N:** H-U-3
Nombre: Desarrollo del API GraphQL (NoSQL)	**Usuario:** Desarrollador Backend
Dependencia: H-U-1	**Estimación:** 35 h
Descripción: Como desarrollador backend requiero un servicio API que me permita consumir datos de 3 base de datos NoSQL, el diseño de la base de datos será la que se ha creado anteriormente.	
Pruebas de aceptación: • La API permitirá enlistar los datos consultados. • La API permitirá consultar los datos por ID. • La API permitirá crear, eliminar y editar los datos. • Integración de todas las APIs realizadas en una sola. • La API GraphQL tendrá su documentación que servirá como guía para la correcta utilización.	

TABLA 8 HISTORIA DE USUARIO 4

Historia de Usuario	**N:** H-U-**4**
Nombre: Desarrollo de un Cliente	**Usuario:** Desarrollador Frontend
Dependencia: H-U-3	**Estimación:** 20 h
Descripción: Como desarrollador frontend requiero un cliente sencillo realizado en ReactJS para consumir datos desde la API - GraphQL.	

2.1.3. Product Backlog.

La siguiente tabla contiene a detalle las historias de usuario que se definieron por el Product Owner.

TABLA 9 PRODUCT BACKLOG.

Orden	ID HU	Descripción	ROL	Estimación
1	H-U-1	Configuración del entorno de desarrollo.	-	14 h
2	H-U-2	Desarrollo del API GraphQL	Desarrollador Backend	20 h
3	H-U-3	Desarrollo del API GraphQL	Desarrollador Backend	20h
4	H-U-4	Desarrollo de un Cliente	Desarrollador Frontend	20h

2.1.4. Casos de uso.

Los siguientes casos de uso son parte del diseño del experimento de laboratorio que se va a realizar para medir la eficiencia del consumo de datos desde la API - GraphQL desarrollada que estará conectada a tres bases de datos relacionales y tres no relacionales. Esto servirá de guía para ejecutar las tareas planeadas.

TABLA 10 CASO DE USO 1

Caso de uso:	C-U-1
Nombre de caso de uso:	Consulta de Usuarios
Creado por:	Andrés Quinche
Actores:	Investigador
Descripción:	Se realiza una consulta a la tabla independiente llamada **usuarios**, contenida en las bases de datos relacionales y no relacionales.
Precondiciones:	La base de datos tiene 4 tablas de datos todas estas con registros pre-cargados.
Flujo Normal:	1.- El investigador realiza la consulta una tabla independiente un nivel. Se hará 4 consultas; • Consulta de 1 Usuario. • Consulta de 100 Usuarios. • Consulta de 1000 usuarios. • Consulta de 50 000 Usuarios. 2.- El investigador accionara la consulta según el número de usuarios seleccionados. 3.- El sistema verifica la efectividad de la consulta realizada. 4.- El sistema hará la consulta del número de usuarios seleccionados, de la tabla **usuarios** de las bases de datos relacionales y no relacionales. 5.- El sistema imprime el tiempo en el que se efectúa la consulta.
Flujo Alternativo:	4.A.- Si los datos consultados no son correctos, el sistema muestra un mensaje al investigador para informar del error, para así repetir la consulta.
Postcondiciones	Se realiza el registro del Tiempo de respuesta.

La estructura de la consulta de Usuarios correspondiente al CASO DE USO 1 se muestra en la Fig. 19.

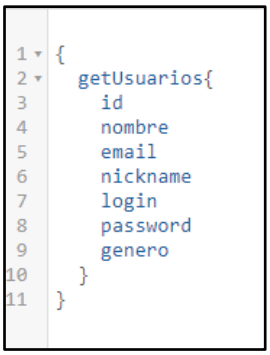


Fig. 19. Estructura de una consulta a GraphQL de los Usuarios (1 nivel)

TABLA 11 CASO DE USO 2

Caso de uso:	C-U-2
Nombre de caso de uso:	Consulta de Posts
Creado por:	Andrés Quinche
Actores:	Investigador
Descripción:	Se realiza una consulta a una tabla con relaciones a dos niveles llamadas **posts** y **usuarios**, contenida en las bases de datos relacionales y no relacionales.
Precondiciones:	La base de datos tiene 4 tablas de datos todas estas con registros pre-cargados.
Flujo Normal:	1.- El investigador realiza la consulta se realizará a una tabla con relaciones a tablas a dos niveles. Se hará 4 consultas; • Consulta de 1 Post y usuario. • Consulta de 100 Posts y usuarios. • Consulta de 1000 Posts y usuarios. • Consulta de 50 000 Posts y usuarios. 2.- El investigador accionara la consulta según el número de posts seleccionados. 3.- El sistema verifica la efectividad de la consulta realizada. 4.- El sistema hará la consulta del número de **posts** seleccionados, de las tablas **posts** y **usuarios** de las bases de datos relacionales y no relacionales. 5.- El sistema imprime el tiempo en el que se efectúa la consulta.
Flujo Alternativo:	4.A.- Si los datos consultados no son correctos, el sistema muestra un mensaje al investigador para informar del error, para así repetir la consulta.
Postcondiciones	Se realiza el registro del Tiempo de respuesta.

La estructura de la consulta de POSTS correspondiente al CASO DE USO 2 se muestra en la Fig. 20.

```
{
  getPosts{
    id
    contenido
    titulo
    fecha_publicacion
    keywords
    estado
    categoriasid{
      id
      nombre_categoria
    }
    usuariosid{
      id
      nombre
      email
      nickname
      login
      password
      genero
    }
  }
}
```

Fig. 20. Estructura de una consulta a GraphQL de los Post (2 niveles)

TABLA 12 CASO DE USO 3

Caso de uso:	C-U-3
Nombre de caso de uso:	Consulta de Post y Comentarios
Creado por:	Andrés Quinche
Actores:	Investigador
Descripción:	Se realiza una consulta a una tabla con relaciones a dos niveles llamadas **posts, usuarios** y **comentarios**, contenida en las bases de datos relacionales y no relacionales.
Precondiciones:	La base de datos tiene 4 tablas de datos todas estas con registros pre-cargados.
Flujo Normal:	1.- El investigador realiza la consulta se realizará a una tabla con relaciones a tres niveles. Se hará 4 iteraciones; • Consulta de 1 posts, usuarios y comentarios. • Consulta de 100 posts usuarios y comentarios. • Consulta de 1000 posts, usuarios y comentarios. • Consulta de 50 000 posts, usuarios y comentarios.

	2.- El investigador accionara la consulta según el número de usuarios seleccionados. 3.- El sistema verifica la efectividad de la consulta realizada. 4.- El sistema hará la consulta del número de **post** seleccionados, de la tabla **posts**, **usuarios** y **comentarios** de las bases de datos relacionales y no relacionales. 5.- El sistema imprime el tiempo en el que se efectúa la consulta.
Flujo Alternativo:	4.A.- Si los datos consultados no son correctos, el sistema muestra un mensaje al investigador para informar del error, para así repetir la consulta.
Postcondiciones	Se realiza el registro del Tiempo de respuesta.

La estructura de la consulta de POSTS y sus COMENTARIOS correspondiente al CASO DE USO 3 se muestra en la Fig. 21.

```
{
  getPosts{
    id
    titulo
    fecha_publicacion
    contenido
    keywords
    estado
    usuariosid{
      id
      nombre
      email
      nickname
      login
      password
      genero
    }
    categoriasid{
      id
      nombre_categoria
    }
        comentarios{
          id
          contenido
                usuariosid{
                id
                nombre
                email
                nickname
                login
                password
                genero
              }
        }
  }
}
```

Fig. 21. Estructura de una consulta a GraphQL de los Post y sus Comentarios (3 niveles)

2.2. Diseño.

El desarrollo de la API GraphQL está distribuido en Sprints, el Sprint denominado Sprint 0 es donde se definió la arquitectura tecnológica y el diseño de la base de datos.

TABLA 13 TABLA DEL SPRINT 0

Sprint	Fecha inicio	Fecha fin	Duración
Sprint 0	26-oct-2020	10-nov-2020	20

- **Planificación del Sprint 0.**

Fecha: 26/10/2020

Objetivo: Definir la arquitectura tecnológica y la base de datos del proyecto.

Miembros: Product Owner, Scrum Master y Equipo de desarrollo.

TABLA 14 PLANIFICACIÓN DEL SPRINT 0.

Fase de Desarrollo	Tarea	Tiempo
Diseño	Definir la arquitectura tecnológica del proyecto	5
Diseño	Definir el modelo de la base de datos.	4
Planificación	Detallar las tareas a realizar en el Sprint 0	4
Revisión	Revisar los resultados del desarrollo del Sprint 0	4
Retrospectiva	Analizar los resultados obtenidos del sprint	1
	TOTAL DE HORAS	**18**

2.2.1. Arquitectura Tecnológica.

Como se mencionó anteriormente esta fase consta de dos proyectos, un servidor que está desarrollado en Node.js con la implementación de tecnologías como, GraphQL, Express, las bases de datos SQL y NoSQL utilizadas son: MySQL, SQL, PostgreSQL, MongoDB, Cassandra, Redis. El cliente esta desarrollado en JavaScript haciendo uso de tecnologías como Apollo Client, este cliente es solo para evidenciar el buen funcionamiento del API - GraphQL.

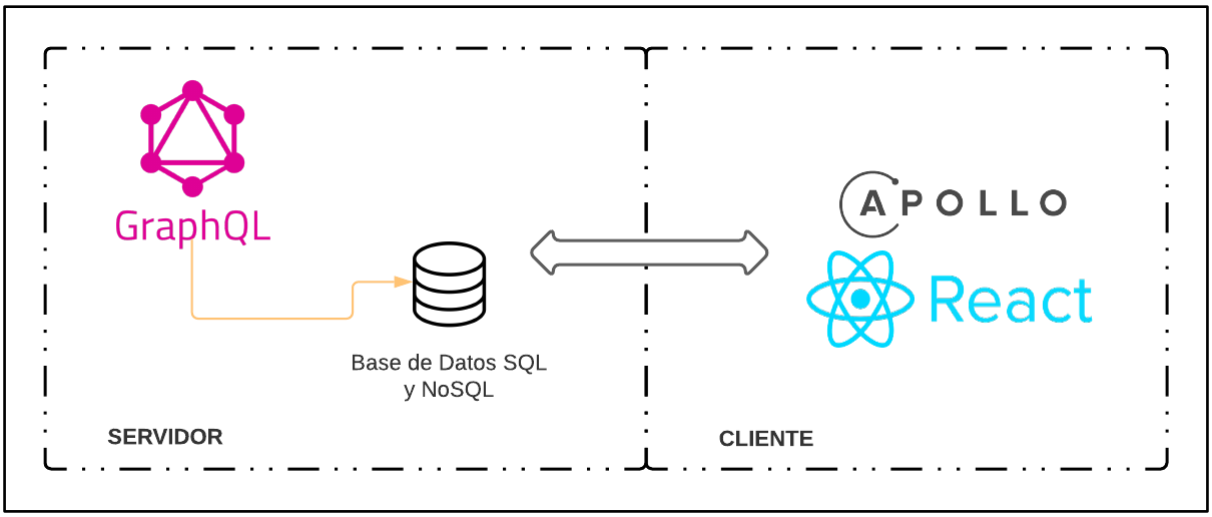


Fig. 22. Arquitectura tecnológica.

2.2.2. Diseño de la Base de Datos.

El diseño de la base de datos se desarrolla en tres fases:

- Diseño Conceptual.
- Diseño Lógico.
- Diseño Físico.

Todo este proceso comienza una vez finalizado la fase de recopilar y analizar los requerimientos de la base de datos (Quintana Rondón, Camejo Domínguez, & Díaz Berenguer, 2011).

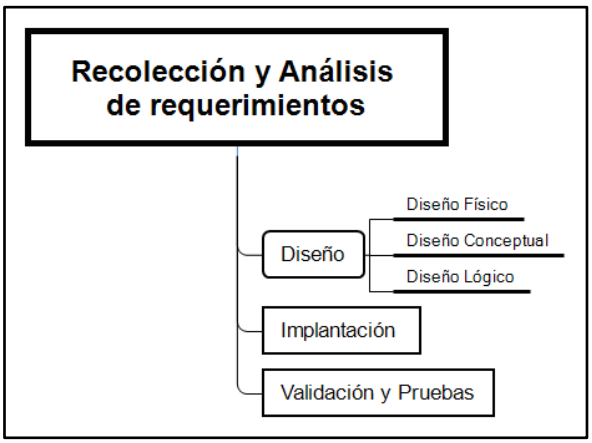


Fig. 23. Esquema para el Diseño de una base de Datos
Fuente: (Quintana Rondón et al., 2011).

En la presente investigación se seleccionó una base de datos de un blog que tiene como entidades; USUARIOS, POSTS, COMENTARIOS, CATEGORÍAS.

- **Diseño Conceptual de la Base de Datos.**

El esquema conceptual tiene como fin la comprensión de la estructura, las relaciones y restricciones de la base de datos, lograr la comunicación entre usuarios y analistas para dar paso al diseño lógico de la base de datos (Quintana Rondón et al., 2011).

- **Diseño Lógico de la Base de Datos.**

El esquema lógico es el proceso en el que se transforma un esquema conceptual en un esquema lógico con las estructuras de datos del modelo de base de datos en el que se basa el (SGBD) que se vaya a utilizar (Quintana Rondón et al., 2011).

- **Diseño Físico de la Base de Datos.**

El esquema físico es el proceso en el que se realiza la implementación de la base de datos, las estructuras de almacenamiento y los métodos para acceder a la información.

Para dar comienzo a esta etapa, se debe haber decidido cuál es el (SGBD) que se va a utilizar, ya que el esquema físico se adapta a este (Quintana Rondón et al., 2011).

La Fig. 24 muestra el esquema base de datos que será utilizada en esta investigación.

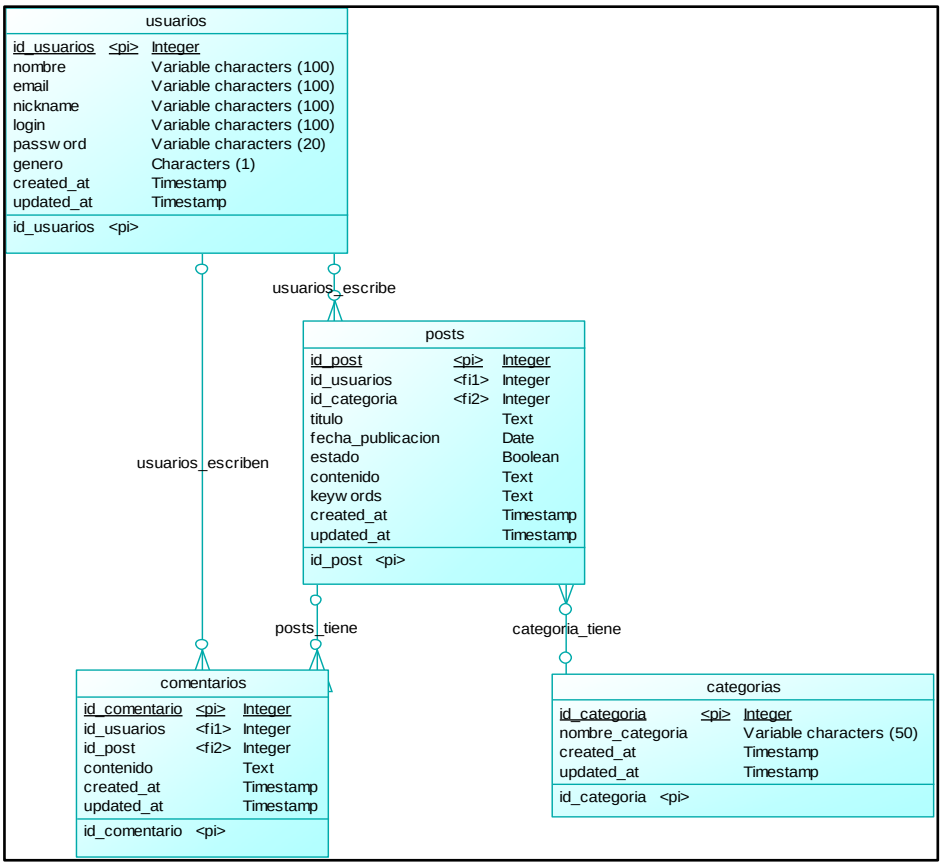


Fig. 24 Esquema Físico de una Base de Datos de un Blog.
DBMS(MYSQL)Realizado en Power Design.

2.3. Desarrollo

En esta fase de desarrollo se realizó en Sprints donde cada Sprint tuvo una duración de 20 horas organizadas en dos semanas a excepción del Sprint 3 que se realizó en 3 semanas con una duración de 35 horas. La TABLA 15. muestra el resumen de los Sprints del proyecto.

TABLA 15 DETALLE GENERAL DE LOS SPRINTS

N·Sprint	Fecha Inicio	Fecha Fin	Duración
1	16-nov-2020	30-nov-2020	20
2	1-dic-2020	15-dic-2020	20
3	16-dic-2020	7-ene-2021	35
4	7-ene-2021	21-ene-2021	20

2.3.1. Sprint 1

- **Planificación del Sprint 1**

  - **Reunión de planificación**

Fecha: 16 de noviembre del 2020

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

Objetivo: Planificación del Sprint 1 (Creación del Sprint backlog)

  - **Sprint Backlog.**

La TABLA 16, muestra las tareas que se van a realizar en el Sprint 1 dedicado a preparar el entorno de trabajo para cumplir con los objetivos de la investigación.

