UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Software

COMPARACIÓN DE LA FACILIDAD DE APRENDIZAJE ENTRE GRAPHQL Y REST MEDIANTE UN MODELO DE CALIDAD INTERNA BASADA EN LAS ISO/IEC 25000

Trabajo de grado previo a la obtención de título de Ingeniera en Software

Autor:

Jonathan Fernando Chulde Benavides

Director:

MSc. Antonio Quiña Mera

Ibarra - Ecuador 2022

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:	100410090-3		
APELLIDOS Y NOMBRES:	JONATHAN FERNANDO CHULDE BENAVIDES		
DIRECCIÓN:	AZAYA, ISLA FERNANDINA 11-28		
EMAIL:	jfchuldeb@utn.edu.ec jfercho9603@hotmail.com		
TELÉFONO FIJO:	062 545 405	**TELÉFONO MÓVIL:**	0968700801

DATOS DE LA OBRA	
TÍTULO:	"COMPARACIÓN DE LA FACILIDAD DE APRENDIZAJE ENTRE GRAPHQL Y REST MEDIANTE UN MODELO DE CALIDAD INTERNA BASADA EN LA ISO/IEC 25000"
AUTOR:	JONATHAN FERNANDO CHULDE BENAVIDES
FECHA:	01/08/2022
PROGRAMA:	■ PREGRADO ☐ POSGRADO
TÍTULO POR EL QUE OPTA:	INGENIERA EN SOFTWARE
ASESOR /DIRECTOR:	ING. MSC. ANTONIO QUIÑA

2. CONSTANCIAS

El autor manifiesta 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 el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 1 días del mes de agosto de 2022

EL AUTOR:

Nombre: Jonathan Fernando Chulde Benavides
Cédula: 100410090-3

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE GRADO

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN DEL ASESOR

Certifico que la Tesis previa a la obtención del título de Ingeniera en Software con el tema: "COMPARACIÓN DE LA FACILIDAD DE APRENDIZAJE ENTRE GRAPHQL Y REST MEDIANTE UN MODELO DE CALIDAD INTERNA BASADA EN LA ISO/IEC 25000" ha sido desarrollada y terminada en su totalidad por el Sr. Jonathan Fernando Chulde Benavides, con cédula de identidad Nro. 100410090-3 bajo mi supervisión para lo cual firmo en constancia.

JOSE ANTONIO QUIÑA MERA

Firmado digitalmente por JOSE ANTONIO QUIÑA MERA
Nombre de reconocimiento (DN): c=EC, serialNumber=1002322384, sn=QUIÑA MERA, cn=JOSE ANTONIO QUIÑA MERA, 1.3.6.1.4.1.37442.10.4=1002322384, givenName=JOSE ANTONIO, email=aquina@utn.edu.ec, st=IMBABURA, l=IBARRA, ou=Certificado de Clase 2 de Persona Fisica EC (FIRMA)

Msc. Antonio Quiña

DIRECTOR DE TESIS

Dedicatoria

Dedico el presente proyecto de titulación a mi madre Anita Benavides, a mi padre Luis Fernando Chulde, a mi hermana Katty Chulde, y mi hermano Santiago Chulde, los cuales son mi inspiración para seguir adelante, gracias a su apoyo, su calidad como familia, la cual hace de los días más amenos y divertidos, el disfrutar de pequeños momentos como familia son el mejor combustible para mantener mi fortaleza y no sentirme solo, y perseverar por mis sueños en busca de mejores oportunidades, siempre manteniendo mis principios y valores como persona, y demostrando que el cambio sucede desde uno mismo.

Agradecimientos

Agradezco principalmente a mis padres, hermanos, por su apoyo, por creer en mí, gracias a cosas tan simples, llenan de motivación para seguir adelante.

Agradezco a mis compañeros de curso, el transcurso de la carrera fue muy amena y enriquecedora gracias a las largas horas de estudio y ocio que no solo te nutren a nivel académico, también te nutren como persona.

Agradezco a mi tutor, MSc. Antonio Quiña, por el apoyo y seguimiento para culminar y poder presentar el trabajo de titulación.

Agradezco a mis opositores, MSc. Cathy Guevara y MSc. Mauricio Rea, su tiempo y observaciones fueron valiosas para la corrección del trabajo de titulación.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Resumen

Los sistemas informáticos actuales presentan arquitecturas en la parte del backend de servicios y microservicios, para lo cual se utiliza tecnologías como REST, la más usada en la actualidad, y GraphQL que nace como respuesta a problemas que presentaba REST como arquitectura de servicios, sin embargo, se tiene entendido que GraphQL tiene una complejidad en comparación a REST con su implementación, por lo que en el desarrollo de APIs es comúnmente ver consumos de servicios con REST.

El presente trabajo compara las tecnologías de GraphQL y REST con respecto a la facilidad de aprendizaje, se diseñó un modelo de calidad interna basado en la ISO/IEC 25000, se utilizó un diseño Crossover para realizar el experimento controlado. Se requirió una base teórica para el conocimiento de los conceptos y poder familiarizarse con el experimento controlado y su procedimiento, en el cual se realizó una preparación a un grupo de sujetos de prueba, para posterior recolectar los resultados del experimento a través de una evaluación de conocimientos prácticos de las tecnologías, y validar estos resultados con la ISO/IEC 25000, y se receptaron resultados esperados del experimento.

Palabras clave: REST, GraphQL, experimento controlado, modelo de calidad interna, ISO/IEC 25000, facilidad de aprendizaje.

Abstract

Current computer systems present architectures into the services and microservices backend, for which technologies such as REST, the most used today, and GraphQL, which was born as a response to problems presented by REST as a service architecture, are used, however, it is understood that GraphQL has a complexity compared to REST with its implementation, so that in the development of APIs it is common to see service consumption with REST.

The present work compares GraphQL and REST technologies with respect to ease of learning, an internal quality model was designed based on ISO/IEC 25000, a Crossover design was used to perform the controlled experiment. A theoretical base was required for the knowledge of the concepts and to be able to familiarize with the controlled experiment and its procedure, in which a group of test subjects was prepared, to later collect the results of the experiment through an evaluation of practical knowledge of the technologies, and validate these results with ISO/IEC 25000, and the expected results of the experiment were received.

Keywords: REST, GraphQL, controlled experiment, internal quality model, ISO/IEC 25000, ease of learning.

Introducción

Antecedentes

El software se encuentra en la mayor parte de las actividades humanas: industria, el comercio, las finanzas, el gobierno, la salud, la educación, etc. La importancia de la calidad de software es ampliamente reconocida en la actualidad, sin embargo, desde el punto de vista de los modelos y estándares hacia el producto, el desarrollo de estos durante décadas, la sobreabundancia de información, el alto costo y el acceso limitado a esta información, impiden un acercamiento de estos a los ingenieros de software en pro de la calidad del producto de software (Vaca & Jácome, 2018).

GraphQL es un lenguaje de consulta novedoso para implementar arquitectura de software basada en servicios, es así como el lenguaje está ganando impulso y ahora lo utilizan las principales empresas de software, como Facebook y GitHub. El lenguaje fue desarrollado e implementado en Facebook, como una solución a varios problemas que enfrentan al usar estilos arquitectónicos estándar, como REST (Brito & Valente, 2020). REST es muy usado para servicios web, pero tiene problemas de over-fetching (descarga de información irrelevante para la aplicación) y under-fetching (proveedor de datos en una sola consulta no ofrece toda la información que el cliente necesita) (Taskula, 2019). Una forma muy útil para comparar de manera objetiva estas arquitecturas basadas en servicios es a través de un modelo de calidad de la ISO/IEC 25000.

El principal instrumento para garantizar la calidad de las aplicaciones o desarrollos web sigue siendo el modelo de calidad, el cual se basa en normas o estándares genéricos (ISO/IEC 25000), lo importante de estos, es que estén escritos, personalizados, adoptados a los procesos del proyecto y que sean cumplidos (Mera Paz, 2016). Pero para realizar la comparativa entre REST Y GraphQL necesitamos también de un experimento controlado.

La ingeniería de software no se trata solo de soluciones técnicas. En gran medida se ocupa también de cuestiones organizativas, gestión de proyectos y comportamiento humano. Para una disciplina como esta, los métodos empíricos son cruciales, ya que permiten incorporar el comportamiento humano al enfoque de investigación adoptado. Un experimento controlado a menudo se realiza para comparar varias técnicas diferentes, métodos, procedimientos de trabajo, etc. En un experimento se tiene control sobre el estudio y como los participantes llevan a cabo las tareas asignadas, la ventaja de esto es que se puede planificar y diseñar para tener validez garantizada (Wohlin et al., 2003). El enfoque del experimento controlado será en la usabilidad de GraphQL y REST, específicamente la facilidad de aprendizaje.

Se tiene que la usabilidad que es parte de las características de la norma ISO/IEC 25000, se define como la medida del grado de facilidad en el uso de un tipo de producto (en este caso “tecnológico” como lo son REST y GraphQL) y del tipo de satisfacción que genera ese uso en el usuario (Alarcón-Aldana et al., 2014), una de las sub características analizar dentro de la usabilidad, es la facilidad de aprendizaje, la cual según la (ISO/IEC 25023, 2020) es: “la medida que se utiliza para evaluar el grado en el cual un producto o sistema puede ser utilizado por usuarios especificados para alcanzar los objetivos de aprendizaje especificados, para usar el producto o sistema con eficacia, eficiencia, ausencia de riesgo y satisfacción en un contexto de uso especifico.

Situación Actual

En la actualidad existen empresas que tienen falencias en la adecuada producción de software, debido a que no cuentan con un modelo que permita asegurar la calidad del mismo (Roa et al., 2015). La estructura de los modelos de calidad se componen de diversos criterios que son evaluados por métricas, con el propósito de reducir la subjetividad en la asignación de un valor; ya sea cuantitativo o cualitativo (Callejas Cuervo et al., 2017), una de estas métricas es la calidad interna, la cual es relativa a la medición de un atributo estático del software relacionado con su arquitectura (Santos et al., 2017). Además tenemos que el buscar un valor agregado al producto de software es lo que impulsa a la aplicación de normas de calidad, y a la vez brindar productos con altos estándares de calidad y con una disminución de fallos (Mera Paz, 2016). Aquí nace la importancia de usar un modelo de calidad para poder comparar dos servicios como REST y GraphQL de manera objetiva.