TABLA 16 SPRINT BACKLOG 1

Historias de usuario	Fase de desarrollo	Tarea	Estimación
H-U-1	Instalación	Instalar base de datos MYSQL	2
	Instalación	Instalar base de datos PostgreSQL	2
	Instalación	Instalar base de datos MS SQL	2
	Instalación	Instalar base de datos Redis	2
	Instalación	Instalar base de datos Cassandra	2
	Instalación	Instalar base de datos Mongo DB	2

- **Revisión del Sprint 1**

  - **Reunión de la revisión**

Fecha: 23 de noviembre del 2020

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

Objetivo: Revisión de la instalación de las Bases de Datos Instaladas.

  - **Incremento del producto potencialmente entregable**

A continuación, se muestra la evidencia de las diferentes Bases de datos instaladas, tanto SQL y NoSQL.

1. MYSQL.

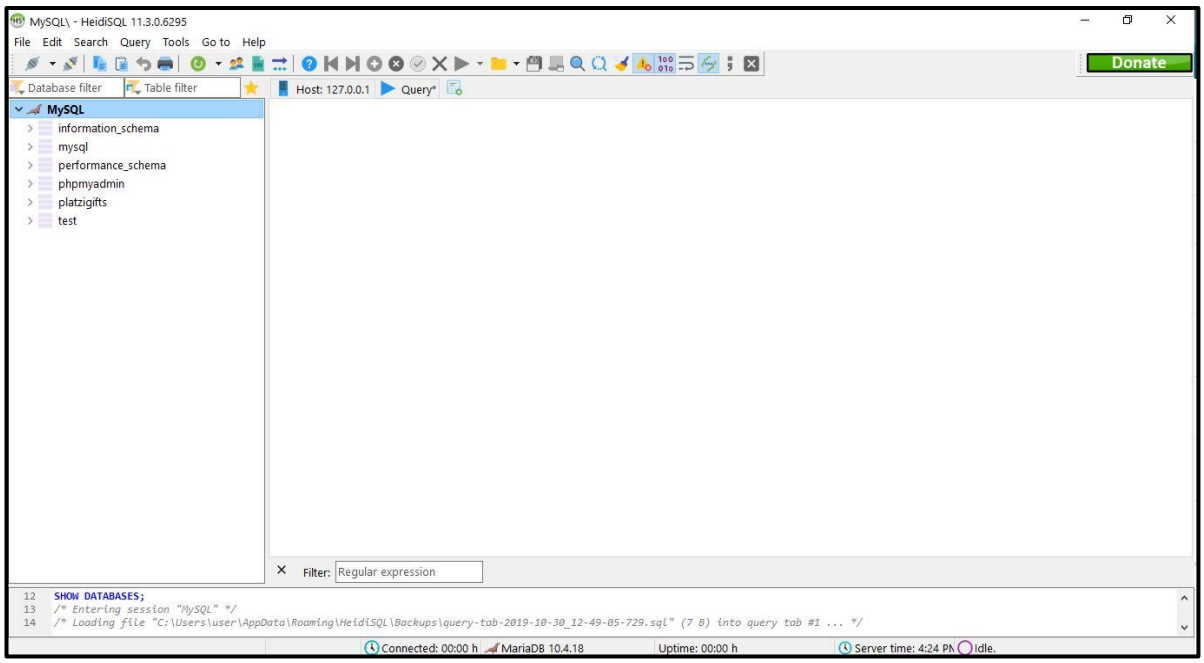


Fig. 25 Interfaz Gráfica del SGBD de la base de datos MYSQL.

2. POSTGRESQL.

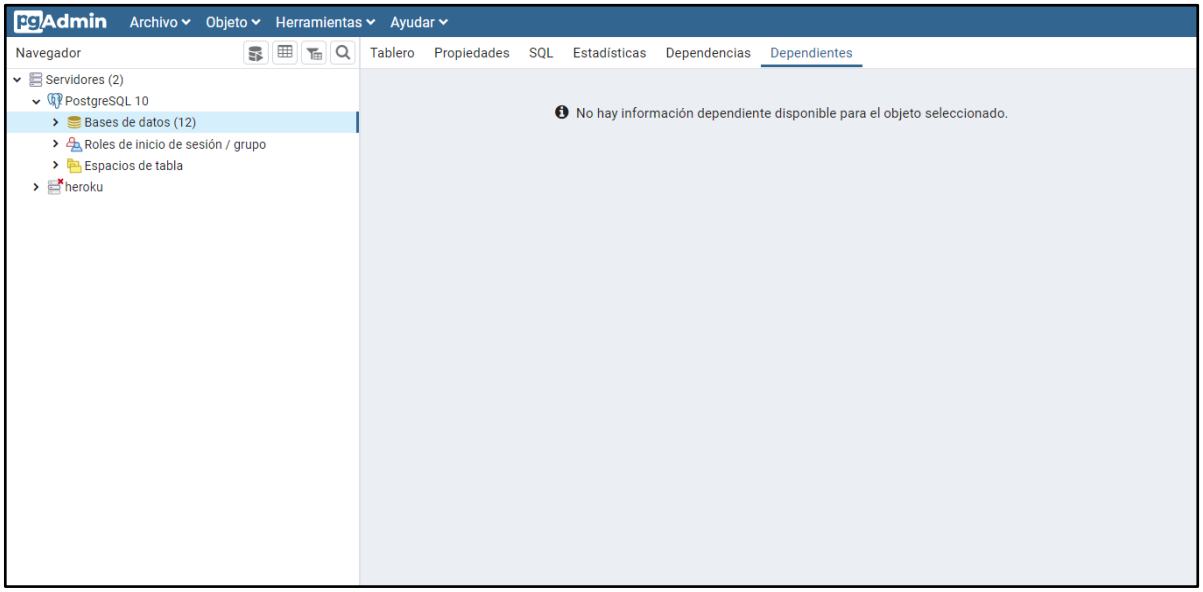


Fig. 26 Interfaz Gráfica del SGBD de la base de datos POSTGRESQL.

3. MSSQL.

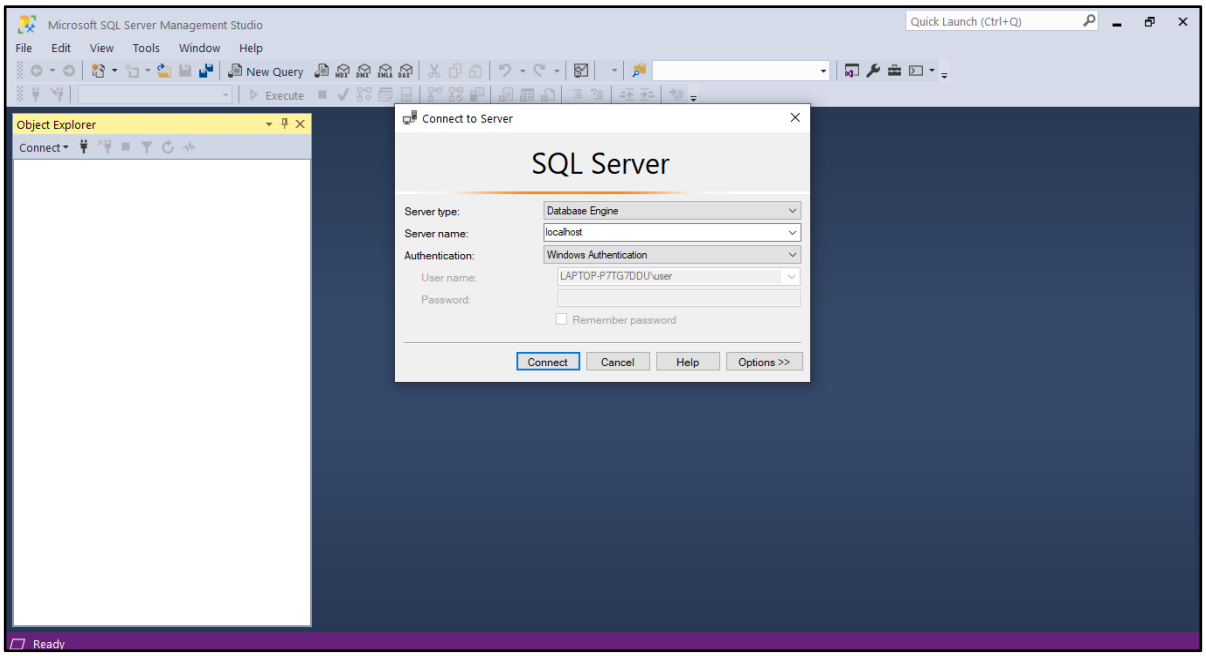


Fig. 27 Interfaz Gráfica del SGBD de la base de datos MSSQL.

4. MONGODB.

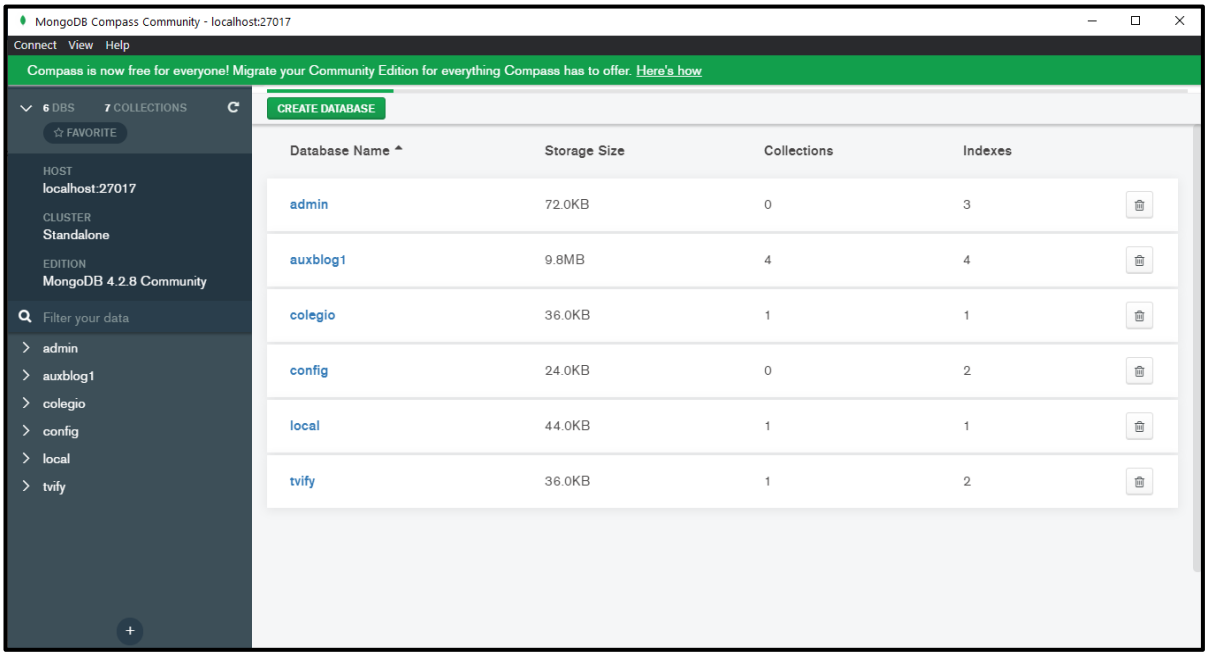


Fig. 28 Interfaz Gráfica del SGBD de la base de datos MongoDB.

5. REDIS.

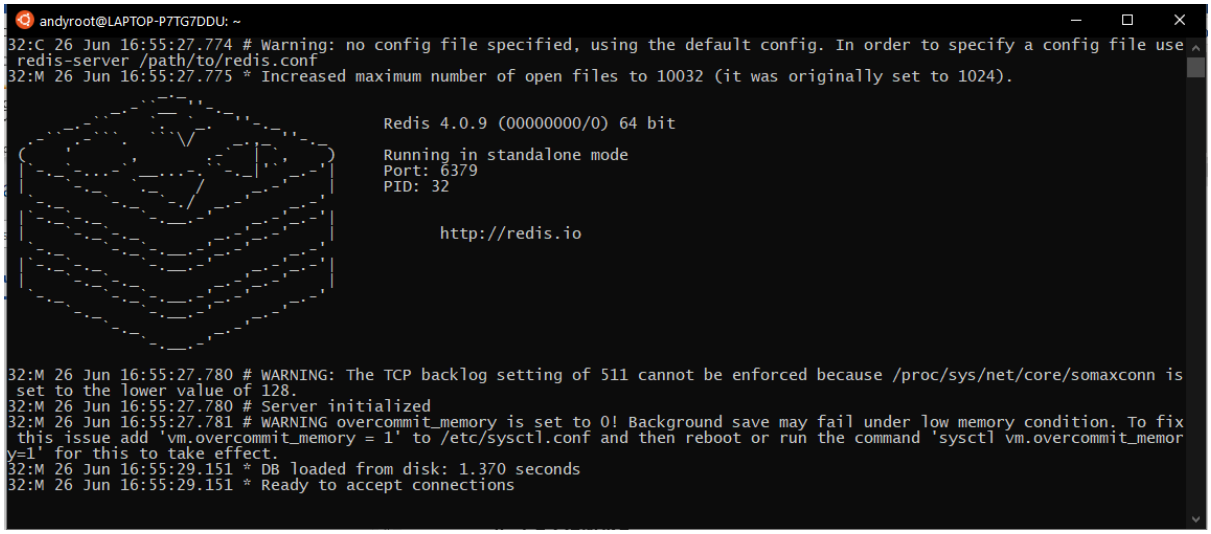


Fig. 29 Terminal de la base de datos Redis.

6. CASSANDRA.

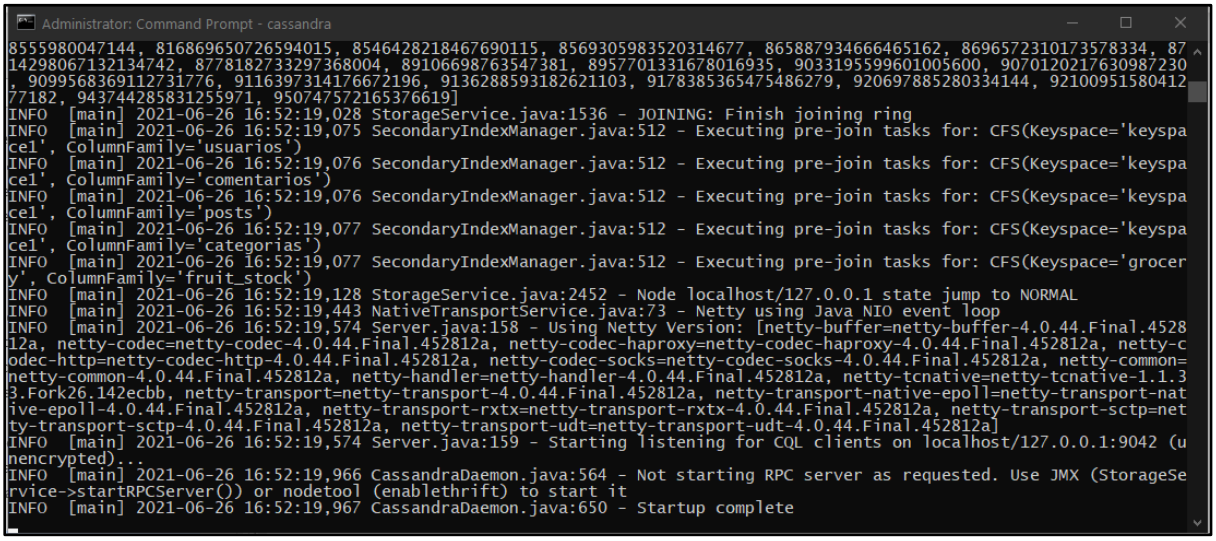


Fig. 30 Terminal de la base de datos Cassandra.

- **Retrospectiva del Sprint 1**

Fecha: 27 de noviembre del 2020

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

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

TABLA 17 RETROSPECTIVA SPRINT 1

Aciertos (¿Qué salió bien del Sprint?)	Rapidez en la instalación de las herramientas para el entorno de trabajo.

Errores (¿Qué no salió bien del Sprint?)	Problemas con los gestores de base de datos.
Mejoras (¿Qué mejoras se implementará?)	Optimización del tiempo.

2.3.2. Sprint 2

- **Planificación del Sprint 2**
  - **Reunión de planificación**

Fecha: 1 de diciembre del 2020

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

Objetivo: Planificación del Sprint 2 (Creación del Sprint backlog)

  - **Sprint Backlog**

La TABLA 18, muestra las tareas que se realizaran en el Sprint 2 dedicado al desarrollo de la API GraphQL conectada a las bases de datos SQL.

TABLA 18 SPRINT BACKLOG 2

Historias de usuario	Fase de desarrollo	Tarea	Estimación
H-U-2	Desarrollo	Crear el proyecto en el entorno de desarrollo NodeJS, instalar las dependencias que se necesitan para el servidor GraphQL.	3
	Desarrollo	Implementar el servidor; crear el esquema, los queries, los mutations y los resolvers.	10
	Pruebas	Pruebas de funcionamiento del servidor.	2
Eventos	Planificación	Detallar las tareas realizadas en el Sprint actual	2
	Revisión	Revisar los resultados del desarrollo del Sprint	2
	Retrospectiva	Analizar los resultados del Sprint	1
		TOTAL	20

- **Revisión del Sprint 2**
  - **Reunión de revisión**

Fecha: 1 de diciembre del 2020

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

Objetivo: Revisión del desarrollo del incremento del producto.

- **Pruebas de Aceptación.**

En la TABLA 19, se muestra el cumplimiento de los requisitos definidos en el Sprint 2.

TABLA 19 PRUEBAS DE ACEPTACIÓN SPRINT 1.