GraphQL es una alternativa a las aplicaciones basadas en REST, y para comprender las diferencias de usabilidad como la facilidad de aprendizaje, es que se necesita estudios que contrasten el esfuerzo y percepciones sobre el desarrollo e implementación con estas tecnologías de servicios (Brito & Valente, 2020), en este sentido hemos revisado en las bases de datos bibliográficas, y no existe un estudio que muestre qué tecnología entre GraphQL y REST es más fácil de aprender, lo cual se convierte en una motivación de estudio en el que aplicando la norma ISO/IEC 25000 se implementara el modelo de calidad para de manera objetiva comparar entre estas tecnologías y ver cuál es más fácil de aprender.

Planteamiento del Problema

La falta de estudios que contrasten el esfuerzo y las percepciones de los desarrolladores al usar REST y GraphQL (Brito & Valente, 2020), como viene siendo el estudio sobre la facilidad de aprendizaje de estas tecnologías, impide tener una visión clara sobre el tema. El experimento controlado nos permitirá clarificar el panorama de aprendizaje de servicios como REST y GraphQL de manera objetiva para una enseñanza más optima en

el área académica, teniendo en cuenta que medir la calidad (interna) de un producto de software implica evaluar el modelo para llegar a este (Roa et al., 2015). A continuación, se muestra en la Fig. 1 el árbol de problemas.

Fig. 1. Árbol de problemas

Fuente: Propia

Objetivos

Objetivo General

Comparar la facilidad de aprendizaje entre GraphQL y REST mediante un modelo de calidad interna basada en las ISO/IEC 25000.

Objetivos Específicos

- Establecer un marco teórico de la investigación.
- Establecer un modelo de calidad interna de la facilidad de aprendizaje.
- Comparar GraphQL y REST mediante un experimento controlado.
- Evaluar los resultados del modelo de calidad de aprendizaje basado en las ISO/IEC 25000.

Alcance

La finalidad del proyecto es realizar una comparativa entre REST y GraphQL para establecer qué tecnología de desarrollo de aplicaciones basada en arquitectura orientada a servicios es más fácil de aprender. Para la medición se tomará en cuenta la subcaracterística de facilidad de aprendizaje como eje principal. La medición y evaluación del proyecto, estará basado en un modelo de calidad interna de productos de software de la serie de normas ISO/IEC 25000 (modelo ISO-25010, medición ISO-25023, evaluación ISO-25040).

El marco teórico se enfocará en conceptos generales de experimentos controlados, la calidad interna de la ISO 25000, la arquitectura SOA y las tecnologías a usar como: NodeJs, Postgres, React, REST y GraphQL.

La comparación entre REST y GraphQL se realizará mediante un experimento controlado que tendrá tres fases:

1) Diseño del experimento, en donde se definirá la manera en el que se ejecutará el experimento.

2) Operación del experimento, se ejecutará el diseño del experimento en donde se enseñará las tecnologías REST y GraphQL a los sujetos seleccionados de acuerdo con el diseño del experimento. Las herramientas y tecnologías para usar son: REST, GraphQL, NodeJS, Postgres, VisualStudio Code.

Se desarrollará 2 casos de uso, el primero consistirá en el desarrollo de 2 crud, y el segundo en la realización de consultas combinadas.

3) Análisis de resultados, se analizará los resultados obtenidos en la operación del experimento para obtener conclusiones y recomendaciones del estudio.

En la Fig. 2 se muestra el alcance del proyecto.

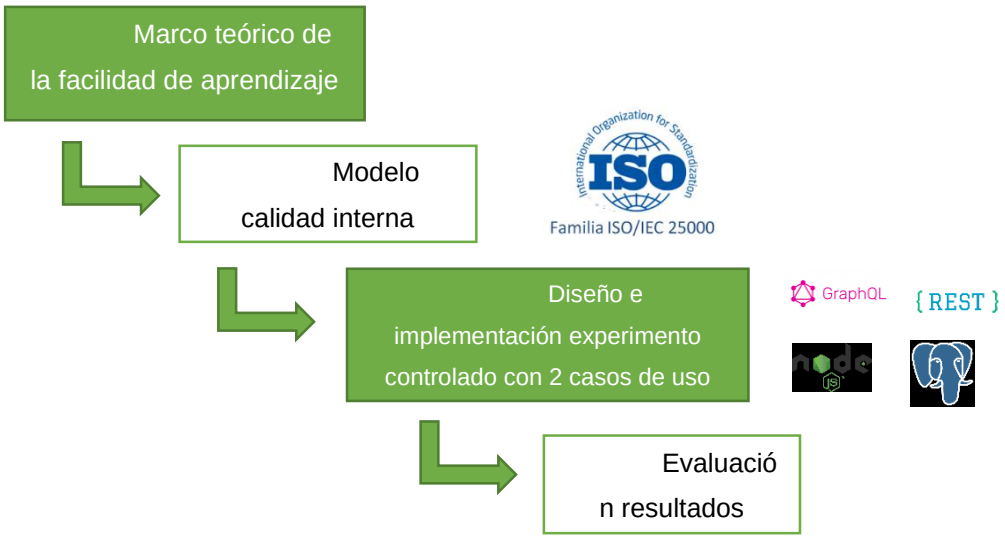


Fig. 2. Alcance

Fuente: Propia

Metodología

El enfoque de esta investigación es cuantitativo pues como dice (Chauhan et al., 2014) “Al medir podemos controlar y mejorar el rendimiento de nuestro proyecto de software, porque sin medir la calidad no podemos lograr el objetivo del software”, el marco teórico tendrá también el enfoque investigativo de manera cualitativa y analítica, los parámetros o métricas a evaluar serán escogidas de la subcaracterística facilidad de aprendizaje que está en la característica de usabilidad de la ISO/IEC 25000, la cual nos brinda el cómo se puede evaluar las métricas. Para poder comprobar las diferencias de facilidad de aprendizaje entre GraphQL

y REST se utilizara un experimento controlado, el cual constara de un diseño, operación, análisis e interpretación de los resultados (Wohlin et al., 2003). A continuación, se muestra en la Fig. 3 el proceso metodológico.

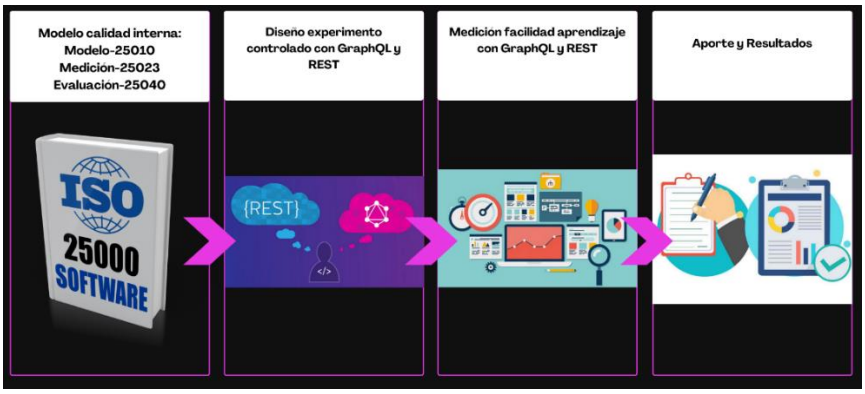


Fig. 3. Proceso metodológico

Fuente: Propia

Justificación

GraphQL tiene gran impulso y se utiliza por grandes empresas como Facebook y GitHub, en reemplazo de REST (Brito & Valente, 2020), como también se tiene que en sistemas distribuidos modernos en los que se involucra una gran carga y flujo de datos, es más utilizado GraphQL, esto se debe a su excelente gestión en el almacenamiento y flexibilidad (Vásquez, 2020), el enfoque de la investigación se dará con el tema de la facilidad de aprendizaje de estos servicios.

Justificación Tecnológica.- El gran impulso que tiene GraphQL en la implementación de arquitecturas basadas en servicios, en remplazo de REST debido a sus problemáticas (Brito & Valente, 2020), abre la interrogante sobre cuál de estos servicios es más fácil de aprender, pero lastimosamente no se tiene evidencia sobre investigaciones relacionadas con el tema, en la mayoría de investigaciones se han enfocado en la eficiencia de estos servicios.

Justificación Teórica.- Es importante tener en cuenta que los modelos de software tienen diversos factores de calidad los cuales se componen de criterios que son evaluados por métricas, el propósito es el de evaluar, desde lo general a particular, y así tener más objetividad en la asignación de un valor cualitativo o cuantitativo (Callejas Cuervo et al., 2017). Es así que la investigación utilizara lineamientos que establece la norma ISO/IEC 25000 para medir la facilidad de aprendizaje entre GraphQL y REST (Vásquez, 2020),y a su vez todo el proceso comparativo realizarlo con un experimento controlado bajo la estructuración del: diseño, operación, análisis e interpretación de los resultados (Wohlin et al., 2003).

Justificación Metodológica.- Los estándares de calidad no han sido muy utilizados, y este es el causante de que los desarrollos sean basados en experiencia y no se logre satisfacer

al cliente final, es así como implementación se vuelve compleja, por lo que requiere un grado de experiencia y conocimiento para poder usar correctamente (VILLANES, 2015), se tiene que la aplicación de un estándar no es suficiente para responder a la problemática de la facilidad de aprendizaje, por lo que realizarlo mediante un experimento controlado permitirá tener una respuesta más clara, y el estudio a realizar podrá ser un referente a futuras investigaciones relacionadas con el tema de aprendizaje de arquitecturas de servicios como REST y GraphQL.

Riesgos

En la

Fig. 4 se muestra la matriz de riesgos.

Bajo	**Nivel de riesgo**	Métricas de calidad interna mal evaluadas		

Medio		Mal diseño de experimento controlado		
Alto			Mala Implementación de modelo de calidad interna ISO 25000	No contar con personal para experimento controlado
		Impacto de riesgo		
		Bajo	**Medio**	**Alto**

Fig. 4. Matriz de riesgo

Fuente: Propia

Mal diseño de experime ado: Para la minimización de este riesgo se tendrá en cuenta la bibliografía de controlados para evitar caer en subjetividades en el diseño y que este perjudi ización de este.

Mala implementación de calidad interna ISO 25000: Debido a que las normas ISO no presentan una guí le implementación se puede tender a implementar mal, por lo que es importante rafía en la que se implemente estos modelos y poder tener como referencia a la futura implementación.

No contar con personal para experimento controlado: Para minimizar este riesgo es necesario la ayuda del tutor, pues representa una gran facilidad el tener como muestra de población para el experimento controlado, a estudiantes que requieran del aprendizaje de GraphQL y REST.