HISTORIAS DE USUARIO	Nombre	Funcionalidad	Aceptación	
			SI	NO
H-U-2	**Desarrollo de API GraphQL (SQL)**	La API permitirá listar los datos consultados.	**X**	
		La API permitirá consultar los datos por ID.	**X**	
		La API permitirá crear, eliminar y editar los datos.	**X**	
		La API tendrá su documentación que servirá como guía para la correcta utilización.	**X**	

- **Incremento del producto potencialmente entregable**

A continuación, se muestra la evidencia de la Api – GraphQL terminada, cabe recalcar que para este Sprint la API solo se conecta a Bases de datos SQL.

Listar Usuarios.

La Fig. 31 es un ejemplo de cómo se realiza una consulta para obtener un listado personalizado, en este caso se va a obtener un listado de todos los usuarios con los datos que se requiera y como resultado envía un objeto JSON.

Fig. 31. Respuesta a una consulta GraphQL para listar todos los usuarios.

Listar un Usuario por id.

La Fig. 32, muestra cómo realizar una consulta para obtener un usuario especifico enviando un id.

Fig. 32. Respuesta a una consulta GraphQL para obtener un Usuario por id.

Crear Usuarios

La Fig. 33**,** indica como crear un Usuario en la Api-GraphQL desarrollada, en este ejemplo se puede apreciar que para crear un Usuario es necesario ingresar ciertos datos y su estructura es muy similar a JSON.

Fig. 33. Crear un Usuario en GraphQL.

Editar un Usuario

La Fig. 34, indica como editar un usuario en la Api-GraphQL desarrollada, en este ejemplo se puede apreciar que para editar el usuario es necesario ingresar ciertos datos; entre ellos el id del usuario a editar y el objeto de los datos editados.

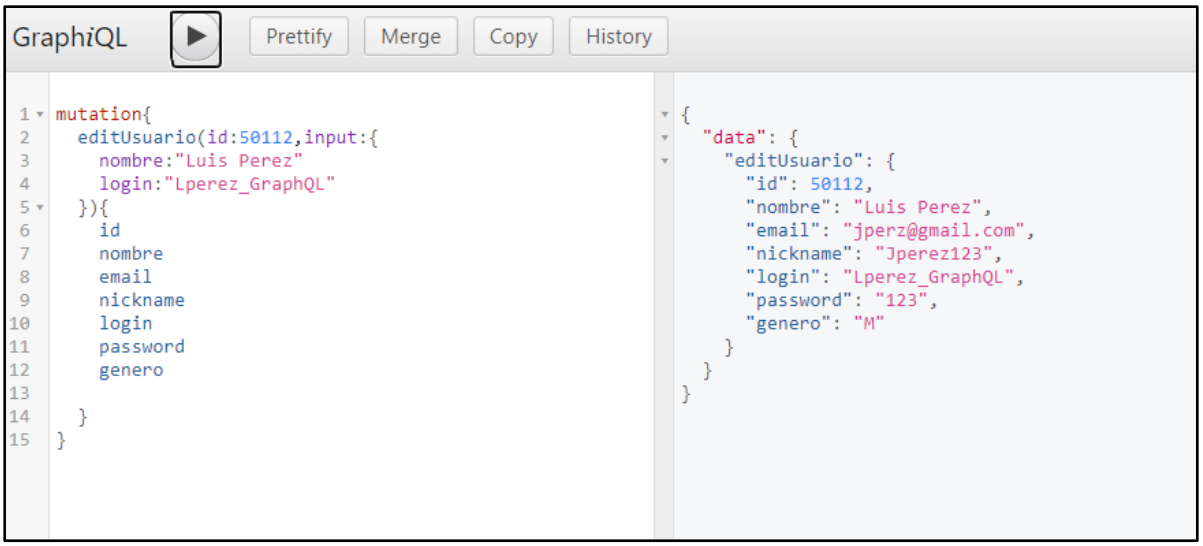


Fig. 34. Editar un Usuario en GraphQL

Eliminar Usuario

La Fig. 35, indica como eliminar un usuario en la API – GraphQL desarrollada, en este ejemplo se puede apreciar que es necesario ingresar el id del usuario que se desea eliminar.

Fig. 35. Eliminar un Usuario en GraphQL.

Listar Categorías

La Fig. 36 es un ejemplo de cómo realizar una consulta de todas las categorías.

Fig. 36. Respuesta a una consulta GraphQL para obtener las todas las categorías.

Listar una Categoría por id.

La Fig. 37 es un ejemplo de cómo realizar una consulta de una categoría.

Fig. 37. Respuesta a una consulta GraphQL para obtener una categoría por id.

Crear Categoría.

La Fig. 38, muestra cómo crear una categoría.

Fig. 38. Crear una categoría en GraphQL.

Editar categoría

La Fig. 39, muestra como editar una categoría.

Fig. 39. Editar una categoría en GraphQL.

Eliminar categoría

La Fig. 40, indica como eliminar una categoría.

Fig. 40. Eliminar una categoría en GraphQL.

Las siguientes entidades son ligeramente diferentes a las dos anteriores (comentarios y usuarios) debido a que los comentarios y los posts tienen claves foráneas.

Listar Comentarios.

La Fig. 41, muestra cómo realizar una consulta que tiene claves foráneas para obtener todos los comentarios, en esta ocasión los comentarios tienen dos relaciones; los usuarios y los posts.

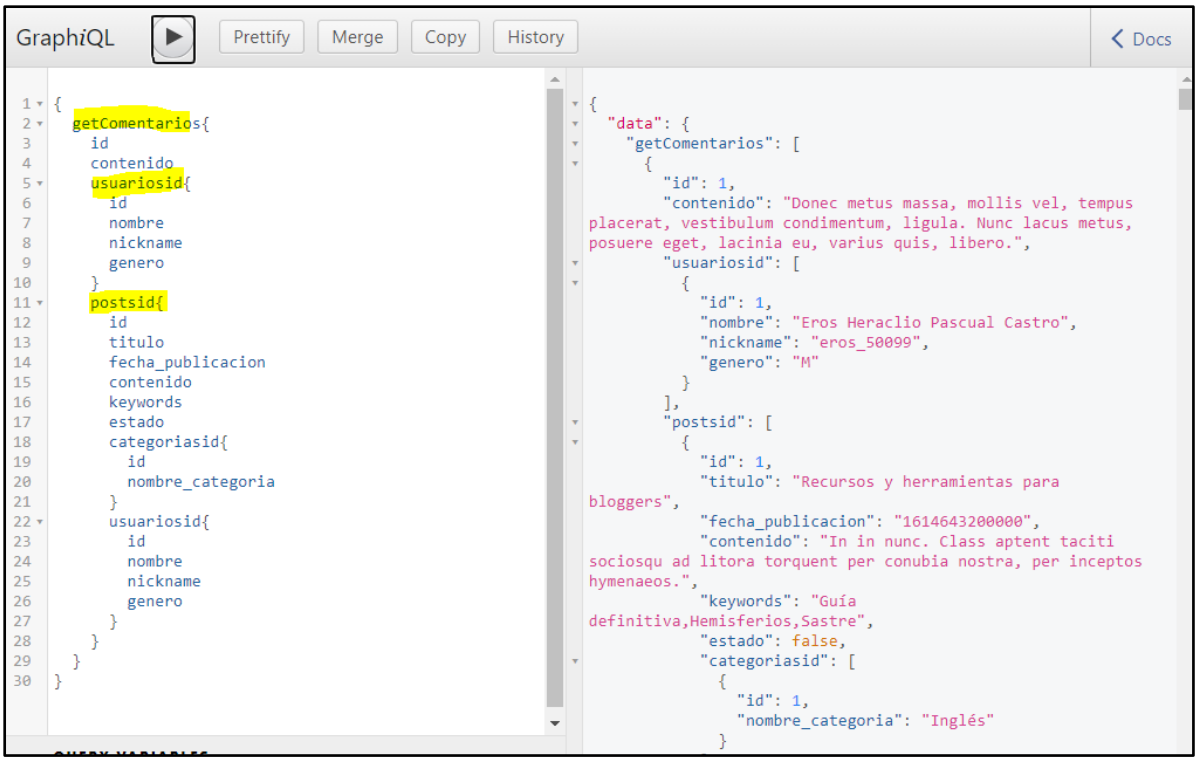


Fig. 41. Respuesta de una consulta GraphQL para obtener todos los Comentarios.

Listar un Comentario por id.

La Fig. 42, es un ejemplo de cómo realizar una consulta de un comentario por id.

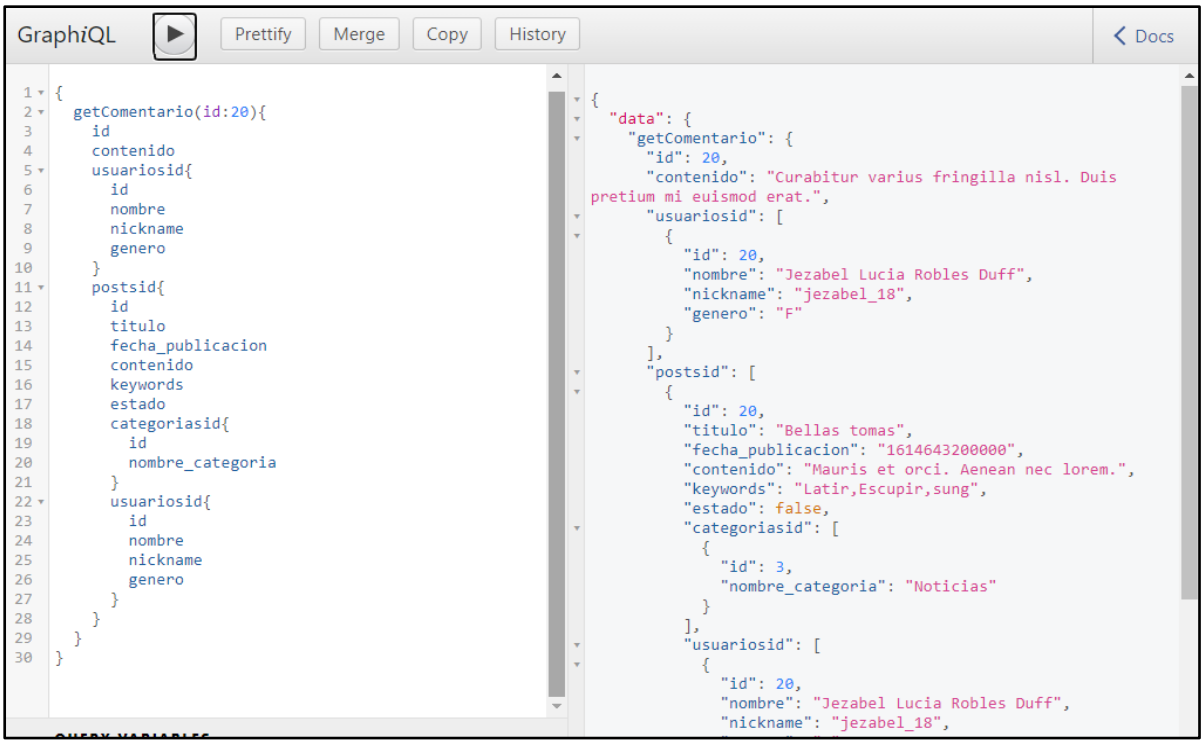


Fig. 42. Respuesta de una consulta GraphQL para obtener un Comentario por id.

Crea Comentarios

La Fig. 43**,** muestra cómo se crea un comentario, en este ejemplo se puede apreciar que para crear un comentario debe existir; ***usuario*** *y* ***post***.

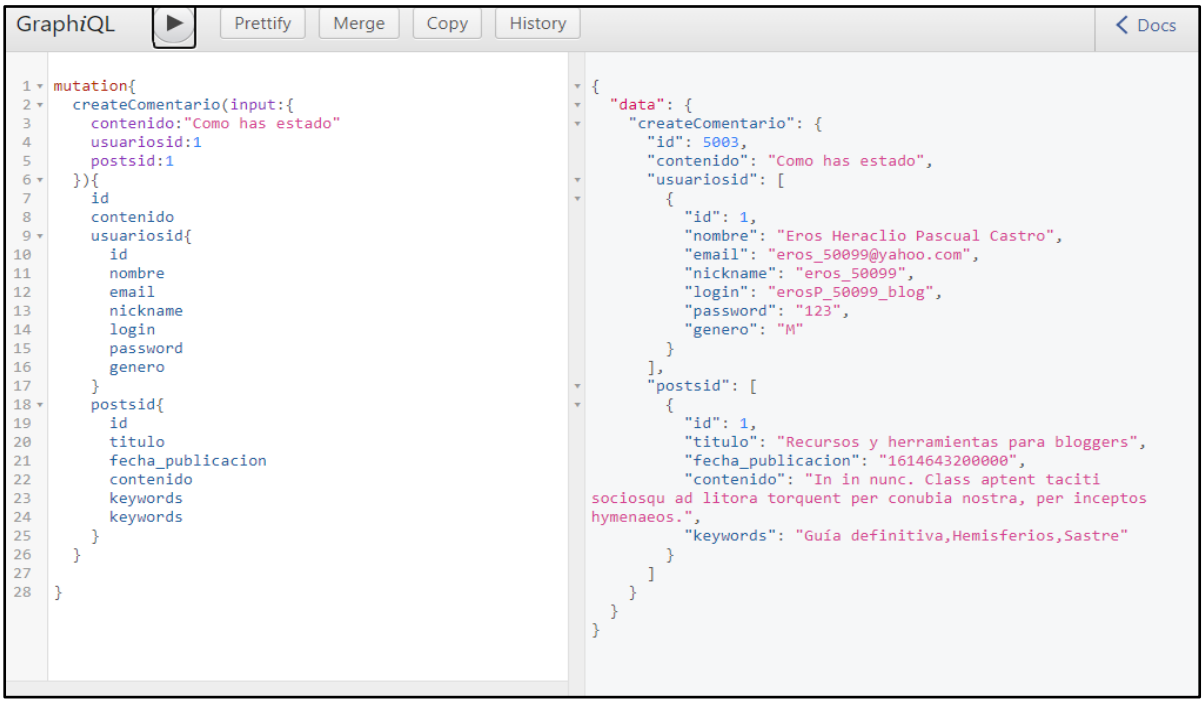


Fig. 43. Crear un Comentario en GraphQL usando GraphiQL.

Editar un Comentario

La Fig. 44, indica como editar un comentario; en este ejemplo no es necesario ingresar el ***usuario*** y el ***post***, ya que en un contexto real el comentario no lo puede editar otro usuario, y tampoco el comentario puede pertenecer a otro post, por lo que el único campo que se puede editar es el ***contenido.***

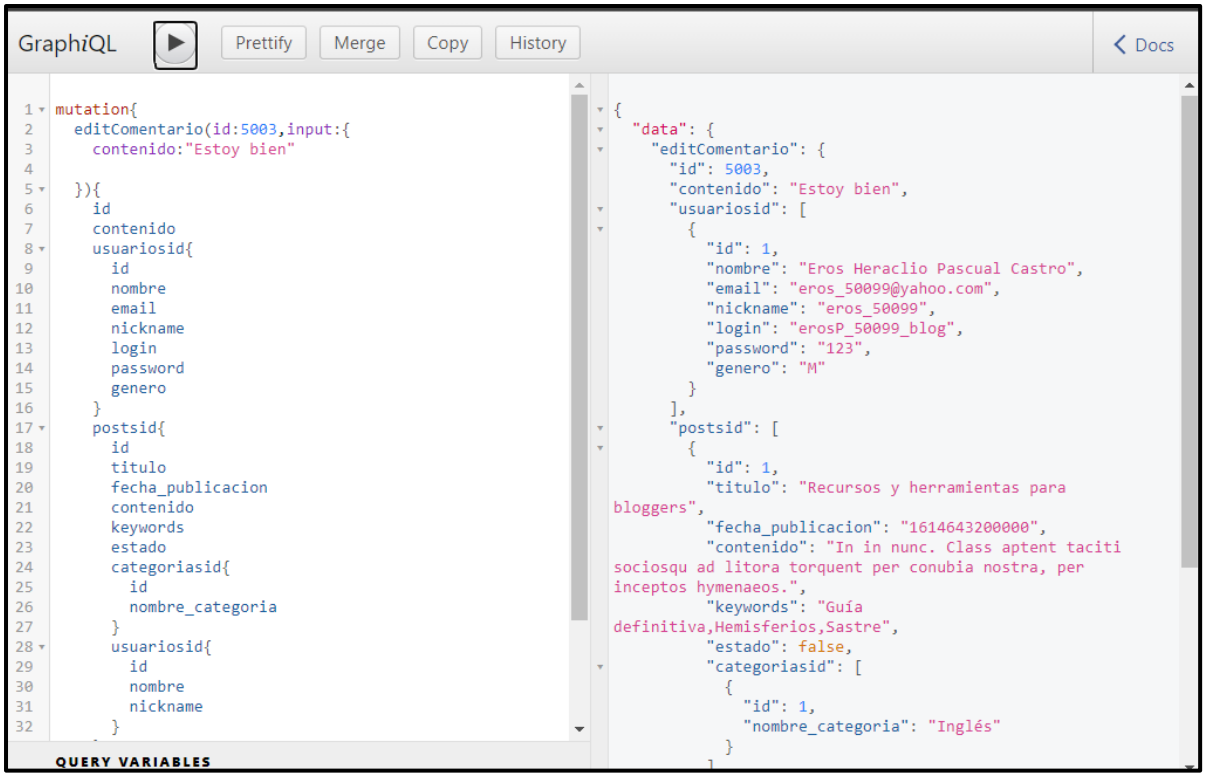


Fig. 44. Editar un Comentario en GraphQL.

Eliminar un Comentario.

La Fig. 45. Indica como eliminar un comentario.

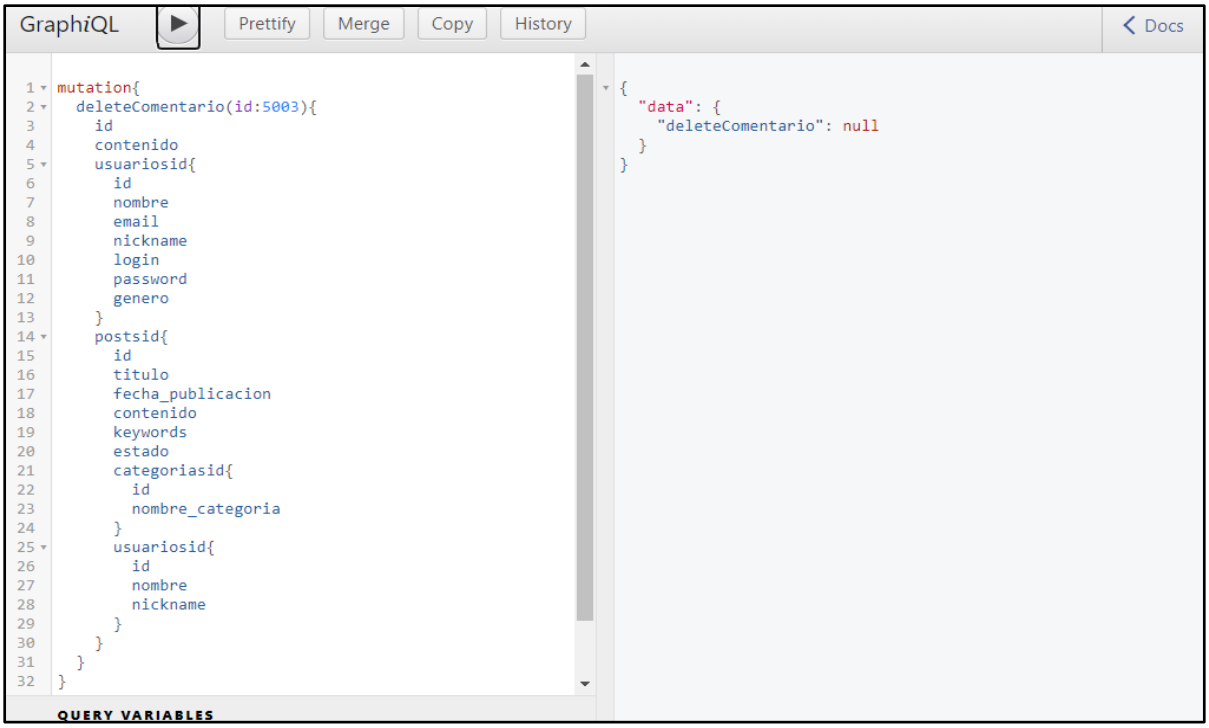


Fig. 45. Eliminar un Comentario en GraphQL.

Crear Posts.

La Fig. 46, muestra cómo se crea un Post en la Api-GraphQL desarrollada, en este ejemplo se puede apreciar que para crear un Post es necesario ingresar ciertos datos y su estructura es muy similar a JSON.

Fig. 46. Crear un Post en GraphQL usando GraphiQL.

Listar Posts.

La Fig. 47 es un ejemplo de cómo se realiza una consulta para obtener un listado personalizado, en este caso se va a obtener un listado de los Posts con los datos que se requiera y como resultado envía un objeto JSON.

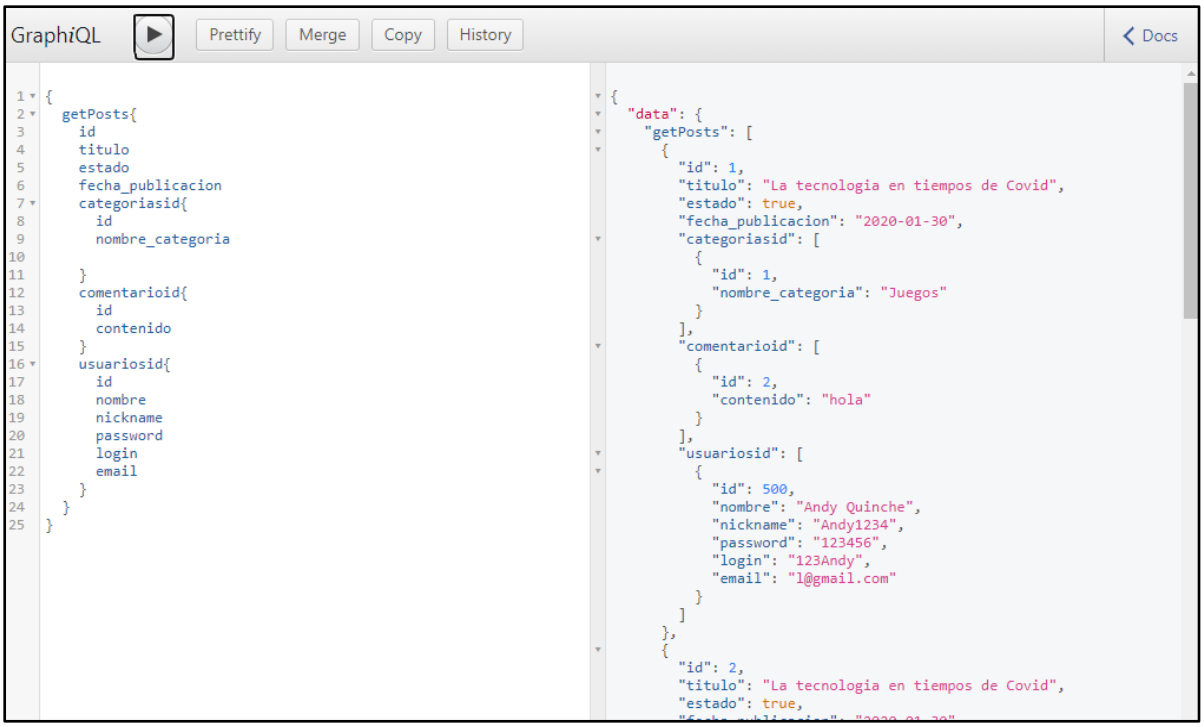


Fig. 47. Respuesta a una consulta GraphQL realizada usando GraphiQL para listar todos los Posts.

- **Repositorio en GitHub.**

TABLA 20 REPOSITORIO DEL PROYECTO DEL API-GRAPHQL SQL

Proyecto	Nombre	GitHub
Api-GraphQL	Api-GraphQLSQL	AndyKingche/API-GraphQLSQL: esta Api-GraphQL es un proyecto de Tesis de la Universidad Técnica del Norte (github.com)

- **Retrospectiva del Sprint 2**
  - **Reunión de Retrospectiva**

Fecha: 1 de diciembre del 2020

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

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

  - **Resultados de Retrospectiva**

TABLA 21 RETROSPECTIVA SPRINT 2

Aciertos (¿Qué salió bien del Sprint?)	Facilidad de realización de pruebas de concepto.
Errores (¿Qué no salió bien del Sprint?)	Problemas con los gestores de base de datos.
Mejoras (¿Qué mejoras se implementará?)	Optimización del tiempo.

2.3.3. Sprint 3.

- **Planificación del Sprint 3**

  - **Reunión de Planificación**

Fecha: 16 de diciembre del 2020

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

Objetivo: Planificación del Sprint 3 (Creación del Sprint backlog)

  - **Sprint Backlog**

La siguiente TABLA 22, muestra a detalle las tareas que se realizaran en este sprint dedicado al desarrollo del API GraphQL conectado a Base de datos NoSQL.

TABLA 22 SPRINT BACKLOG 3

Historias de usuario	Fase de desarrollo	Tarea	Estimación
H-U-3	Desarrollo	Crear el proyecto en el entorno de desarrollo NodeJS, instalar las dependencias que se necesitan para el servidor GraphQL NoSQL.	3
	Desarrollo	Implementar el servidor; crear el esquema, los queries, los mutations y los resolvers, manejo de errores. Integrar todas las Apis realizadas en una sola.	20 5
	Pruebas	Pruebas de funcionamiento del servidor. (Postman)	2
Eventos	Planificación	Detallar las tareas realizadas en el Sprint actual	2
	Revisión	Revisar los resultados del desarrollo del Sprint	2
	Retrospectiva	Analizar los resultados del Sprint	1
		TOTAL	35

- **Revisión del Sprint 3**

  - **Reunión de revisión**

Fecha: 5 de enero del 2021

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

Objetivo: Revisión del desarrollo del incremento del producto.

  - **Pruebas de Aceptación.**

En la TABLA 23, muestra el cumplimiento de los requisitos definidos en el Sprint 3, mediante pruebas de aceptación.

TABLA 23 PRUEBAS DE ACEPTACIÓN SPRINT 3.

HISTORIAS DE USUARIO	NOMBRE	FUNCIONALIDAD	ACEPTACIÓN	
			SI	NO
H-U-2	**Desarrollo de API GraphQL (NoSQL)**	La API permitirá listar los datos consultados.	**X**	
		La API permitirá consultar los datos por ID.	**X**	
		La API permitirá crear, eliminar y editar los datos.	**X**	
		La API tendrá su documentación que servirá como guía para la correcta utilización.	**X**	

- **Incremento del producto potencialmente entregable**

Las operaciones de la API – GraphQL que se conecta a bases de datos NoSQL son muy similares a las operaciones del Sprint anterior, el resultado es el mismo.

- **Repositorio en GitHub.**

TABLA 24 REPOSITORIO DEL PROYECTO DEL API-GRAPHQL NoSQL

Proyecto	Nombre	GitHub
Api-GraphQL	ApiGraphQLMongoDB	AndyKingche/ApiGraphQLMongoDB: Es una Api GraphQL realizada con una base de datos MongoDB (github.com)
	ApiGraphQLRedis	AndyKingche/ApiGraphQLRedis: Es una APiGraphQL desarrollada en node Js y Base de Datsos Redis (github.com)
	ApiGraphQLCassandra	AndyKingche/ApiGraphQLCassandra: Es una APiGraphQL desarrollada en node Js y Base de Datsos Cassandra (github.com)
	ApiGraphQLmultidatabase	AndyKingche/apiGraphQLmultidatabase: Esta Api GraphQL es parte del proyecto de Trabajo de Grado, puede conectarse a 3 bases de datos SQL y 3 NoSQL (github.com)

- **Retrospectiva del Sprint 3**

- **Reunión de Retrospectiva**

Fecha: 5 de enero del 2021

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

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

- **Resultados de Retrospectiva**

TABLA 25 RETROSPECTIVA SPRINT 3

Aciertos (¿Qué salió bien del Sprint?)	Facilidad de realización de pruebas de concepto
Errores (¿Qué no salió bien del Sprint?)	Problemas con los gestores de base de datos, falta de conocimiento de las tecnologías como; Redis y Cassandra
Mejoras (¿Qué mejoras se implementará?)	Optimización del tiempo

2.3.4. Sprint 4

- **Planificación del Sprint 4**

  - **Reunión de Planificación**

Fecha: 5 de enero del 2021

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

Objetivo: Planificación del Sprint 4 (Creación del Sprint backlog)

  - **Sprint Backlog**

La siguiente TABLA 26, muestra a detalle las tareas que se realizaran en este sprint dedicado al desarrollo de un cliente sencillo que se conectara a la API GraphQL desarrollada.

TABLA 26 SPRINT BACKLOG 4

Historias de usuario	Fase de desarrollo	Tarea	Estimación
	Desarrollo	Crear el proyecto en ReactJS con Apollo Client	**2**
	Desarrollo	Crear la vista donde se pueda visualizar la lista de usuarios	**3**
	Desarrollo	Crear la vista donde se pueda visualizar la lista de Comentarios	**3**
	Desarrollo	Crear la vista donde se pueda visualizar la lista de Categorías	**3**
	Desarrollo	Crear la vista donde se pueda visualizar la lista de Etiquetas	**3**
	Desarrollo	Crear la vista donde se pueda visualizar la lista de Posts	**3**
	Desarrollo	Crear una landing page	**3**
		Total	**20**

- **Revisión del Sprint 4**

  - **Reunión de Revisión**

Fecha: 6 enero 2021

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

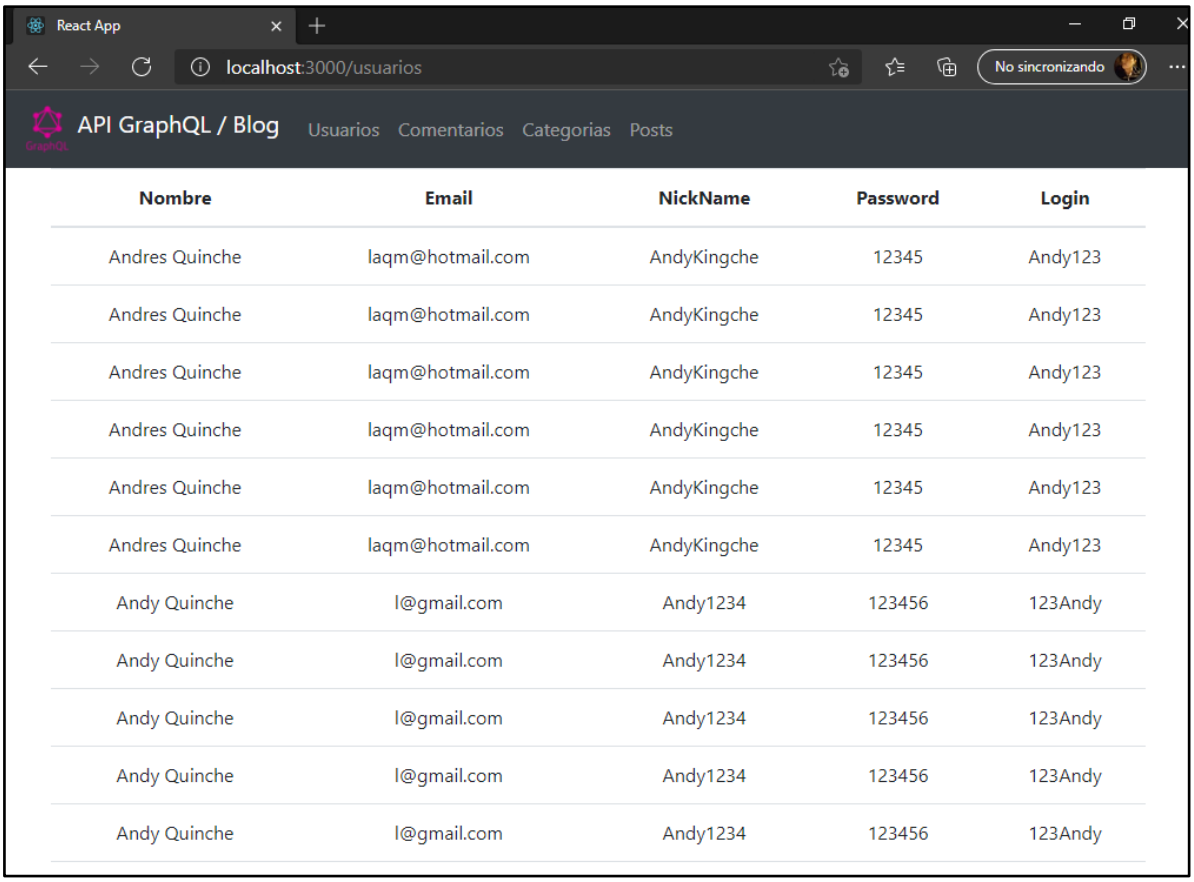
Objetivo: Revisión del Sprint 4 (Creación del Sprint backlog)

  - **Incremento del producto potencialmente entregable**

Las siguientes vistas son realizadas con el framework React y Apollo que ayudaron a construir este proyecto para el consumo de datos de la API-GraphQL. Estas vistas tan solo son una muestra para evidenciar el funcionamiento del consumo de datos de una API-GraphQL.

- **Listar Usuarios**

La Fig. 48 muestra el resultado de la vista el cual imprime todos los usuarios registrados en la base de datos.