Métricas de modelo de calidad interna mal evaluadas: Para minimizar este riesgo es necesario comprender y tener muy en cuenta las fórmulas que nos brinda la ISO 25023.

CAPITULO I

1. MARCO TEORICO

En este capítulo se estableció un marco conceptual para fundamentar teóricamente los aspectos para realizar la comparación de la facilidad de aprendizaje entre GraphQL y REST mediante un modelo de calidad interna basada en la familia de estándares de la ISO/IEC 25000. A continuación, en la Fig. 5, se muestra la estructura del contenido del capítulo.

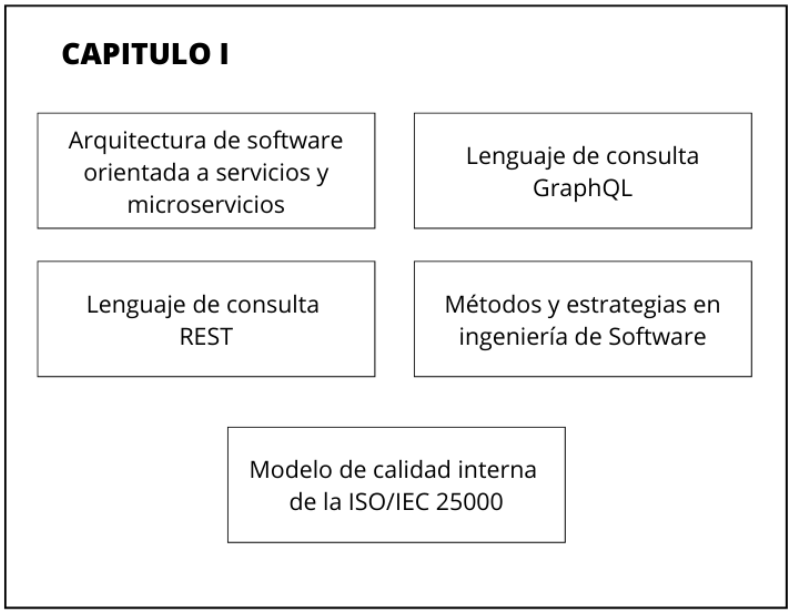


Fig. 5. Mapa conceptual capítulo I

Fuente: Propia

1.1. Arquitectura de software orientada a servicios y microservicios

1.1.1. Arquitectura orientada a Servicios

La arquitectura orientada a Servicios SOA (Service Oriented Architecture), es un modelo de organización de los componentes del software, en el cual, sus recursos son distribuidos e independientes los cuales pueden estar controlados por una variedad de dominios. Esta separación entre los recursos llamados también servicios, es uno de los objetivos de SOA. El uso de esta estandarización logra que los recursos de una aplicación se pueda usar por cualquier usuario que necesite de este servicio (Snell et al., 2001).

El manejo de este patrón arquitectónico incluye proveedores y consumidores de los servicios, la facilidad que ofrece esta arquitectura es que tanto las aplicaciones proveedores, como clientes de los servicios pueden estar en distintos lenguajes o plataformas, sin perder la eficiencia y objetivo de su uso, en donde, se logra la interoperabilidad e integración entre varios sistemas (Albertos, 2018).

1.1.2. Arquitectura orientada a Microservicios

La arquitectura orientada a Microservicios es un modelo en el cual existen una variedad de pequeños servicios los cuales tienen sus propios procesos, siendo servicios independientes y ligeros, esto genera una descentralización en la organización permitiendo la escalabilidad de esta sin afectar el desempeño final del conjunto. Todos estos pequeños servicios pueden tener su propia gestión de datos con una tecnología de

almacenamiento y estar en un lenguaje especifico, esta arquitectura garantiza la tolerancia a fallos, pues los servicios son independientes de los otros, permitiendo el funcionamiento del resto al tener un fallo en un servicio específico (Albertos, 2018).

Características según Newman (2015):

- Usa tecnología heterogénea.
- Servicios independientes.
- Gran escalabilidad.
- Despliegue independiente.

La Fig. 6 muestra, un ejemplo de una arquitectura microservicios, la cual consta de 4 capas:

- La primera capa, la capa de Clientes es donde se podrá realizar el consumo de todos los servicios y mostrar en una interfaz a los usuarios finales.
- La segunda capa, la capa API es donde se realizará las peticiones del servicio a al cliente a través de la comunicación HTTP.
- La tercera capa, la capa de orquesta de contenedores es donde se tendrá la lógica del negocio, en la que se tendrá varios microservicios en diferentes idiomas y frameworks.
- La cuarta capa, la capa del almacenamiento de datos es donde se guardará la información sin la necesidad de tener una sola base de datos para los microservicios.

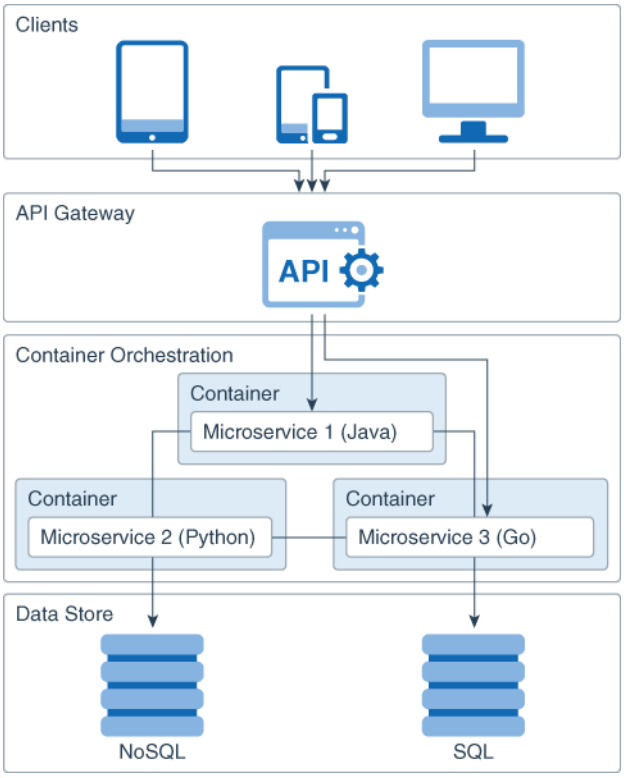


Fig. 6. Arquitectura de microservicios

Fuente: https://docs.oracle.com/es/solutions/learn-architect-microservice/index.html

1.2. Lenguaje de consulta GraphQL

GraphQL es un lenguaje de consulta con una sintaxis flexible para describir requisitos de datos, además se lo menciona como un marco de desarrollo de soluciones del lado del servidor para posteriormente ser usado en el lado del cliente (Vogel et al., 2018). El lenguaje de consulta nace en Facebook como alternativa a los aplicativos REST debido a la sobrecarga de datos que estos presentaban, muchos usuarios al consultar información recibían datos innecesarios y es así como se desarrolló internamente desde el año 2012 y se hace público en 2015 como un borrador (Bryant, 2017).

1.2.1. Schema

GraphQL se describe por medio de un esquema (schema) en el cual se especifica todos los tipos y operaciones que admite este marco de trabajo, en el contexto que muestra se verifica que la solicitud recibida sea sintácticamente correcta, inequívoca y sin errores (Vogel et al., 2018). Un esquema tiene 2 divisiones: consultas (queries) y mutaciones (mutation) (Sayago H. et al., 2019).

Como se puede observar en la Fig. 7, el esquema en GraphQL contiene 3 tipos de operaciones llamados: query, mutation y subscription.

```
schema {
    query : Query,
    mutation: Mutation,
    subscription: Subscription
}
```

Fig. 7. Ejemplo de un esquema en GraphQL
Fuente: (Vogel et al., 2018)

1.2.2. Operaciones

Queries.

La consulta en GraphQL es un conjunto de campos jerárquicos compuesto por datos que se desea obtener, por ende el resultado es solo el dato especifico que se consulta, siendo de gran ayuda para evitar una sobrecarga de datos que se presenta en REST, llegando a ser de esta manera más eficiente (Vazquez-Ingelmo et al., 2017).

La Fig. 8 muestra una consulta (query) de tipo persona, en la que se necesita como parámetro el id de la persona, a su vez se describe los parámetros que se desea consultar, como en este caso del ejemplo se encuentra: name, age, height, weight, parents y el nombre de estos parents.

```
{
    person(id: some_id)
    {
        name,
        age,
        height,
        weight,
        parents
        {
            name
        }
    }
}
```

Fig. 8. Ejemplo de una consulta en GraphQL
Fuente: (Vazquez-Ingelmo et al., 2017)

La estructura de las respuestas está compuesta del mismo orden en el que se realizó la consulta como se puede observar en la Fig. 9; dando libertad al cliente para organizar los datos que desea obtener, y que, además, podría cambiarlos si requiere con otros campos, cabe mencionar que las consultas utilizan un solo end point para realizar todas las operaciones de manipulación de datos. En contraste, REST utiliza varios end points (URLs) para realizar diferentes tipos de consultas y operaciones (Vazquez-Ingelmo et al., 2017).

Fig. 9. Ejemplo de una respuesta en GraphQL
Fuente: (Vazquez-Ingelmo et al., 2017)

Mutations.

La forma como se modifica los datos en el lado del servidor en GraphQL es mediante las mutaciones, en la cual necesita un tipo de entrada, aquí es donde se pasara los valores u objetos en los campos, el tipo de entrada ya sea para valores normales u objetos, la única diferencia es que se necesita la palabra reservada input en lugar de type (The GraphQL Foundation, 2021).

La Fig. 10 muestra un ejemplo de mutación para crear una persona, en donde, se pasa los datos de entrada con el objeto input y los respectivos parámetros que requiere para poder realizar correctamente la operación mutation de creación.

```
mutation {
  createPerson(input: {
    name: "Alan",
    age: 32,
  }) {
    id
  }
}
```

Fig. 10. Ejemplo de una mutación en GraphQL
Fuente: (Vazquez-Ingelmo et al., 2017)

Subscriptions.

Las subscripciones es la información o datos que se desea enviar un cliente con una suscripción al servidor, un servidor tiene una lista de suscripciones que admite evidenciando los eventos a los que se puede suscribir. Al realizar un mutation se recibe los datos que se especifican, pero con la suscripción envía una actualización como evento después de cada dato enviado al servidor (Schafer & Kuenzel, 2015).