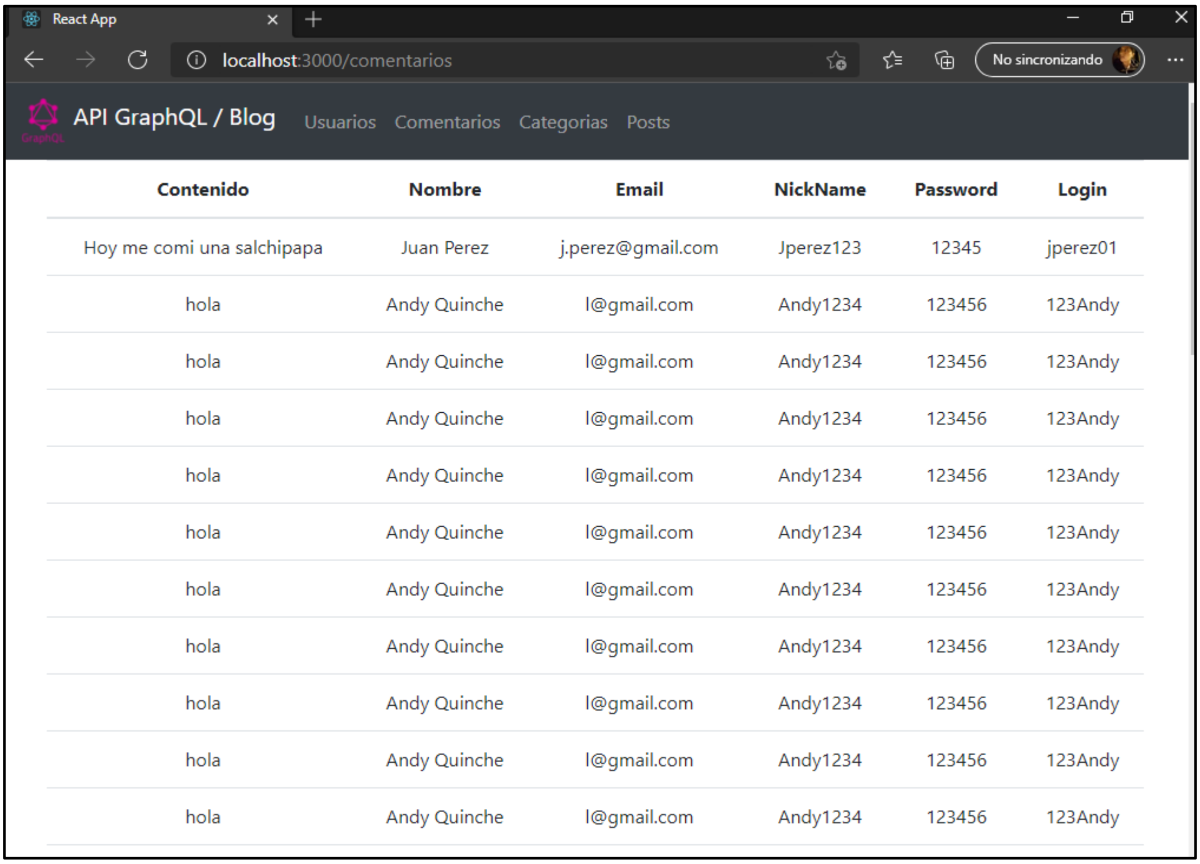


Nombre	Email	NickName	Password	Login
Andres Quinche	laqm@hotmail.com	AndyKingche	12345	Andy123
Andres Quinche	laqm@hotmail.com	AndyKingche	12345	Andy123
Andres Quinche	laqm@hotmail.com	AndyKingche	12345	Andy123
Andres Quinche	laqm@hotmail.com	AndyKingche	12345	Andy123
Andres Quinche	laqm@hotmail.com	AndyKingche	12345	Andy123
Andres Quinche	laqm@hotmail.com	AndyKingche	12345	Andy123
Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
Andy Quinche	l@gmail.com	Andy1234	123456	123Andy

Fig. 48. Vista de la lista de usuarios usando React y Apollo

- **Listar comentarios.**

La Fig. 49 muestra el resultado de la vista el cual imprime todos los comentarios registrados en la base de datos.



Contenido	Nombre	Email	NickName	Password	Login
Hoy me comi una salchipapa	Juan Perez	j.perez@gmail.com	Jperez123	12345	jperez01
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy
hola	Andy Quinche	l@gmail.com	Andy1234	123456	123Andy

Fig. 49. Vista de la lista de comentarios usando React y Apollo.

Listar Categorías.

La Fig. 50 muestra el resultado de la vista el cual imprime todas las categorías registradas en la base de datos.

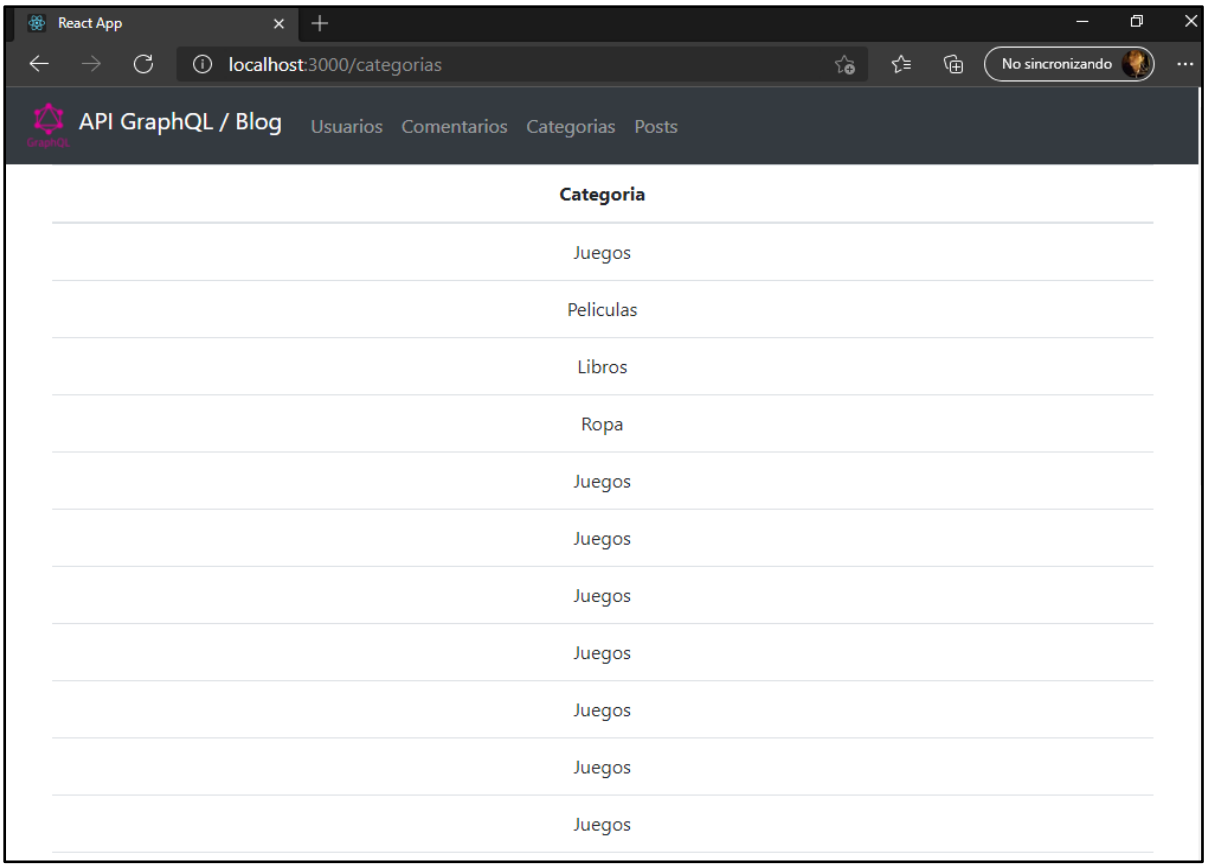


Fig. 50. Vista de la lista de categorías usando React y Apollo.

Listar Post.

La Fig. 51 muestra el resultado de la vista el cual imprime todos los Posts registrados en la base de datos.

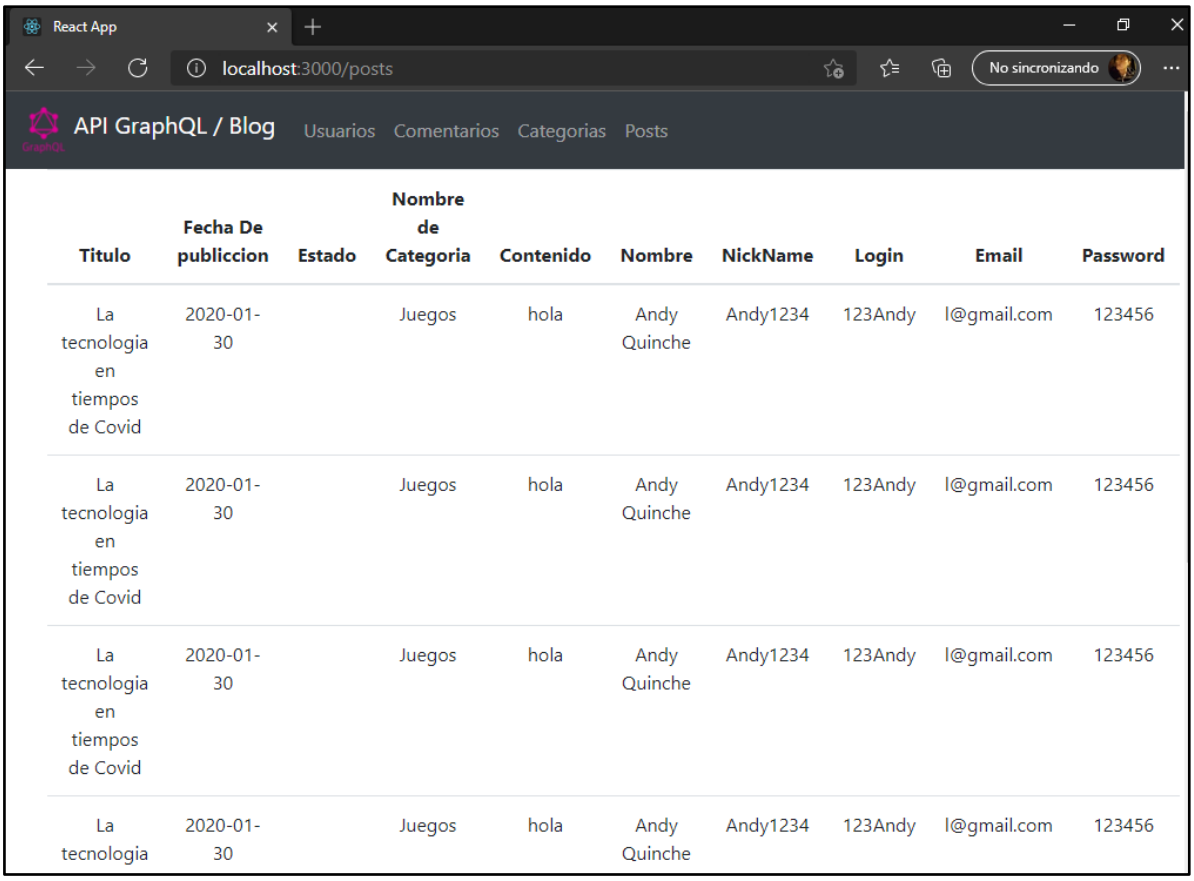


Fig. 51. Vista de la lista de usuarios usando React y Apollo

- **LandingPage**

La Fig.52 es la primera vista que se observara al iniciar la aplicación.

Fig. 52. Vista del Landingpage del cliente desarrollado.

Repositorio en GitHub.

TABLA 27 REPOSITORIO DEL PROYECTO DEL Cliente React.

Proyecto	Nombre	GitHub
Cliente React	Cliente React	AndyKingche/reactApiGraphQL: Este es un cliente desarrollado en React para consumir los datos de una API GraphQL (github.com)

- **Retrospectiva del Sprint 4**
    - **Reunión de Retrospectiva**

Fecha: 19 de enero del 2021

Asistentes: Scrum Master, Product Owner, Equipo de Desarrollo

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

- **Resultados de Retrospectiva**

TABLA 28 RETROSPECTIVA SPRINT 4

Aciertos (¿Qué salió bien del Sprint?)	Facilidad de realización de pruebas de concepto
Errores (¿Qué no salió bien del Sprint?)	Falta de conocimiento de las tecnologías como React
Mejoras (¿Qué mejoras se implementará?)	Optimización del tiempo

CAPITULO 3

EXPERIMENTO: COMPARATIVA DE EFICIENCIA

La comparativa entre las bases de datos relaciones y no relacionales se las realizó mediante un experimento de laboratorio que tuvo las siguientes fases:

- Diseño experimento
- Operación de experimento
- Resultado del experimento

3.1. Diseño del experimento.

El siguiente experimento se diseñó con la finalidad de comparar la eficiencia de consumo de datos desde una API GraphQL que se conecta a tres bases de datos SQL y a tres bases de datos NoSQL. El objetivo de este experimento es demostrar la base de datos más eficiente (es decir que requiere menos tiempo para realizar las consultas) consumida desde una API GraphQL.

Antes de diseñar el experimento es importante declarar las variables independientes y dependientes.

- **Variables independientes:** La API – GraphQL que gestiona la conexión a las diferentes bases de datos (3 Bases de datos SQL y 3 Bases de datos NoSQL).
- **Variable dependiente:** La eficiencia de consumo de datos con respecto al tiempo de respuesta según la norma ISO / IEC 25023**.**

Para lograr el objetivo del experimento se estableció las siguientes preguntas investigación:

- **RQ1:** ¿Qué motor de base de datos relacional es más eficiente al realizar consultas a una API – GraphQL?
- **RQ2:** ¿Qué motor de base de datos no relacional es más eficiente al realizar consultas a una API – GraphQL?
- **RQ3:** ¿Qué motor de base de datos relacional o no relacional es más eficiente al realizar consultas a una API – GraphQL?

Para responder a las preguntas de investigación se diseñó un experimento controlado que involucra tres tipos de consultas detalladas en los casos de uso en la (sección 2.1.4) los cuales ayudaron a construir las tareas del experimento detalladas en la TABLA 29.

TABLA 29 TAREAS DEL EXPERIMENTO

Tipo	ID Caso de Uso	Tareas	Descripción
Consulta de Usuarios	C-U-1	T1	Implementar una consulta que devuelva los datos completos de un Usuario e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T2	Implementar una consulta que devuelva los datos completos de 100 Usuarios e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T3	Implementar una consulta que devuelva los datos completos de 1000 Usuarios e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T4	Implementar una consulta que devuelva los datos completos de 50 000 Usuarios e iterar 3 veces seguidas; tomar el tiempo de respuesta.
Consulta de Posts	C-U-2	T5	Implementar una consulta que devuelva los datos completos de un Post e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T6	Implementar una consulta que devuelva los datos completos de 100 Post e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T7	Implementar una consulta que devuelva los datos completos de 1000 Posts e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T8	Implementar una consulta que devuelva los datos Posts de 50 000 Posts e iterar 3 veces seguidas; tomar el tiempo de respuesta.
Consulta de Posts y sus Comentarios	C-U-3	T9	Implementar una consulta que devuelva los datos completos de un Post y sus Comentarios e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T10	Implementar una consulta que devuelva los datos completos de 100 Posts y sus Comentarios e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T11	Implementar una consulta que devuelva los datos completos de 1000 Post y sus Comentarios e iterar 3 veces seguidas; tomar el tiempo de respuesta.
		T12	Implementar una consulta que devuelva los datos completos de 50 000 comentarios e iterar 3 veces seguidas; tomar el tiempo de respuesta.

Para realizar la ejecución del experimento se lo realizará en dos grupos. El primer grupo consta de los motores de base de datos relacionales y el segundo grupo de los motores de base de datos no relacionales, cada grupo tendrá que cumplir con todas las tareas establecidas, para obtener el motor de base de datos más eficiente bajo la perspectiva del tiempo de respuesta obtenido en el experimento. Se realizará la comparativa de la eficiencia del tiempo clasificado en los dos grupos de tipos de base de datos.

El entorno experimental utilizado en esta sección se muestra en la TABLA 30.

TABLA 30. ESPECIFICACIONES DE LA MAQUINA

Marca	Procesador	Memoria RAM	Tipo de sistema	Disco Duro
Laptop-Lenovo	Intel Core i5-8250U 1.80 GHZ	8 GB	64 bits. Windows 10	1TB

3.2. Procedimiento experimental.

Para cumplir con la ejecución de las tareas del experimento se diseñó un sencillo software que consume los datos desde un cliente desarrollado en NodeJS a la API – GraphQL. El software contiene todos los casos de usos y es el medio por el cual se realiza el cálculo del tiempo de respuesta dada por la API – GraphQL.

La fórmula del tiempo de respuesta está dada por la norma ISO / IEC 25023 detallada en la TABLA 3.

Tiempo de respuesta = *Tiempo en recibir la respuesta – tiempo de envió de la petición*

Todas las operaciones realizadas en este experimento han sido rastreadas y registradas en las siguientes tablas de acuerdo con las tareas del experimento.

En la sección 3.2.1. las tablas muestran los resultados del tiempo de respuesta (tiempo en segundos) obtenidos al realizar la ejecución de las peticiones por medio del cliente al servidor API – GraphQL cumpliendo con las tareas asignadas las cuales consisten en realizar consultas a la API – GraphQL en las diferentes bases de datos a las que se conecta.

3.2.1. Resultados del experimento

- **Resultados generales del caso de uso 1.**

Las Tablas 31 y 32 muestran los tiempos de respuesta obtenidos al realizar las consultas detalladas en el caso de uso 1 donde se pedía una consulta de un número determinado de Usuarios.