Como se observa en la Fig. 11, es un ejemplo de una operación mutation normal basado en historias las cuales tienen un me gusta con un contador y la sentencia de like a manera de texto. Cabe resaltar que con esta operación no se realizara actualizaciones al servidor a manera de eventos para comunicar al cliente de los cambios realizados a la historia.

```
mutación StoryLikeMutation ( $ entrada : StoryLikeInput ) {
  storyLike ( entrada : $entrada ) {
    historia {
      Me gusta { contar }
      likeSentence { texto }
    }
  }
}
```

Fig. 11. Ejemplo mutación StoryLike
Fuente: (Schafer & Kuenzel, 2015)

La Fig. 12 muestra cómo sería un ejemplo con la operación suscripción, la cual es idéntica al mutation, a diferencia del nombre de la operación, en la que realizara una actualización a manera de eventos al servidor para poder comunicar al cliente los cambios realizados a la historia.

```
suscripción StoryLikeSubscription ( $ entrada : StoryLikeSubscribeInput ) {
  storyLikeSubscribe ( entrada : $entrada ) {
    historia {
      Me gusta { contar }
      likeSentence { texto }
    }
  }
}
```

Fig. 12. Ejemplo subscribe StoryLike
Fuente: (Schafer & Kuenzel, 2015)

1.3. Estilo Arquitectónico REST

REST(Representational State Transfer) se lo define como un estilo arquitectónico, el cual, se introdujo por primera con Roy T. Fielding en su tesis doctoral, quien a su vez fue uno de los principales arquitectos del proyecto de HTTP y también autor de la sintaxis genérica del identificador uniforme de recursos (URI) (Xinyang et al., 2009).

Las API REST en la actualidad son populares para el diseño de arquitecturas orientadas a servicios, especialmente con el auge de la computación en la nube y la distribución de los sistemas, el cual REST soporta y da cabida a una mayor escalabilidad (Haupt et al., 2017).

1.3.1. Recursos

Los recursos son la parte fundamental de una plataforma web, y estos se los puede definir como un objeto físico o abstracto el cual puede ser referenciado siempre y cuando este genere importancia, es por eso que un recurso puede ser un objeto abstracto que puede almacenarse en una computadora y representarse con un flujo de bits (Xinyang et al., 2009).

En REST la representación de los recursos vienen acompañados de un identificador único, es así como pueden existir una infinidad de recursos en la web, gracias

al identificador universal URI, en la web esta representados por el localizador de recursos uniforme URL como por ejemplo http://google.com (D. Kumar, 2018).

1.3.2. Endpoints

Kumar (2019) explica que los Endpoints son el URI / URL en donde el cliente de una aplicación puede acceder al servicio API. En la Fig. 13 se puede observar los endpoints de los diferentes tipos de operaciones HTTP a manera de ejemplo.

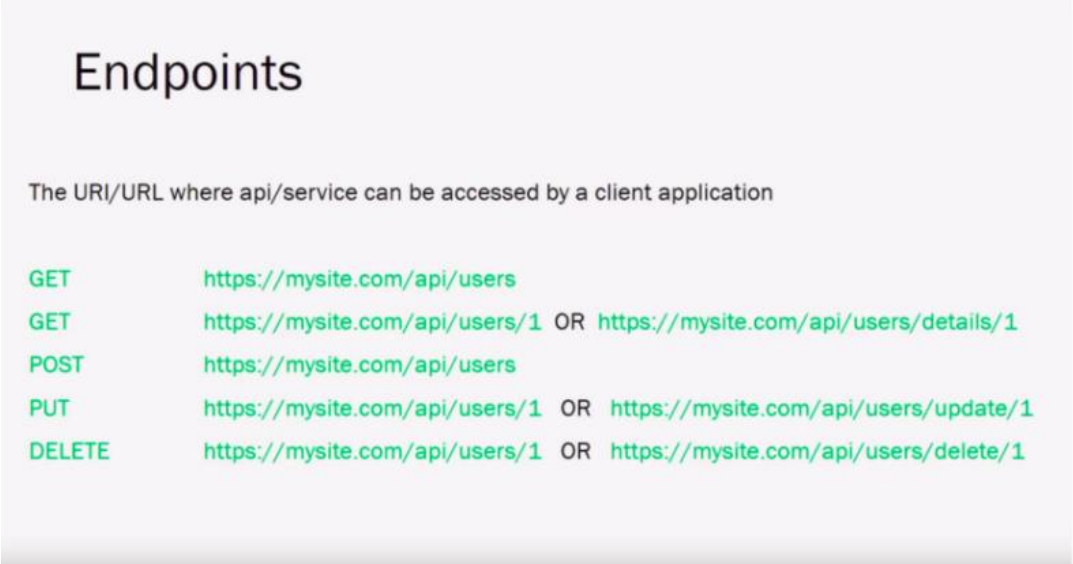


Fig. 13. Endpoints
Fuente: (S. Kumar, 2019)

1.3.3. Estado

Xinyang (2009) define 2 tipos de estados de la información cuando se realiza una solicitud por parte del cliente:

- Estado del recurso: Información sobre un recurso el cual permanece en el servidor.
- Estado de la aplicación: Información sobre la ruta que el cliente toma a través de la aplicación y que permanece en el cliente.

El estado cambia conforme el cliente realice una solicitud, y el servidor procesa y envía una respuesta, las respuestas contienen un código de estado como se puede observar en la Fig. 14 en la que se realiza una petición a través de un Url + Verb el cual es la operación HTTP, y esta recibe como respuesta el estado del código + la respuesta, los estados se dividen en 5 grupos como lo dice Kumar (2019):

1. Informativo 1XX: No se encuentra muy a menudo.
2. Exitoso 2XX: 200 OK (obteniendo una página).
3. Redirección 3XX: 304 no modificado (el CSS en caché no ha cambiado).

4. Error del cliente 4XX: 404 no encontrado (la página no existe).

5. Error del servidor 5XX: 500 Error interno del servidor (error genérico).

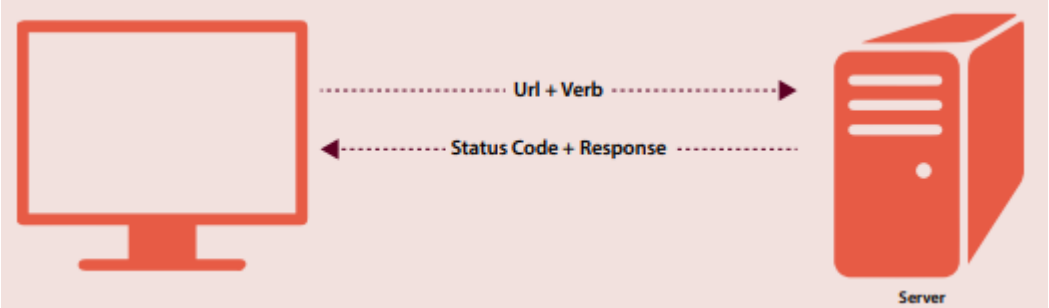


Fig. 14. Estado

Fuente: (D. Kumar, 2018)

1.4. Operaciones

En REST existe una clara base de HTTP como nos dice Xinyang (2009) puesto que es el protocolo de transferencia de estado, y también da los recursos para poder operarlos, las 4 operaciones básicas de HTTP utilizadas en REST son:

- GET: Recupera el estado actual de un recurso en alguna representación (lectura de datos).
- POST: Transfiere un nuevo estado a un recurso (creación de datos).
- PUT: Actualiza un nuevo recurso (actualización de datos).
- DELETE: Elimina un recurso existente (eliminación de datos).

1.4.1. Representación

La forma que los recursos se muestran a los clientes depende de la capacidad que tienen los clientes, REST permite utilizar cualquier formato para representar los recursos, pero los más usados son JavaScript Object Notation (JSON) y Extensible Markup Language (XML).

1.5. Proceso, métodos y estrategias en ingeniería de Software

La realización de software requiere de un proceso en el que se describe pasos a seguir y actividades a realizar para poder tener un producto final, tal como se observa en la Fig. 15, lo cual requiere de mucho tiempo de desarrollo, y es por eso que de modelos tradicionales como: cascada, incremental, espiral, se pasó a enfoques metodológicos más agiles como SCRUM, al ser una tarea compleja que requiere de un gran equipo dependiendo del proyecto, y de tiempo, es por eso que los investigadores no optan mucho por la experimentación con

humanos pese a que sea crucial la evaluación de productos, herramientas, procesos en humanos (Wohlin et al., 2012).

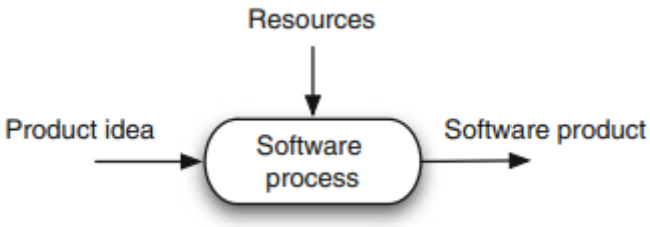


Fig. 15. Proceso de Software
Fuente: (Wohlin et al., 2012)

Los métodos (procedimientos a seguir con un fin) y estrategias (acciones a tomar para llegar a un fin) de software según Guevara, Vega et al. (2021) tienen gran interés en los investigadores en Ingeniería de Software (IS), debido a una creciente necesidad de validar científicamente resultados realizados en estudios y trabajos de investigación, por lo que la experiencia del investigador sobre los métodos y las estrategias a tomar tiene un gran peso sobre el diseño del estudio, en este sentido se toma como referencia a investigaciones de disciplinas diferentes a la IS para llenar el hueco de la poca experticia en una disciplina en crecimiento y con falta de evidencia en investigaciones empíricas experimentales.

1.5.1. Métodos de investigación de Ingeniería de software

Wohlin et al (2012) habla sobre 4 métodos, los cuales son:

- Científico: Se observa el mundo y se construye un modelo basado en la observación, por ejemplo, un modelo de simulación.
- Ingeniería: Se estudian las soluciones actuales y se proponen cambios luego de ser evaluado.
- Empírico: Se propone y evalúa un modelo mediante estudios empíricos, ejemplo, encuestas, estudios de caso o experimentación.
- Analítico: Se propone una teoría formal y luego se compara con teorías empíricas.

1.5.2. Estrategias empíricas

Encuesta.

Una encuesta es un sistema para recopilar información sobre personas en la cual se busca comparar o describir los conocimientos, actitudes o comportamientos actuales. Principalmente se recopila la información por medio de encuestas o entrevistas, estos datos pueden ser cualitativos o cuantitativos (Wohlin et al., 2012).

Estudio de caso.

Es una investigación empírica que utiliza múltiples fuentes de evidencia para investigar un tema en particular de un fenómeno de la ingeniería de software, estos se utilizan para proyectos, actividades o asignaciones, los datos resultantes se recopilan y se utilizan para análisis estadísticos (Wohlin et al., 2012).

Experimento controlado.

En el experimento controlado se trata 2 grupos de participantes los cuales serán comparados de manera aleatoria, se tratará 2 temas, por lo que solo existe una variante diferente que es el tema, el diseño y ejecución será el mismo para los dos grupos, por lo que otros factores externos como la estacionalidad, impacto de otros cambios en el producto, movimientos de la competencia, etc. Se distribuyen uniformemente en el tratamiento, por lo que cualquier diferencia en las métricas resultantes se debe al cambio del tema (Fabijan et al., 2018).

1.5.3. Experimentos en la ingeniería de software

Los experimentos en la ingeniería de software están para realizar una investigación sobre la causa-efecto entre: procesos, métodos, técnicas, lenguajes, herramientas, etc., como también para revisar los resultados con sus variables de tiempo, eficacia, calidad, eficiencia, etc. (Kampenes et al., 2007).

Los experimentos son aplicados en situaciones que el experimentador tiene control de manera directa, precisa y sistemática sobre al menos un tratamiento o las variables, las cuales se desarrollan en laboratorios con humanos y software, también se toma en cuenta si el estudio presenta resultados descriptivos, correlacionales, de causa efecto (generalmente se realiza con factor humano), cuando un estudio presenta un ambiente de laboratorio controlado o condiciones normales (generalmente con expertos), y este tiene un estudio observacional, se suele incluir análisis cualitativos que al menos incluye una forma de entrevista (Guevara-Vega et al., 2021).

Métricas.

Las métricas son una herramienta importante en el diseño de experimento debido a que es de ayuda para establecer los objetivos que tendrá la investigación, proyecto, etc. Y para el establecimiento de estos objetivos será necesario la formulación de preguntas a ser resueltas en la investigación, todo este proceso ayudará a tener un horizonte y tener un enfoque practico en lo que se realiza. También tenemos que estas métricas se las etiquetan como ‘Criterios de evaluación’, ya que

van ligado al objetivo y si este resulta ser completado o no, y estas pueden ser más intuitivas para ser medidas y tener unos resultados más formales (Fabijan et al., 2017).

Proceso.

Un proceso consta de fases o pasos a seguir como se observa en la Fig. 16, los cuales se usan como verificación de si se realiza de manera adecuada el experimento, el punto de partida de un experimento comienza con la idea sobre que investigar y como será evaluada la misma, así como también definir las métricas de manera formal como se lo menciona en el apartado anterior sobre Métricas (Wohlin et al., 2012).

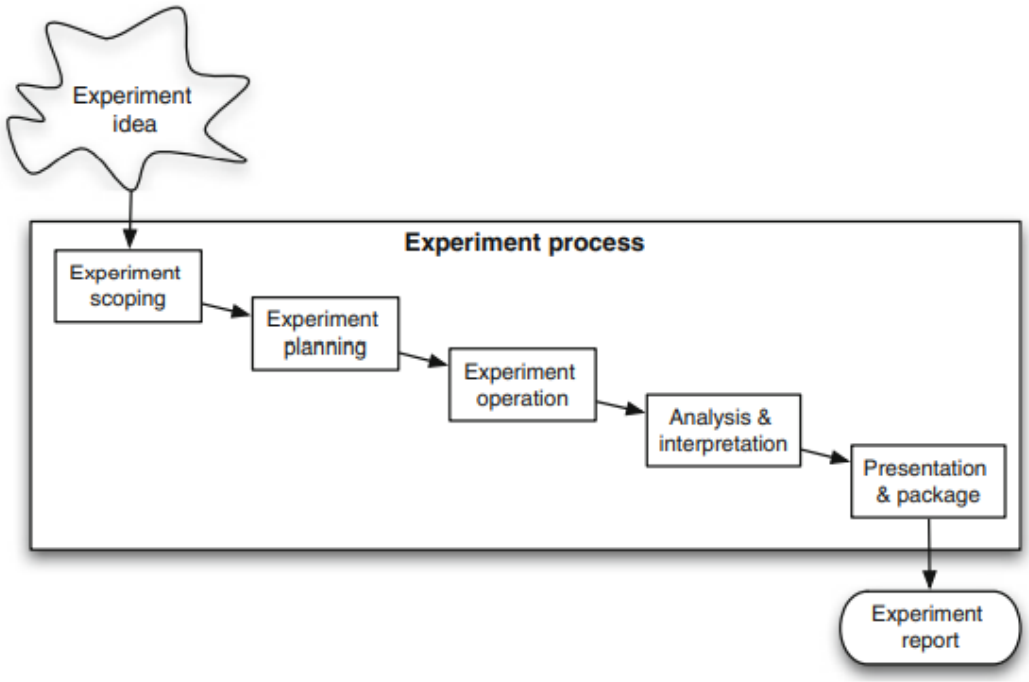


Fig. 16. Fases del proceso de un experimento
Fuente: (Wohlin et al., 2012)

- ***Diseño.***

Alcance: El alcance (Fig. 17, Sección 7) permite ver los límites de la investigación y poder formular la hipótesis, además de definir el objetivo a resolver del experimento (Wohlin et al., 2012). Para capturar el alcance Wohlin (2012) define un marco:

- ¿Qué se estudia?
- ¿Cuál es la intención?
- ¿Qué efecto se estudia?
- ¿La perspectiva de quién?

- ¿Dónde se realiza el estudio?

Planificación: La planificación (Fig. 17, Sección 8) forma las bases del experimento en el cual se selecciona a detalle el personal, entorno en el que se llevara a cabo. Además, se establece formalmente la hipótesis, las variables dependientes, independientes, el orden aleatorio de los sujetos de prueba. Un punto importante es la selección de los instrumentos del experimento, como tecnologías a usar, herramientas, material de capacitación y las métricas definidas para poder evaluar

(Wohlin et al., 2012).

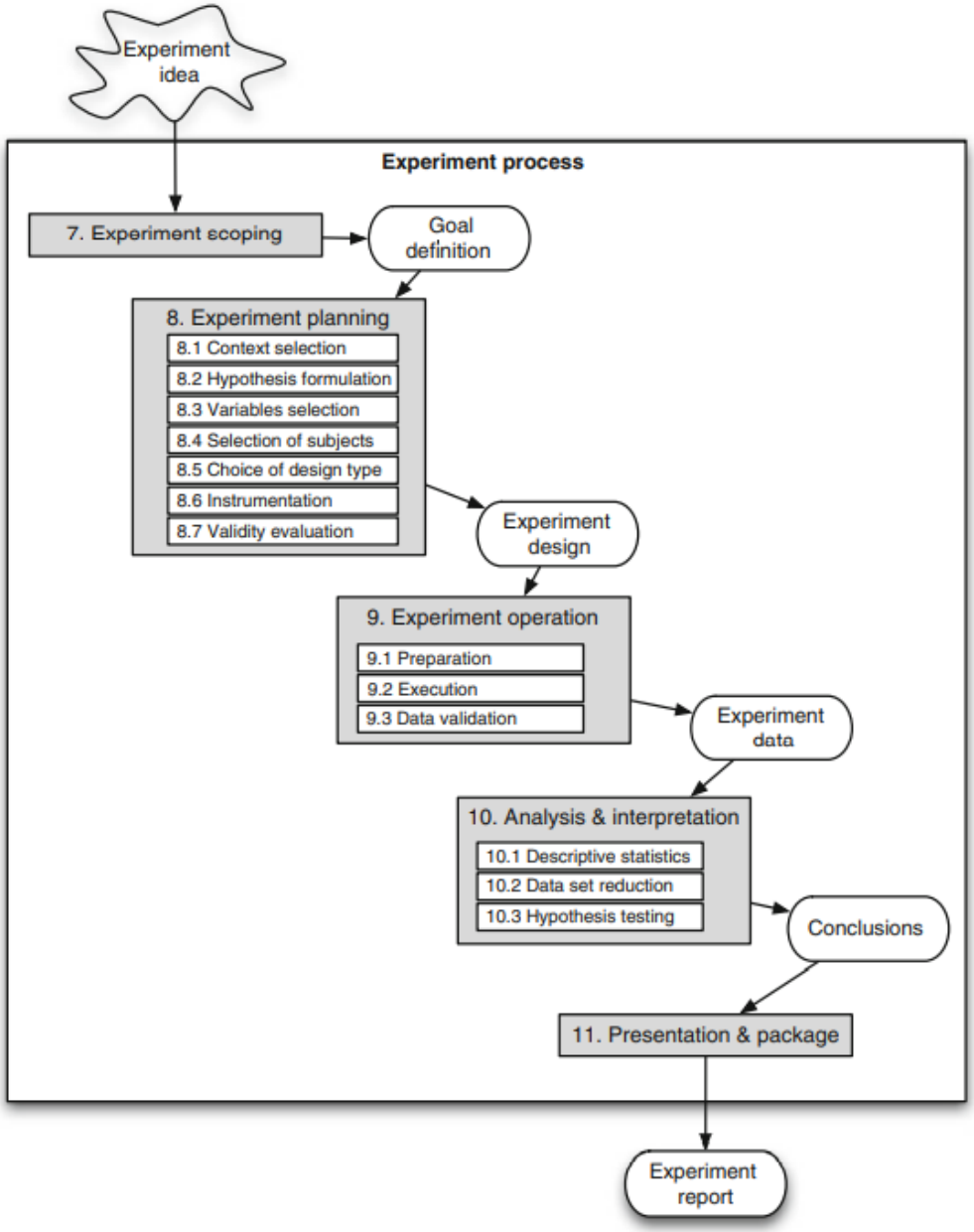


Fig. 17. Proceso y secciones del experimento
Fuente: (Wohlin et al., 2012)

- ***Operación.***

La operación (Fig. 17, Sección 9) consta de 3 pasos: preparación, ejecución y validación de datos. En la preparación tenemos listos los materiales necesarios ya mencionados en el apartado del Diseño, planificación y se informa a los sujetos de prueba sobre cómo se realizara el experimento, seguido de esto se ejecuta y revisa que todo el experimento este de acuerdo a la planificación, por último se recopila los datos (Wohlin et al., 2012).