TABLA 31. TIEMPOS RESPUESTA – CASOS DE USO 1 REALIZADA CON BASES DE DATOS SQL

	Usuarios	1era iteración	2da iteración	3er iteración	Promedio
MYSQL	**1**	0.030	0.028	0.028	0.028
	100	0.033	0.036	0.032	0.034
	1000	0.051	0.057	0.052	0.053
	50000	0.62	0.65	0.57	0.613
POSTGRESQL	**1**	0.031	0.028	0.027	0.029
	100	0.032	0.032	0.030	0.031
	1000	0.059	0.051	0.052	0.054
	50000	0.63	0.61	0.54	0.593
MSSQL	**1**	0.029	0.034	0.028	0.030
	100	0.039	0.037	0.036	0.037
	1000	0.060	0.064	0.058	0.061
	50000	0.92	0.89	0.90	1.90

TABLA 32. TIEMPOS DE ESPERA – CASOS DE USO 1 REALIZADA CON BASES DE DATOS NOSQL

	Usuarios	1era iteración	2da iteración	3er iteración	Promedio
MongoDB	**1**	0.027	0.027	0.029	0.028
	100	0.031	0.037	0.029	0.032
	1000	0.047	0.053	0.052	0.051
	50000	0.55	0.60	0.56	0.57
Redis	**1**	0.039	0.030	0.029	0.033
	100	0.055	0.041	0.040	0.045
	1000	0.049	0.065	0.089	0.068
	50000	1.56	0.99	1.46	1.34
Cassandra	**1**	0.029	0.029	0.029	0.029
	100	0.040	0.041	0.049	0.043
	1000	0.061	0.052	0.049	0.054
	50000	1.44	0.99	1.36	1.26

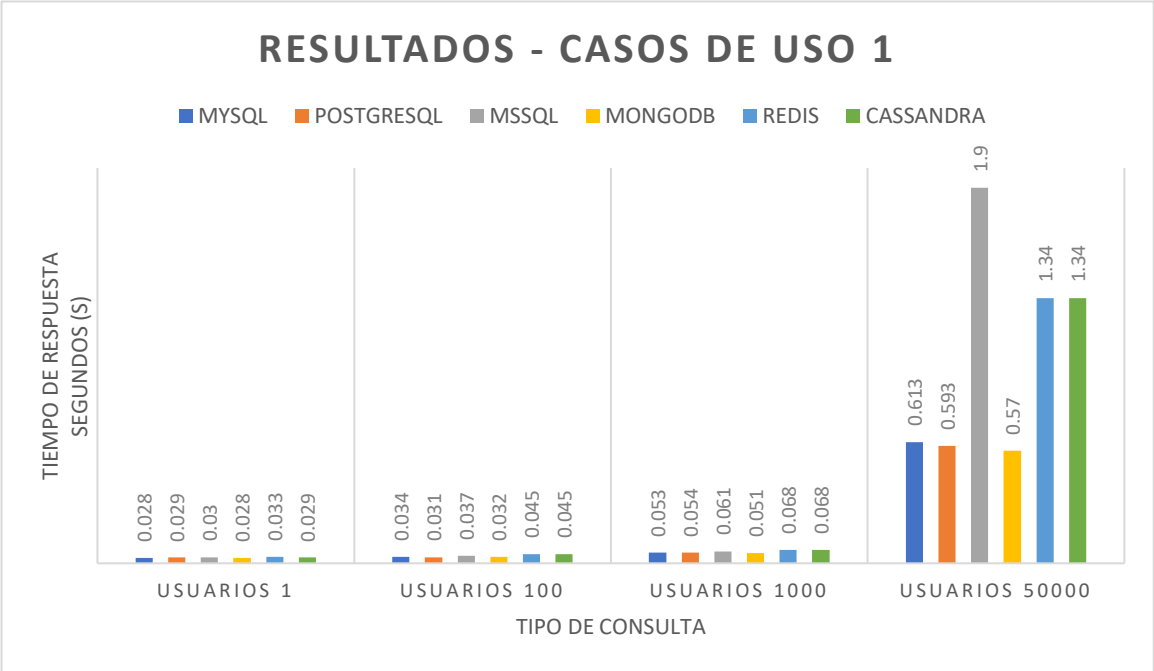


Fig. 53. Gráfico de los resultados generales del Caso de Uso 1.

- **Resultados generales del caso de uso 2.**

Las Tablas 33 y 34 muestran los tiempos de respuesta obtenidos al realizar las consultas detalladas en el caso de uso 2 donde se pedía una consulta de un número determinado de Posts.

TABLA 33. TIEMPOS DE RESPUESTA – CASOS DE USO 2 REALIZADA CON BASES DE DATOS SQL

	Posts	1era iteración	2da iteración	3er iteración	Promedio
MYSQL	**1**	0.028	0.027	0.027	0.027
	100	0.040	0.039	0.037	0.039
	1000	0.073	0.065	0.065	0.067
	50000	2.28	2.51	2.35	2.38
POSTGRESQL	**1**	0.030	0.027	0.035	0.030
	100	0.036	0.039	0.037	0.037
	1000	0.076	0.074	0.077	0.068
	50000	2.53	2.51	2.96	2.66
MSSQL	**1**	0.045	0.037	0.029	0.037
	100	0.049	0.047	0.045	0.047
	1000	0.10	0.084	0.089	0.091
	50000	3.82	4.31	3.98	4.03

TABLA 34. TIEMPOS DE RESPUESTA – CASOS DE USO 2 REALIZADA CON BASES DE DATOS NOSQL

	Posts	1era iteración	2da iteración	3er iteración	Promedio
MongoDB	**1**	0.026	0.027	0.026	0.026
	100	0.041	0.041	0.042	0.041
	1000	0.069	0.068	0.069	0.068
	50000	2.19	2.19	2.67	2.35
Redis	**1**	0.029	0.037	0.029	0.031
	100	0.053	0.047	0.046	0.048
	1000	0.090	0.087	0.089	0.088
	50000	2.45	2.58	3.47	2.83
Cassandra	**1**	0.027	0.026	0.026	0.026
	100	0.049	0.036	0.037	0.041
	1000	0.086	0.086	0.076	0.082
	50000	2.17	2.18	2.17	2.17

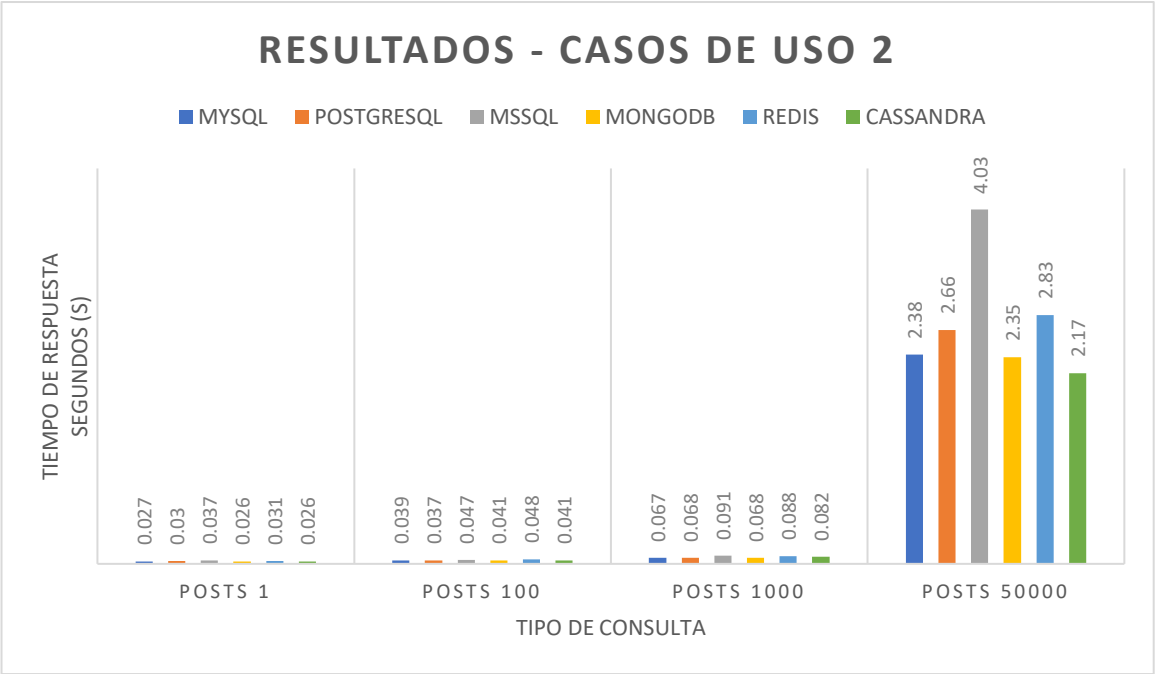


Fig. 54. Gráficos de los resultados generales del caso de uso 2.

- **Resultados generales del caso de uso 3.**

Las Tablas 35 y 36 muestran los tiempos de respuesta obtenidos al realizar las consultas detalladas en el caso de uso 3 donde se pedía una consulta de un número determinado de Posts y Comentarios.

TABLA 35. TIEMPOS DE RESPUESTA – CASOS DE USO 3 REALIZADA CON BASES DE DATOS SQL

	Comentario y Post	1era iteración	2da iteración	3er iteración	Promedio
MYSQL	**1**	0.030	0.030	0.033	0.031
	100	0.040	0.047	0.041	0.042
	1000	0.13	0.12	0.12	0.12
	50000	4.85	4.11	3.72	4.22
POSTGRESQL	**1**	0.027	0.030	0.031	0.029
	100	0.050	0.050	0.047	0.049
	1000	0.16	0.15	0.20	0.17
	50000	4.10	3.92	3.93	3.98
MSSQL	**1**	0.031	0.030	0.032	0.031
	100	0.059	0.069	0.045	0.057
	1000	0.11	0.13	0.14	0.13
	50000	5.20	6.07	5.16	5.48

TABLA 36. TIEMPOS DE RESPUESTA – CASOS DE USO 3 REALIZADA CON BASES DE DATOS NOSQL

	Comentario y Post	1era iteración	2da iteración	3er iteración	Promedio
MongoDB	**1**	0.050	0.047	0.055	0.051
	100	0.078	0.084	0.060	0.074
	1000	0.15	0.24	0.17	0.19
	50000	6.29	7.12	7.00	6.80
Redis	**1**	0.055	0.060	0.062	0.059
	100	0.089	0.080	0.095	0.088
	1000	0.17	0.35	0.55	0.356
	50000	78.5	11.01	13.42	34.31
Cassandra	**1**	0.050	0.055	0.034	0.046
	100	0.060	0.073	0.071	0.068
	1000	0.16	0.16	0.25	0.19
	50000	6.98	8.72	5.89	7.19

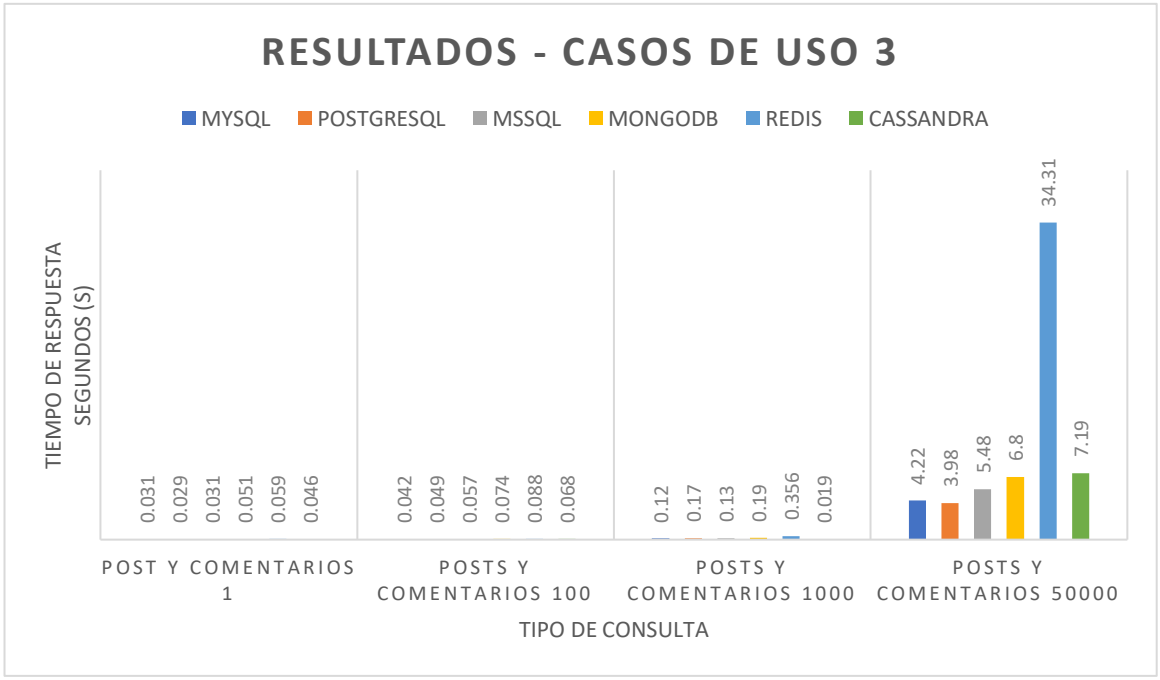


Fig. 55. Gráfico de los resultados generales del caso de uso 3.

CAPITULO 4

ANÁLISIS DE RESULTADOS

4.1. Análisis y Resultados.

En esta sección se realizó el análisis de los resultados obtenidos en la sección 3.1 del proceso experimental y antes de sacar una conclusión, el análisis se hará en base a las preguntas de investigación planteadas en el diseño de experimento (sección 3.1).

- ***RQ1:*** *¿Qué motor de base de datos relacional es más eficiente al realizar consultas a una API – GraphQL?*
    - **Resultados del caso de uso 1 – Bases de datos SQL.**

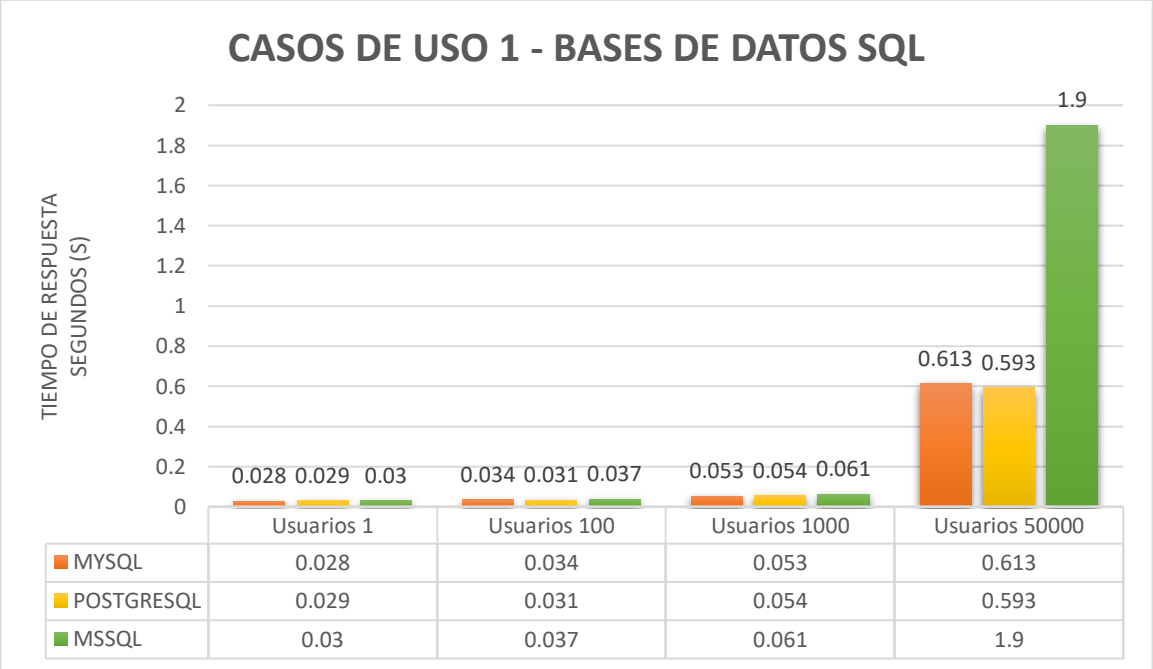


Fig. 56. Gráfico de los resultados del caso de uso 1 - comparativa de las bases de datos SQL.

La base de datos más eficiente tomando en cuenta los resultados del caso de uso 1 es PostgreSQL.

- **Resultados del caso de uso 2 – Bases de datos SQL.**

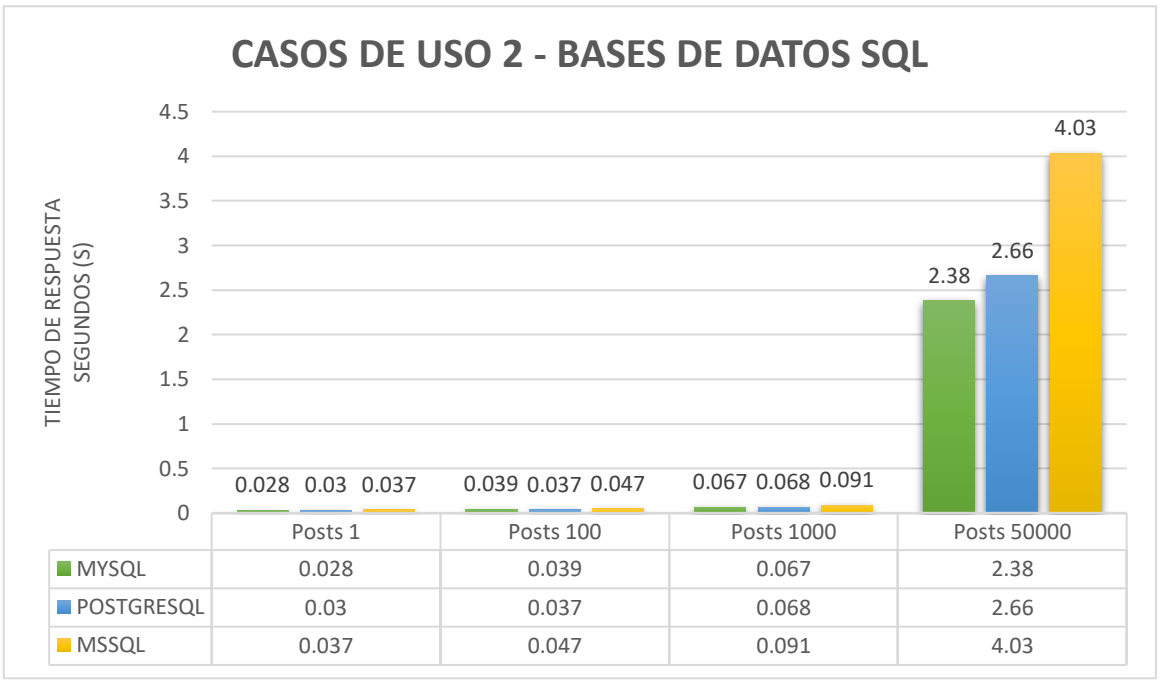


Fig. 57. Gráfico de los resultados del caso de uso 2 - comparativa de las bases de datos SQL

La base de datos más eficiente tomando en cuenta los resultados del caso de uso 2 es MYSQL.