- ***Análisis e interpretación.***

El análisis e interpretación (Fig. 17, Sección 10) de los datos consta como paso principal el tratamiento, en el que a juicio del investigador podrá eliminar o reducir la información por métodos específicos para la realización de esto. Posteriormente se realizara las pruebas de hipótesis y los resultados se evaluaran con las métricas establecidas para la investigación, dando así una aceptación o rechazo de las hipótesis (Wohlin et al., 2012).

- ***Presentación y empaque.***

La presentación y empaque (Fig. 17, Sección 11) de los resultados incluye la documentación de los resultados para su posterior publicación (Wohlin et al., 2012).

Pasos para desarrollar un experimento controlado.

Andrew J. Ko et al. (2015) define una serie de pasos para realizar un experimento controlado, el cual consta de:

- ***Reclutamiento.***

El reclutamiento (Fig. 18, Sección 1) se lo puede realizar de distintas formas dependiendo de a quién va dirigida la investigación, por ejemplo: cuando la investigación se realiza con el objetivo de mejorar un proceso interno de una empresa, este reclutamiento se lo realizara entre los trabajadores de esta, caso contrario si es para los clientes, se intentará reclutar a clientes habituales para realizar la investigación.

- ***Selección.***

La selección (Fig. 18, Sección 2) va ligada con el reclutamiento debido a que se debe seleccionar el público al que va dirigido, si la investigación busca

comparar 2 lenguajes de programación será más fácil buscar en empresas de desarrollo o como muchas veces se lo realiza en la academia, con estudiantes familiarizados con el área de desarrollo.

- ***Consentimiento.***

El consentimiento (Fig. 18, Sección 3) va ligado con la probabilidad de que la investigación se realice en una institución académica y sea necesario recibir algún tipo de aprobación de los sujetos de prueba, por lo que se envía un formulario sobre los aspectos éticos del diseño del experimento y se pida el consentimiento a los participantes (Ko et al., 2015).

- ***Procedimiento.***

El procedimiento (Fig. 18, Sección 4) determina lo que se realizara desde el comienzo del experimento hasta el final, en este se detalla los materiales a usar y lo que realizara el participante (Ko et al., 2015).

- ***Medidas demográficas.***

Las medidas demográficas (Fig. 18, Sección 5) son formas comunes de recopilar información y medir variables demográficas de los participantes (Ko et al., 2015).

- ***Asignación grupal.***

La asignación grupal (Fig. 18, Sección 6) generalmente se la realiza de forma aleatoria.

- ***Capacitación.***

La capacitación (Fig. 18, Sección 7) se la realiza con el fin de enseñar a los participantes a cómo usar las herramientas para posteriormente realizar las tareas que se les indique (Ko et al., 2015).

- ***Tareas.***

Las tareas (Fig. 18, Sección 8) son seleccionadas por el investigador para posteriormente las realice el participante (Ko et al., 2015).

- ***Medición de resultados.***

La medición de resultados (Fig. 18, Sección 9) van de acuerdo a las métricas seleccionadas para evaluarlas y medirlas (Ko et al., 2015).

- ***Informar y compensar.***

Informar y compensar (Fig. 18, Sección 10), este paso es opcional, e incluye brindar información sobre el objetivo del estudio (Ko et al., 2015).

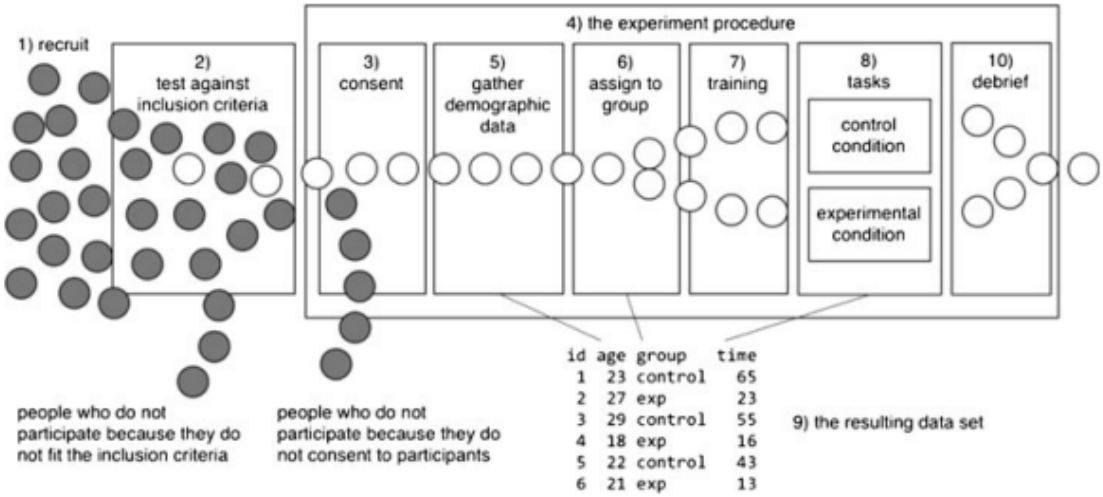


Fig. 18. Diseño canónico experimento
Fuente: (Ko et al., 2015)

1.6. Modelo de calidad interna de la ISO/IEC 25000

La norma ISO/IEC 25000 o como también se la llama SquaRE (Requisitos y evaluación de calidad de productos de software) están conformadas por las normas ISO/IEC 9126 e ISO/IEC 14598, las cuales se utilizan como estándar para crear modelos, métricas, procesos y herramientas de evaluación de calidad de software como producto (Roa et al., 2015).

1.6.1. División modelo SQuaRE

La ISO (2015) consta de 5 divisiones como se muestra en la Fig. 19.

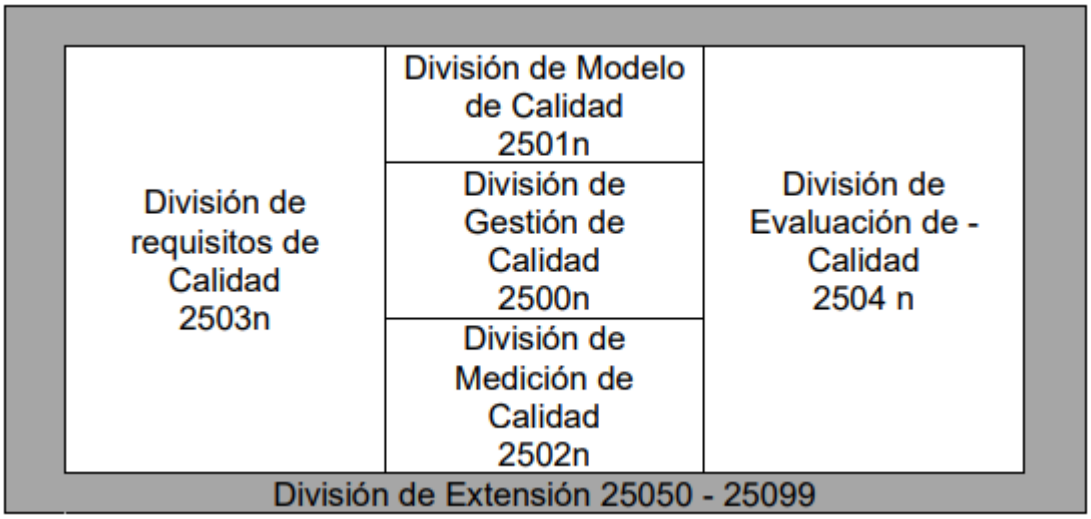


Fig. 19. División ISO
Fuente: (ISO/IEC 25010, 2015)

- **ISO/IEC 2500n – División de Gestión de Calidad.** Las normas que conforman esta división definen los modelos, términos y definiciones comunes referidos a las otras normas SQuaRE.
- **ISO/IEC 2501n – División de Modelo de Calidad.** Las normas que conforman esta división presentan modelos de calidad detallados para sistemas de computación y productos de software, calidad en uso y datos.
- **ISO/IEC 2502n – División de Medición de Calidad.** Las normas que conforman esta división incluyen un modelo de referencia de medición de calidad de producto de software, definiciones matemáticas de las medidas de calidad y una guía práctica de su aplicación.
- **ISO/IEC 2503n – División de Requisitos de Calidad.** Las normas que conforman esta división ayudan a especificar los requisitos de calidad, en base a modelos de calidad y medidas de calidad.
- **ISO/IEC 2504n – División de Evaluación de Calidad.** Las normas que conforman esta división proveen requisitos, recomendaciones y directrices para evaluar el producto de software, ya sean ejecutados por evaluadores, compradores o desarrolladores.
- **ISO/IEC 25050 – 25099 División de Modelo de Calidad.** Las normas que conforman esta división incluyen requisitos de calidad de software listos para la comercialización de paquetes y formatos comunes para la industria para reportes de usabilidad.

1.6.2. Calidad de Software

Como menciona Vaca y Jácome (2018) en el que según varios autores definen a la calidad de software como “el conjunto de atributos deseables que posee un producto

de software, los cuales son medibles (cuantitativa o cualitativamente), permitiendo hacer comparaciones para conocer si se cumple con las expectativas del cliente o no". Entonces se puede percibir a la calidad de software como el cumplimiento de requisitos los cuales se pueden medir con el fin de satisfacer al cliente, entonces la calidad de software es aquella que brinda un mejor producto conforme a parámetros medibles que enaltecen el valor final.

1.6.3. Modelo de Calidad

Según la ISO (2020) un modelo de calidad es un "conjunto definido de características, y de las relaciones entre ellas, que proporciona un marco de trabajo para especificar los requerimientos de la calidad y para evaluar la calidad".

Roa (2015) nos ayuda con la definición de las características de la calidad de producto de software, las cuales se observan en la Fig. 20:

Fig. 20. Modelo de calidad del producto software ISO/IEC 25010
Fuente: (Rodríguez & Piattini, 2015)

- **Adecuación funcional.** Permite medir la capacidad que tiene un producto de software para proveer las funciones que satisfacen requerimientos explícitos e implícitos cuando el software se usa en determinadas condiciones.
- **Rendimiento.** Es el comportamiento del sistema: funcionalidad, capacidad, utilización de recursos y respuesta temporal. Dentro de sus características se encuentra que el sistema requiere la utilización de un mínimo de recursos (por ejemplo, tiempo de CPU) para ejecutar una tarea determinada.
- **Compatibilidad.** Es el proceso en el cual dos o más sistemas intercambian información y llevan a cabo funciones requeridas en cuanto a su entorno hardware o software compartido.

- **Usabilidad.** Algunas de las características que la conforman son: comprensibilidad, aprendibilidad, operabilidad, atractivo, cumplimiento de usabilidad, capacidad de aprendizaje, capacidad de ser usado, protección contra errores de usuario, accesibilidad. También fácil de usar, fácil de aprender, atractivo para el usuario, conforme a normas, uso intuitivo.
- **Fiabilidad.** Madurez, toleración a defectos, recuperabilidad, cumplimiento de fiabilidad. En determinadas condiciones, el software mantendrá su capacidad funcional a lo largo de un periodo de tiempo.
- **Seguridad.** Capacidad de proteger la información y los datos de manera que no puedan ser leídos o modificados por personas o sistemas no autorizados. La autenticidad, la confidencialidad y responsabilidad son sus principales características.
- **Mantenibilidad.** Es la medida del esfuerzo requerido para realizar cambios en los componentes de un sistema de manera efectiva y eficiente. Algunas de sus características son analizabilidad, modificabilidad, estabilidad, testeabilidad, cumplimiento de mantenibilidad, modularidad.

- **Portabilidad.** Es la capacidad del software de ser transferido a un nuevo entorno (software, hardware, organización). Es fácil de instalar y desinstalar, además permite ser adaptado de forma efectiva a diferentes entornos de hardware o software.

1.6.4. Ciclo de vida de la calidad del producto de software

La calidad de software cumple un ciclo como también un producto de software, a continuación, en la Fig. 21 se observa un gráfico que detalla el ciclo.

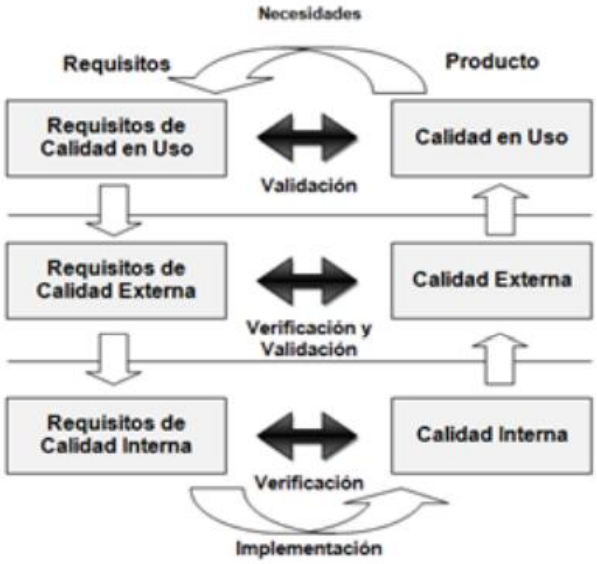


Fig. 21. Ciclo de vida de la calidad de software
Fuente: (Vaca & Jácome, 2018)

Vaca y Jácome (2018) nos describe las tres fases del ciclo del producto de software:

- **Calidad Interna:** producto de software en desarrollo.
- **Calidad Externa:** producto de software en funcionamiento.
- **Calidad en Uso:** producto de software en uso.

1.6.5. Facilidad de aprendizaje

La medida de facilidad de aprendizaje es utilizada para evaluar en que grado un producto de software puede ser usado por usuarios con eficacia, eficiencia, ausencia de riesgo y satisfacción en un contexto de uso especifico, cabe señalar que esta medida es usada tanto para calidad interna como externa del software. La sub característica utilizada es la completitud de la guía de usuario, la cual tiene la interpretación de uso de ayuda del software como en línea, video de guía, sistema de instrucción, etc. Se tomo en cuenta la realización de la guía para el uso de las herramientas y su posterior evaluación (ISO/IEC 25023, 2020).

CAPITULO II

2. DESARROLLO

2.1. Introducción

En este capítulo se definió y realizo el desarrollo del experimento controlado para comparar la facilidad de aprendizaje entre GraphQL y REST mediante un modelo de calidad interna basada en las ISO/IEC 25000. A continuación, en la Fig. 22, se muestra la estructura del contenido del capítulo.

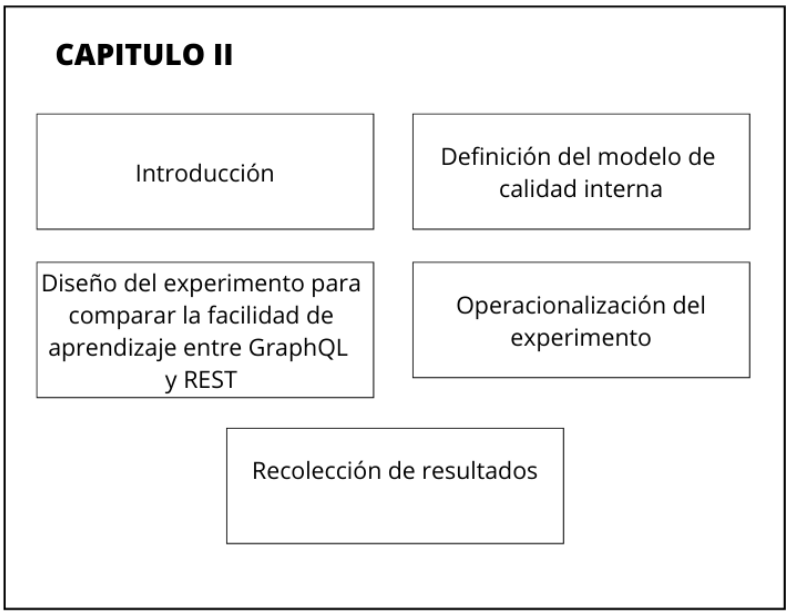


Fig. 22. Mapa conceptual Capitulo II
Fuente: Propia

2.2. Definición del modelo de calidad interna

El modelo de calidad interna se definió con los investigadores del experimento: Ing. Cathy Guevara, MSc. Antonio Quiña y Fernando Chulde, es decir, este trabajo se realizó en el contexto del desarrollo del proyecto de investigación interno “Gobierno de arquitecturas híbridas REST y GraphQL mediante contratos” de la carrera de Software, concretamente, el MSc. Antonio Quiña en el rol de definir la calidad, la Ing. Cathy Guevara en calidad de definir el diseño del experimento.

Se aplicó el modelo de calidad de sistema/producto a partir de la ISO/IEC 25010 y conforme a la necesidad encontrada se seleccionó la característica de la usabilidad. Según la ISO/IEC 25023 la medida de calidad interna la cual es la usada, mide el conjunto de atributos estáticos de un producto de software y si este satisface a las necesidades

establecidas e implícitas del producto o sistema (ISO/IEC 25023, 2020). La estructura utilizada es la que se muestra en la TABLA 1.

TABLA 1. Modelo de calidad interna de Producto/Sistema

Característica	**Sub característica**	**Porcentaje Sub característica**	**Porcentaje Característica**
Usabilidad	Adecuación	0%	100%
	Reconocibilidad	0%	
	Facilidad de aprendizaje	100%	
	Operabilidad	0%	
	Protección contra errores de usuario	0%	
	Estética del interfaz	0%	
	Accesibilidad	0%	

La medición identificada para la sub característica de la facilidad de aprendizaje de la ISO/IEC 25023 se muestra en la TABLA 2.

TABLA 2. Medidas de Facilidad de Aprendizaje

Característica	**Sub característica**	**Nombre función**	**Función de medición**
Usabilidad	Facilidad de aprendizaje	Completitud de la guía del usuario	X = A/B A = Numero de funciones descritas en la documentación del usuario o en la función de ayuda según sea requerido B = Numero de funciones implementadas que se requiere que sean documentadas

Se tomo en cuenta la nota presente en la ISO/IEC 25023 en la que se describe la función de ayuda, esta incluye: ayuda en línea, video de guía operacional, sistema de introducción operacional, etc. Por lo cual se utilizó la capacitación en línea como guía de usuario.

2.3. Diseño del experimento para comparar la facilidad de aprendizaje entre GraphQL y REST

Para diseñar el experimento se utilizó como base la guía de Experimentación en Ingeniería de Software de Wohlin (2012) y se lo realizo en conjunto con el MSc. Antonio Quiña y la Ing. Cathy Guevara, docentes de la Universidad Técnica del Norte, en base a las estrategias empíricas de la ingeniería de Software se realizó la experimentación controlada.

2.1.1. Objetivo

El objetivo del experimento es comparar la calidad interna entre GraphQL y REST con respecto a la facilidad de aprendizaje, la cual es una sub característica de la Usabilidad. El experimento se logró con los siguientes pasos:

1. Enseñar a un grupo de estudiantes de ingeniería en Software sobre: API REST, GraphQL.
2. Realizar un taller a los estudiantes sobre REST y GraphQL.
3. Realizar un deber a los estudiantes aumentando la dificultad sobre las consultas SQL sobre REST y GraphQL.
4. Realizar la evaluación y analizar los resultados obtenidos con el experimento.

2.1.2. Variables

Como se observa en la TABLA 3, la variable independiente fue identificada con los lenguajes de consulta GraphQL y REST.

A su vez como variable dependiente se identificó a la calidad interna (la sub característica de Usabilidad, facilidad de aprendizaje).

TABLA 3. Variables

Variable Independiente	Desarrollo de software con el lenguaje de consultas GraphQL
	Desarrollo de software con estilo arquitectónico REST
Variable Dependiente	Mejorar la Calidad interna (facilidad de aprendizaje) del producto de software

La métrica de facilidad de aprendizaje con respecto a la completitud de la guía de usuario se define tal como indica la ISO 25023 (2020):

$$x = A/B$$

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

2.1.3. Población

Los sujetos del experimento fueron 23 estudiantes de la carrera de Ingeniería en Software de la Universidad Técnica del Norte (Ibarra - Ecuador) los cuales eran estudiantes de la materia de Construcción de Software.

2.1.4. Diseño

El experimento se realizó tomando en cuenta el diseño Crossover (cruzado), el cual es particular por el tratamiento que aplica, en este diseño experimental todos los sujetos de prueba aplican el tratamiento en ordenes diferentes (Vegas et al., 2016).

Vegas y sus colaboradores (2016) explican las tres formas en que se asignan sujetos a los diferentes tipos de tratamientos, en el experimento se utilizó el diseño de pares combinados para la asignación de unidades experimentales, el cual se forma por pareja de sujetos, y cada par se le asigna un tratamiento diferente, además de que el tratamiento tiene 2 niveles como se puede observar en la TABLA 4, la cual tiene objetos diferentes en cada tratamiento, se realizó 3 con objetos, y estos objetos hacen referencia a ejercicios (Factura, Evento, Nota) con la misma lógica de resolución pero diferente estructura de datos.