- **Resultados del caso de uso 3 – Bases de datos SQL.**

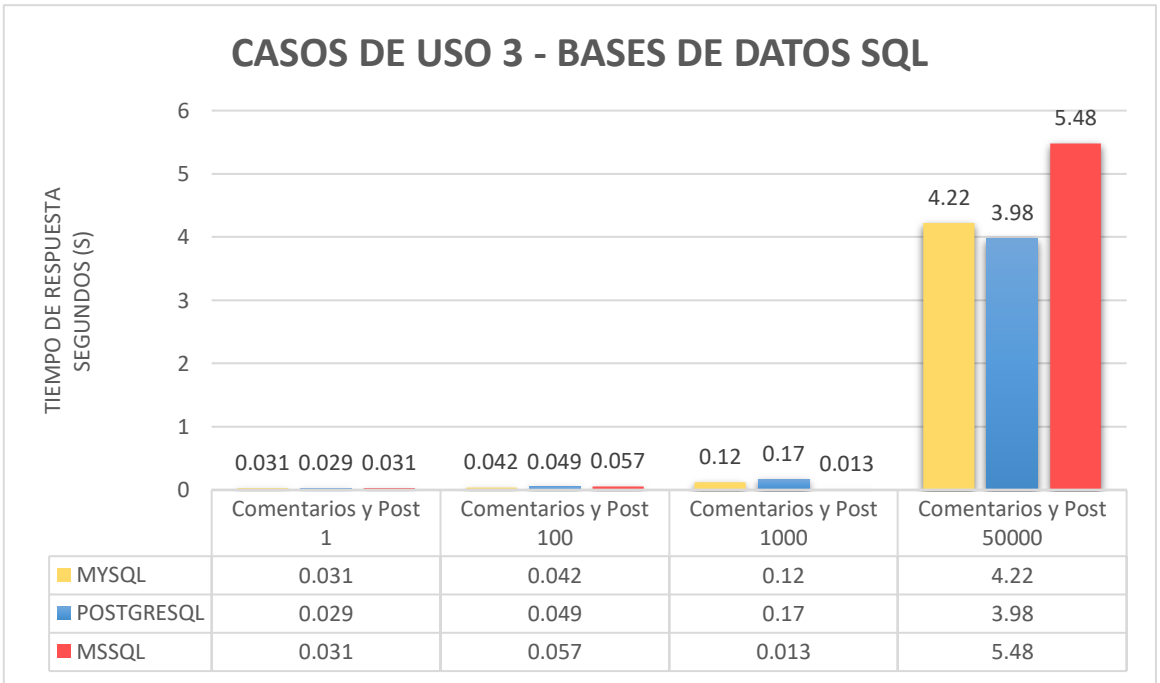


Fig. 58. Gráfico de los resultados del caso de uso 3 - comparativa de las bases de datos SQL

La base de datos relacional más eficiente tomando en cuenta los resultados del caso de uso 3 es PostgreSQL.

La TABLA 37 muestra un resumen de la comparativa del tiempo de respuesta promedio (segundos) de los casos de uso de las bases de datos relacionales ordenadas de menor a mayor tiempo.

TABLA 37 RESUMEN DE LOS RESULTADOS - BASES DE DATOS RELACIONALES.

Caso de uso	Base de datos	Tiempo promedio de respuesta (segundos)	Base de datos más eficiente
Caso de uso 1	POSTGRESQL	0.178	POSTGRESQL
	MYSQL	0.182	
	MSSQL	2.028	
Caso de uso 2	MYSQL	0.628	MYSQL
	POSTGRESQL	0.698	
	MSSQL	1.051	
Caso de uso 3	POSTGRESQL	1.057	POSTGRESQL
	MYSQL	1.103	
	MSSQL	1.42	

Tomando en cuenta el caso de uso 1, 2 y 3 se llegó a la conclusión que el mejor motor de base de datos relacional es PostgreSQL ya que demostró ser la base de datos más eficiente obteniendo el menor tiempo de respuesta durante todo el experimento.

- ***RQ2:*** *¿Qué motor de base de datos no relacional es más eficiente al realizar consultas a una API – GraphQL?*

    - **Resultados del caso de uso 1 – Bases de datos NoSQL.**

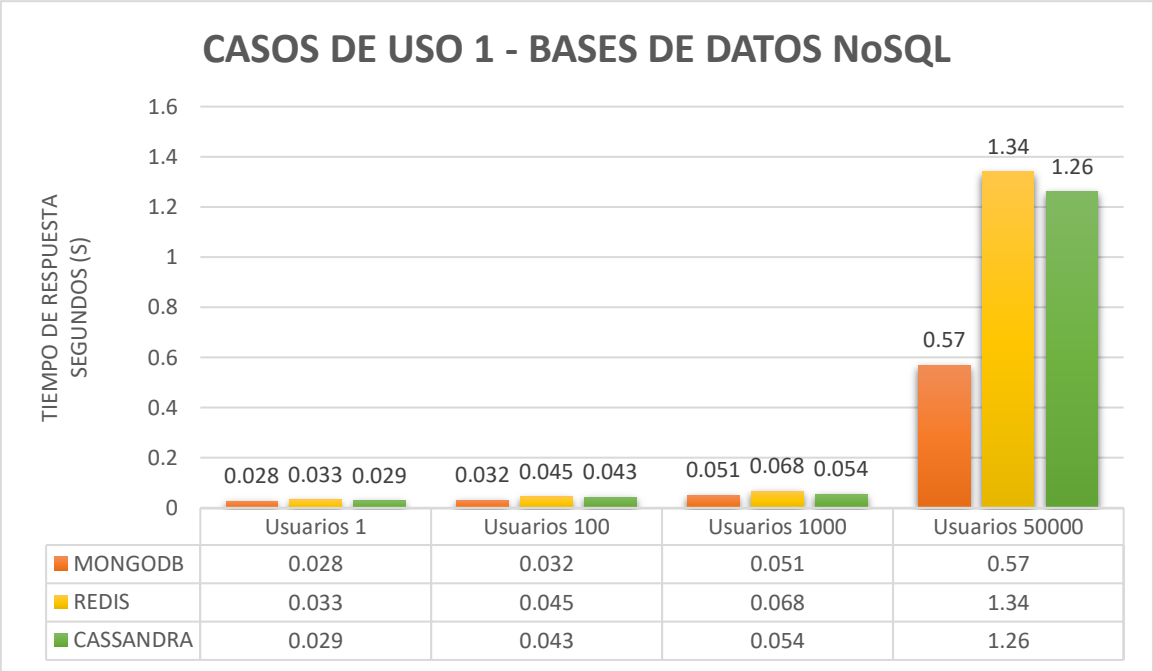


Fig. 59. Gráfico de los resultados del caso de uso 1 - comparativa de las bases de datos NoSQL

La base de datos más eficiente tomando en cuenta los resultados del caso de uso 1 es MongoDB.

- **Resultados del caso de uso 2 – Bases de datos NoSQL.**

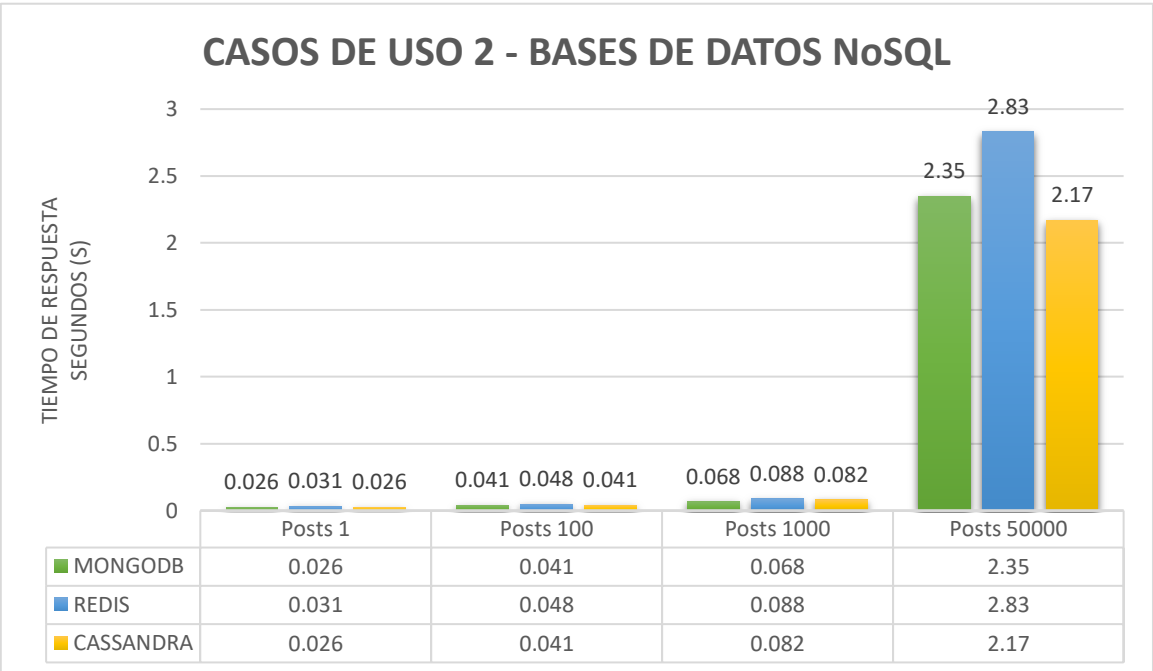


Fig. 60. Gráfico de los resultados del caso de uso 2 - comparativa de las bases de datos NoSQL

La base de datos más eficiente tomando en cuenta los resultados del caso de uso 2 es CASSANDRA.

- **Resultados del caso de uso 3 – Bases de datos NoSQL.**

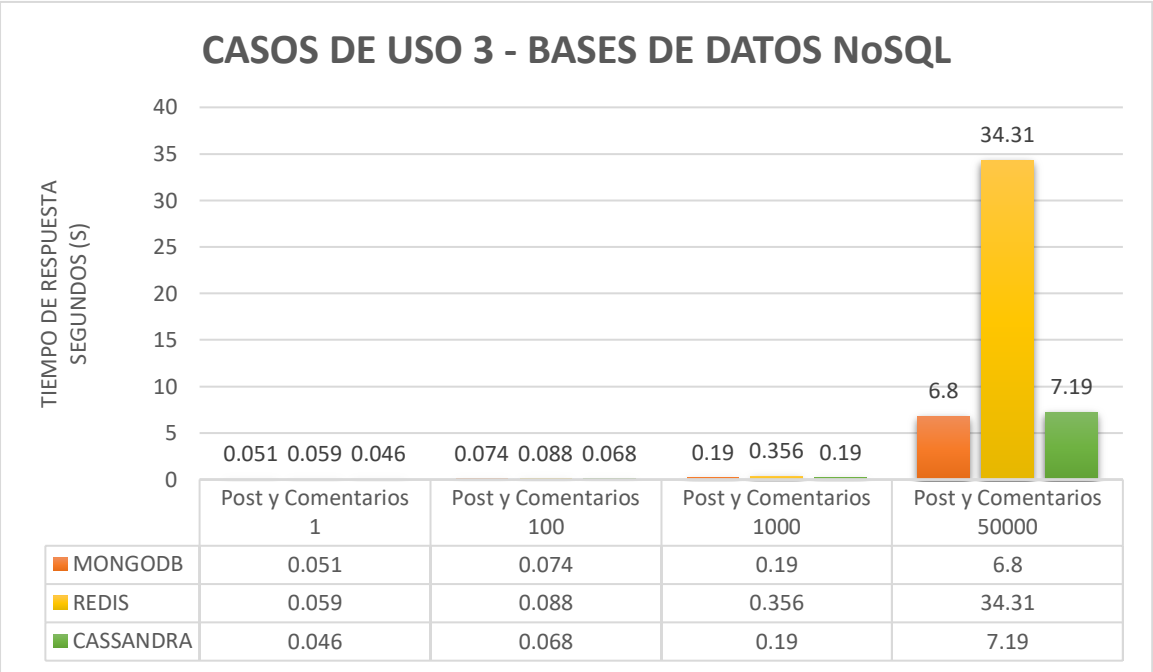


Fig. 61. Gráfico de los resultados del caso de uso 3 - comparativa de las bases de datos NoSQL

La base de datos más eficiente tomando en cuenta los resultados del caso de uso 3 es MongoDB.

La TABLA 38 muestra un resumen de la comparativa del tiempo de respuesta promedio (segundos) de los casos de uso de las bases de datos no relacionales ordenadas de menor a mayor tiempo.

TABLA 38 RESUMEN DE LOS RESULTADOS - BASES DE DATOS NO RELACIONALES.

Caso de uso	Base de datos	Tiempo promedio de respuesta (segundos)	Base de datos más eficiente
Caso de uso 1	MONGODB	0.170	MONGODB
	CASSANDRA	0.346	
	REDIS	0.371	
Caso de uso 2	CASSANDRA	0.579	CASSANDRA
	MONGODB	0.621	
	REDIS	0.749	
Caso de uso 3	MONGODB	1.778	MONGODB
	CASSANDRA	1.873	
	REDIS	8.703	

Tomando en cuenta los resultados del caso de uso 1, 2 y 3 se llegó a la conclusión que el mejor motor de base de datos no relacional es MongoDB demostrando ser la base de datos más eficiente obteniendo el menor tiempo de respuesta durante todo el experimento.

- ***RQ3:*** *¿Qué motor de base de datos relacional o no relacional es más eficiente al realizar consultas a una API – GraphQL?*

Para responder a esta pregunta se realizará una comparativa entre los mejores motores de base de datos SQL y NoSQL.

- **Resultados del caso de uso 1 – Bases de datos SQL Y NoSQL.**

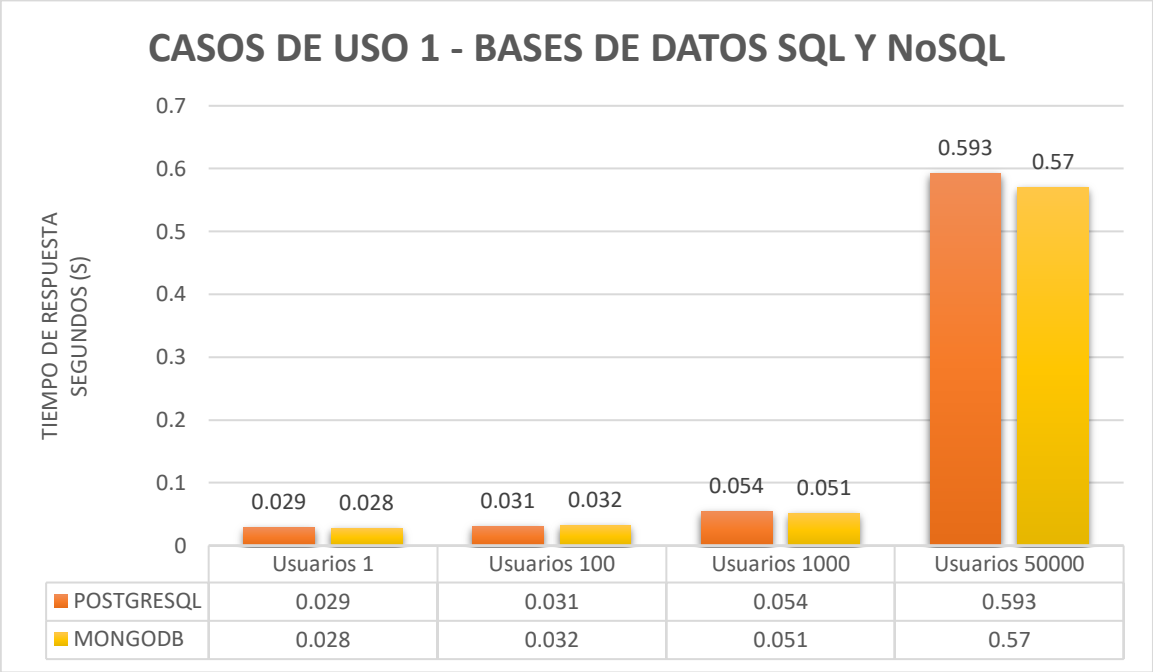


Fig. 62. Gráfico de los resultados del caso de uso 1 - comparativa de las bases de datos SQL Y NoSQL

La base de datos más eficiente tomando en cuenta los resultados del caso de uso 1 es MongoDB.

- **Resultados del caso de uso 2 – Bases de datos SQL Y NoSQL.**

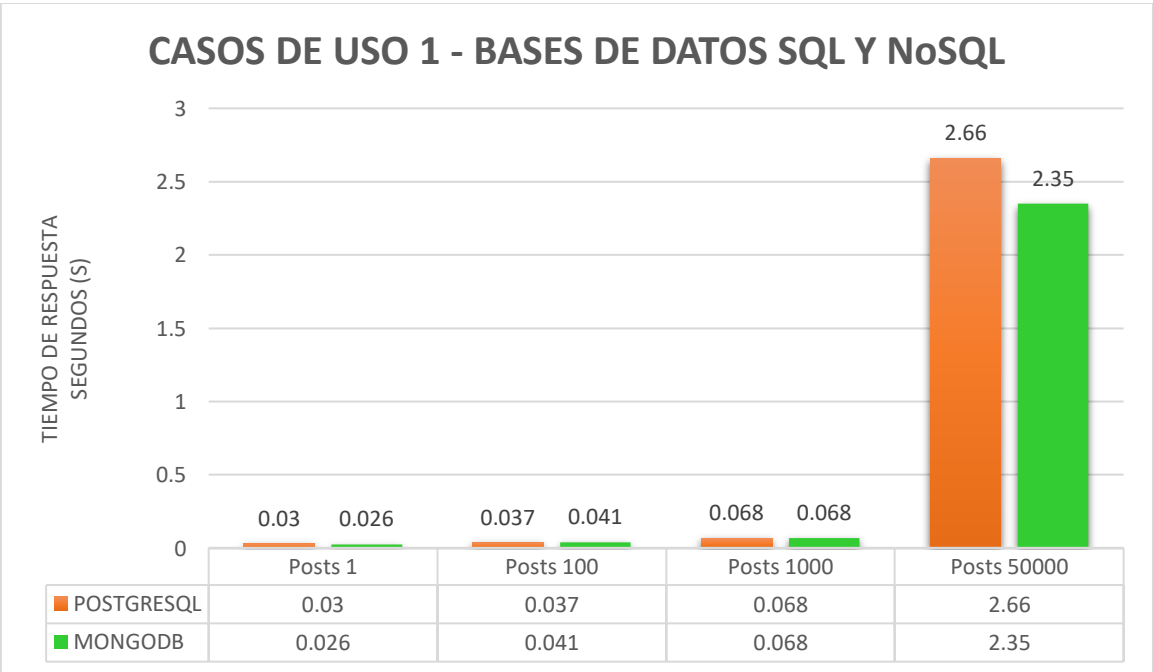


Fig. 63. Gráfico de los resultados del caso de uso 2 - comparativa de las bases de datos SQL Y NoSQL

La base de datos más eficiente tomando en cuenta los resultados del caso de uso 2 es MongoDB.

- **Resultados del caso de uso 3 – Bases de dato SQL Y NoSQL.**

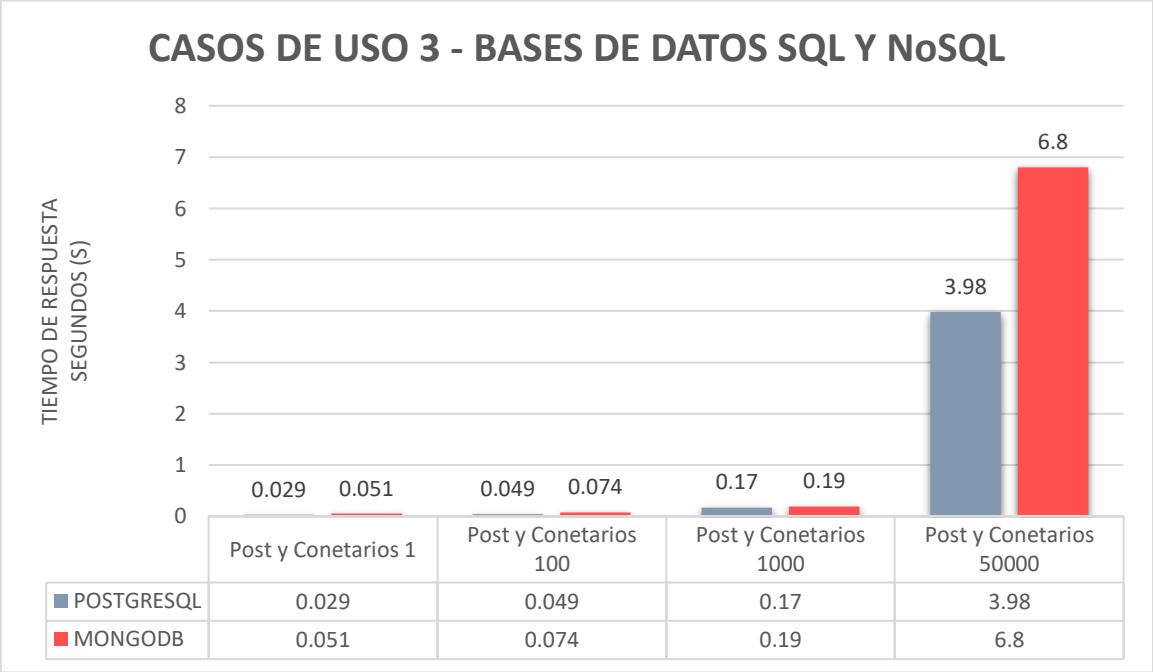


Fig. 64. Gráfico de los resultados del caso de uso 3 - comparativa de las bases de datos SQL Y NoSQL

La base de datos más eficiente tomando en cuenta los resultados del caso de uso 3 es PostgreSQL.

La TABLA 39 muestra un resumen de la comparativa del tiempo de respuesta promedio (segundos) de los casos de uso de los mejores motores de bases de datos no relacionales y relacionales ordenadas de menor a mayor tiempo.

TABLA 39 RESUMEN DE LOS RESULTADOS – LOS MEJORES MOTORES DE BASES DE DATOS NO RELACIONALES Y RELACIONALES.

Caso de uso	Base de datos	Tiempo promedio de respuesta (segundos)	Base de datos más eficiente
Caso de uso 1	MONGODB	0.170	MONGODB
	POSTGRESQL	0.178	
Caso de uso 2	MONGODB	0.621	MONGODB
	POSTGRESQL	0.698	
Caso de uso 3	POSTGRESQL	1.778	POSTGRESQL
	MONGODB	1.057	

Tomando en cuenta los casos de uso 1, 2 y 3 se ha llegado a la conclusión que el motor de base de datos que se lleva el título a mejor base de datos integrada con una API –

GRAPHQL es MongoDB, debido a que es la base de datos que obtuvo los mejores tiempos de respuesta.

4.2. Resumen de análisis de resultados.

En la TABLA 40. se presenta un resumen de la comparativa de los resultados obtenidos de las bases de datos relacional y no relacional que más se destacaron durante el experimento.

TABLA 40 RESUMEN DE LA COMPARATIVA ENTRE LAS BASES DE DATOS MÁS EFICIENTES.

Casos de uso	Bases de datos.	Tiempo por nro. de registro				Base de datos más eficiente
		1	100	1000	50000	
1	PostgreSQL	0.029	0.031	0.054	0.593	MongoDB
	MongoDB	0.028	0.032	0.051	0.057	
2	PostgreSQL	0.030	0.037	0.068	2.66	MongoDB
	MongoDB	0.026	0.041	0.068	2.35	
3	PostgreSQL	0.029	0.049	0.17	3.98	PostgreSQL
	MongoDB	0.051	0.074	0.19	6.80	

Una vez analizados todos los resultados obtenidos en el experimento se pudo observar que la base de datos más EFICIENTE en el consumo de datos mediante una API - GraphQL es la base de datos MongoDB pues se cumple lo que dijo Now (2020) que los documentos hacen que el procesamiento sea más fácil y rápido para distribuir los datos a través de los servidores.

MongoDB es la base de datos que obtuvo los mejores tiempos en todo el experimento a excepcion del caso de uso 3, en donde el que más se destaco es la base de datos PostgreSQL. Por lo que la base de datos relacional PostgreSQL se puede utilizar como alternativa ante MongoDB ya que también sus tiempos de respuesta fueron los mejores en este experimento.

CONCLUSIONES Y RECOMENDACIONES.

CONCLUSIONES.

- Con el desarrollo del marco teórico se definió la base conceptual científica que sirvió para determinar la arquitectura tecnológica, las tecnologías para el desarrollo del API-GraphQL, el diseño de la base de datos y el diseño del experimento para realizar la comparativa de la eficiencia con respecto al tiempo de respuesta de la API - GraphQL. También se pudo definir la base conceptual de la norma ISO/IEC 25023 para a evaluar la calidad del software.
- En cuanto al desarrollo del API-GraphQL, se pudo utilizar las tecnologías establecidas en la planificación del proyecto como: Javascript y NodeJS, su librería npm como gestor de dependencias y express como marco de desarrollo de aplicaciones web, la cual permitió gestionar el acceso a seis bases de datos (tres SQL y 3 NoSQL).
- Mediante el diseño de un experimento controlado, usando el API-GraphQL desarrollada se logró comparar la eficiencia del consumo de datos entre Bases de datos relacionales y no relacionales, basadas en las metricas ISO / IEC 25023. En donde la ejecución del experimento mostro que las variables dependientes se ven afectadas por las variables independientes, es decir que el tiempo de respuesta es afectada por la API – GraphQL que se conecta a las diferentes bases de datos SQL y NoSQL.
- Con los resultados obtenidos en cuanto al tiempo de respuesta de acceso a datos mediante una API – GraphQL se pudo contestar las preguntas de investigación establecidas en el experimento: (RQ1) *¿Qué motor de base de datos relacional es más eficiente al realizar consultas a una API – GraphQL?* La base de datos relacional más es eficiente es PostgresQL. (RQ2) *¿Qué motor de base de datos no relacional es más eficiente al realizar consultas a una API – GraphQL?* La base de datos no relacional más es eficiente es MongoDB. (RQ3) *¿Qué motor de base de datos relacional o no relacional es más eficiente al realizar consultas a una API – GraphQL?* La base de datos relacional o no relacional más es eficiente es MongoDB.
- La base de datos relacional PostgreSQL se destacó en el caso de uso 3 donde la consulta fue a tres niveles, obteniendo el mejor puntaje a diferencia de la base de datos MongoDB, por lo cual se puede concluir que esta base de datos puede ser una alternativa adecuada cuando se usa consultas de este tipo.
- La diferencia entre una API – GraphQL contruida con bases de datos NoSQL y contruida con bases de datos SQL no fueron tan diferentes como se esperaba, ya que en ocaciones llegaban a tener los mismos resultados.

RECOMENDACIONES.

- Se recomienda realizar un estudio de como la API – GraphQL puede comportarse con las bases de datos que no se mencionaron en este experimento.
- Se recomienda a la comunidad de desarrollo y tecnológico utilizar GraphQL con MongoDB y los implementen en sus aplicaciones ya que tendrán un buen rendimiento en sus proyectos.
- En cuanto a la base de datos REDIS, sus componentes para el desarrollo en los frameworks y realizar las consultas no estan 100% optimizadas, por lo que se recomienda crear funciones adicionales para realizar consultas a la bases de datos, lo que puede influir en el tiempo de respuesta.
- Se recomienda a la comunidad de desarrollo y tecnológico analizar el código fuente que está alojado en GitHub llamado apiGraphQLmultidatabse (link: AndyKingche/apiGraphQLmultidatabase: Esta Api GraphQL es parte del proyecto de Trabajo de Grado, puede conectarse a 3 bases de datos SQL y 3 NoSQL (github.com)) y proponer mejoras en el código, corrección de errores o agregar nuevas funcionalidades.
- Se recomienda investigar más acerca de GraphQL ya que es un tema que no se ha abordado tanto y es una herramienta fácil de usar e integrar en sus proyectos.
- Como trabajo futuro se recomienda realizar un estudio de la comparativa de un cliente GraphQL desarrollado en los frameworks React, Angular, Vue.js, según stateofjs (stateofjs, 2020) estos tres son los frameworks más utilizados en el 2020. El cliente tiene que consumir datos de la API – GraphQL conectada a la base de datos ganadora, es decir MongoDB. De esta forma estaríamos contribuyendo a la comunidad científica y tecnológica con un STACK de desarrollo.
- Se recomienda realizar investigaciones del comportamiento de una API – GraphQL con frameworks de desarrollo de aplicaciones móviles como Android, Xamarin y Flutter.
- Se recomienda realizar un estudio de investigación entrevistando y encuestando a profesionales para analizar sus puntos de vista y experiencia del uso de GraphQL.

REFERENCIAS:

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

Brito, G., & Valente, M. T. (2020). REST vs GraphQL: A controlled experiment. *Proceedings - IEEE 17th International Conference on Software Architecture, ICSA 2020*, (Dcc), 81–91. https://doi.org/10.1109/ICSA47634.2020.00016

Čerešňák, R., & Kvet, M. (2019). Comparison of query performance in relational a non-relation databases. *Transportation Research Procedia*, *40*, 170–177. https://doi.org/10.1016/j.trpro.2019.07.027

Db-Engines. (2020, October 20). historical trend of the popularity ranking of database management systems. Retrieved November 20, 2020, from Db-Engines website: https://db-engines.com/en/ranking_trend

Explore Group. (2019, October 20). Las bases de datos más populares de 2019 | Blog | Explorar grupo. Retrieved November 20, 2020, from Explore Group website: https://www.explore-group.com/blog/the-most-popular-databases-2019/bp46/

GraphQL. (2020a, November 19). Consultas y mutaciones | GraphQL. Retrieved November 19, 2020, from GraphQL website: https://graphql.org/learn/queries/

GraphQL. (2020b, December 20). Running an Express GraphQL Server | GraphQL. Retrieved March 15, 2021, from Running an Express GraphQL Server | GraphQL website: https://graphql.org/graphql-js/running-an-express-graphql-server/

Hartina, D. A., Lawi, A., & Panggabean, B. L. E. (2018). Performance Analysis of GraphQL and RESTful in SIM LP2M of the Hasanuddin University. *Proceedings - 2nd East Indonesia Conference on Computer and Information Technology: Internet of Things for Industry, EIConCIT 2018*, 237–240. https://doi.org/10.1109/EIConCIT.2018.8878524

Haupt, F., Leymann, F., Scherer, A., & Vukojevic-Haupt, K. (2017). A Framework for the Structural Analysis of REST APIs. *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, 55–58. https://doi.org/10.1109/ICSA.2017.40

Isha, Sharma, A., & Revathi, M. (2018). Automated API Testing. *Proceedings of the 3rd International Conference on Inventive Computation Technologies, ICICT 2018*, 788–791. https://doi.org/10.1109/ICICT43934.2018.9034254

ISO 25023. (2020). *Inte/iso/iec 25023:2020*. (506). https://doi.org/https://www.iso.org/standard/35747.html

JerBrains. (2020, October 20). Bases de datos 2019 - Infografía del estado del ecosistema del desarrollador en 2019. Retrieved November 20, 2020, from JerBrains website: https://www.jetbrains.com/es-es/lp/devecosystem-2019/databases/

Jindal, A., & Madden, S. (2014). *G RAPH iQL : A Graph Intuitive Query Language for Relational Databases*. 441–450.

Li, S. (2018). Understanding quality attributes in microservice architecture. *Proceedings - 2017 24th Asia-Pacific Software Engineering Conference Workshops, APSECW 2017*, *2018-Janua*, 9–10. https://doi.org/10.1109/APSECW.2017.33

Li Zhang, Jia-Hao Tian, Jing Jiang, Yi-Jun Liu, M.-Y. P. & T. Y. (2018). Empirical Strategies in Software Engineering Research : A Literature Survey. *Journal of Computer Science and Technology*, 7. https://doi.org/https://doi.org/10.1007/s11390-018-1864-x

Martinek, P. (2019). A comparative review of microservices and monolithic architectures. *ArXiv*, 149–154.

Mesbah, A. (2015). Advances in testing javascript-based web applications. In *Advances in Computers* (1st ed., Vol. 97). https://doi.org/10.1016/bs.adcom.2014.12.003

Munaf, R. M., Ahmed, J., Khakwani, F., & Rana, T. (2019). Microservices architecture: Challenges and proposed conceptual design. *2019 International Conference on Communication Technologies, ComTech 2019*, (ComTech), 82–87. https://doi.org/10.1109/COMTECH.2019.8737831

Now, L., Database, R., & Systems, M. (2020). *Sql 3.1*. 31–43. https://doi.org/10.1016/B978-0-12-814915-7.00003-X

Petrasch, R. (2017). Model-based engineering for microservice architectures using Enterprise Integration Patterns for inter-service communication. *Proceedings of the 2017 14th International Joint Conference on Computer Science and Software Engineering, JCSSE 2017*, 5–8. https://doi.org/10.1109/JCSSE.2017.8025912

Platzi. (2020, October 20). Bases de Datos desde Cero. Retrieved November 20, 2020, from Platzi website: https://platzi.com/base-de-datos/

Quintana Rondón, Y., Camejo Domínguez, L., & Díaz Berenguer, A. (2011). Diseño de la Base de Datos para Sistemas de Digitalización y Gestión de Medias. *Revista de Informática Educativa y Medios Audiovisuales*, *8*(15), 17. Retrieved from http://laboratorios.fi.uba.ar/lie/Revista/Articulos/080815/A3mar2011.pdf

Ret Hat. (2019, October 9). Microservicios. Retrieved December 10, 2020, from Ret Hat website: https://www.redhat.com/es/topics/microservices/what-are-microservices

Sill, A. (2016). The Design and Architecture of Microservices. *IEEE Cloud Computing*, *3*(5), 76–80. https://doi.org/10.1109/MCC.2016.111

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

Sousa, M. De. (2020). *humanportal – A React . js case study*. (June), 24–27.

stateofjs. (2020). State of JS 2020: Front-end Frameworks. Retrieved June 28, 2021, from Front-end Frameworks website: https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/

Stol, K. J., & Fitzgerald, B. (2015). A Holistic Overview of Software Engineering Research Strategies. *Proceedings - 3rd International Workshop on Conducting Empirical Studies in Industry, CESI 2015*, 47–54. https://doi.org/10.1109/CESI.2015.15

Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Computing*, *14*(6), 80–83. https://doi.org/10.1109/MIC.2010.145

Vazquez-Ingelmo, A., Cruz-Benito, J., & García-Penalvo, F. J. (2017). Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL. *ACM International Conference Proceeding Series*, *Part F1322*, 1–8. https://doi.org/10.1145/3144826.3145437

Wohlin, C., Höst, M., & Henningsson, K. (2003). Empirical research methods in software engineering. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2765*, 7–23. https://doi.org/10.1007/978-3-540-45143-3_2

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. In *Experimentation in Software Engineering* (Vol. 9783642290). https://doi.org/10.1007/978-3-642-29044-2