Se definió de manera aleatoria los pares de sujetos y los tratamientos con las tareas a realizar, el experimento fue realizado en un solo día con duración de 2 horas cada periodo y un descanso de 15 minutos.

TABLA 4. Diseño CrossOver

Secuencia	Periodo	Tratamiento	Objeto
Grupo 1	Sesión 1	REST	Facturas
	Sesión 2	GraphQL	Eventos
Grupo 2	Sesión 1	REST	Facturas
	Sesión 2	GraphQL	Notas
Grupo 3	Sesión 1	REST	Eventos
	Sesión 2	GraphQL	Facturas
Grupo 4	Sesión 1	REST	Eventos
	Sesión 2	GraphQL	Notas
Grupo 5	Sesión 1	REST	Notas
	Sesión 2	GraphQL	Facturas
Grupo 6	Sesión 1	REST	Notas
	Sesión 2	GraphQL	Eventos
Grupo 7	Sesión 1	GraphQL	Facturas
	Sesión 2	REST	Eventos

Grupo 8	Sesión 1	GraphQL	Facturas
	Sesión 2	REST	Notas
Grupo 9	Sesión 1	GraphQL	Eventos
	Sesión 2	REST	Facturas
Grupo 10	Sesión 1	GraphQL	Eventos
	Sesión 2	REST	Notas
Grupo 11	Sesión 1	GraphQL	Notas
	Sesión 2	REST	Facturas
Grupo 12	Sesión 1	GraphQL	Notas
	Sesión 2	REST	Eventos

2.1.5. Tareas

Fase de Preparación.

La fase de preparación conto con una introducción de las tecnologías REST y GraphQL, además de una guía de como instalar los materiales a utilizar, se realizó por parte del MSc. Antonio Quiña profesor de la universidad Técnica del Norte.

Primera Fase

En la primera fase se realizó un ejercicio de desarrollo de una API GraphQL y REST, utilizando el caso de estudio de una tienda de Pizzas. El ejercicio de Pizzas consistió en desarrollar un CRUD básico, en donde se ingresó las entidades pizzas, ingredientes, y la relación entre estas dos. Además, se realizó las funciones de creación, modificación y eliminación de registros.

Cabe mencionar que los sujetos de prueba tuvieron acceso como material de practica un backup de la base de datos de pizza, para que estos puedan simplemente realizar un restore en cada base de datos de los mismos, y proceder a realizar la practica sin problema alguno. En la Fig. 23 se puede observar la estructura de la primera fase.

Fig. 23. Estructura Pizza
Fuente: Propia

- **Pizza – GraphQL**

En la Fig. 24 se puede observar la estructura del proyecto pizza con GraphQL, el cual fue realizado en el lenguaje JavaScript con el framework Node JS.

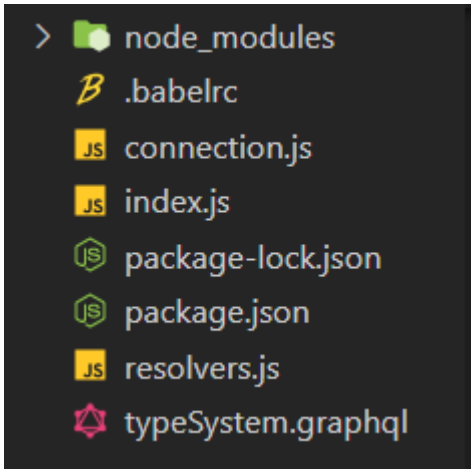


Fig. 24. Estructura del Proyecto GraphQL
Fuente: Propia

El archivo index contiene las importaciones necesarias para utilizar las herramientas de express, graphQL, además del uso de los servicios y métodos necesarios para lanzar el aplicativo, el código se muestra en la Fig. 25.

index.js

```
const express = require("express");
const cors = require("cors");

const { graphqlExpress, graphiqlExpress } = require("graphql-server-express");
const bodyParser = require("body-parser");
const { importSchema } = require("graphql-import"); //Importar un schema
const { makeExecutableSchema } = require("graphql-tools");

const port = 3000;
const endPoint = "/pizza_api";

const typeDefs = importSchema("./typeSystem.graphql");

import resolvers from "./resolvers";

const schema = makeExecutableSchema({ typeDefs, resolvers });

let server = express().use(cors());

server.use(endPoint, bodyParser.json(), graphqlExpress({ schema, rootValue: resolvers }));
server.use("/graphiql", graphiqlExpress({ endpointURL: endPoint }));

server.listen(port, () => {
    console.log("Server on Port: ", port);
    console.log("Graphql localhost:" + port + endPoint);
    console.log("Graphiql: localhost:" + port + "/graphiql");
});
```

Fig. 25. Index Pizza – GraphQL
Fuente: Propia

En el archivo que se muestra en la Fig. 26 se muestra la conexión a la Base de Datos (BDD).

```
const pgPromise= require('pg-promise');

const config={
    host: 'localhost',
    port: '5432',
    database: 'proyecto',
    user: 'postgres',
    password: '256371'
}

const pgp= pgPromise({});
const db= pgp(config);

exports.db=db;
```

Fig. 26. Conexión Pizza – GraphQL
Fuente: Propia

En el archivo typeSystem se muestra el schema en la Fig. 27 de GraphQL, la cual define las operaciones Query y Mutation usadas para el CRUD de Pizza.

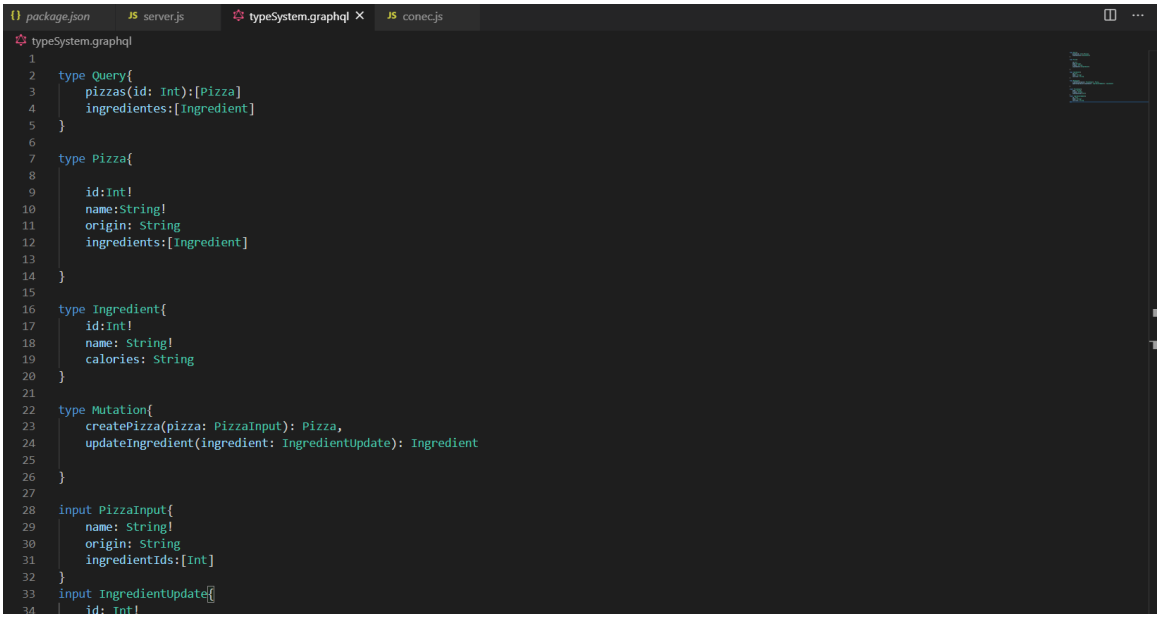


Fig. 27. Schema Pizza
Fuente: Propia

En el archivo resolvers que se observa en la Fig. 28 se realiza las consultas a la BDD con los respectivos queries y mutations del CRUD definido en el typeSystem.

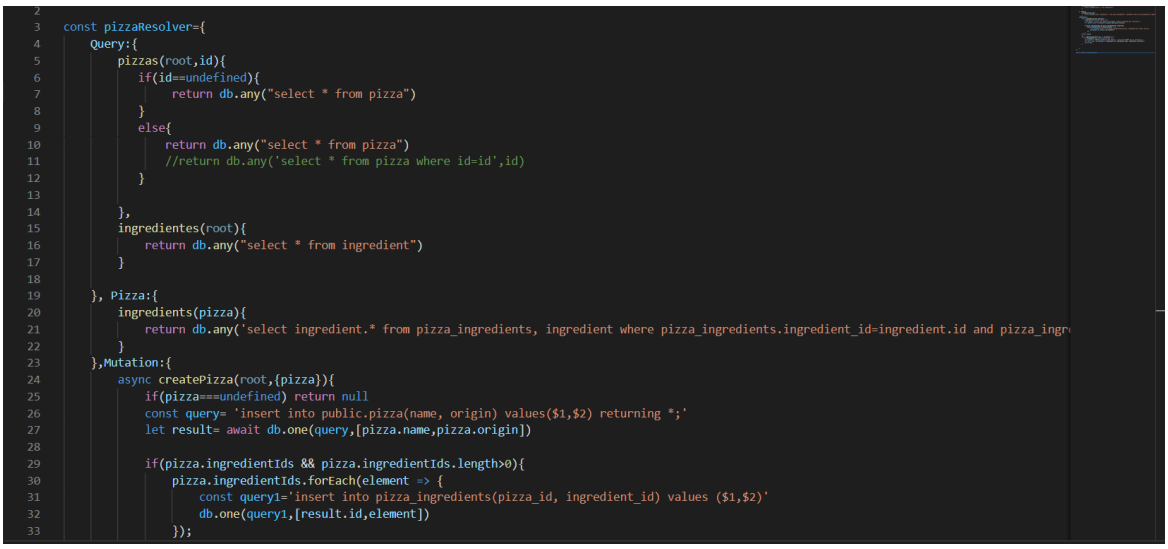


Fig. 28. Resolvers Pizza
Fuente: Propia

Babel es un compilador para JavaScript el cual ayuda a transformar características nuevas de código a código que un navegador antiguo pueda entender, y todo esto es gracias al estándar ECMAScript (ES) el cual desarrolla los nuevos estándares con características y da reportes técnicos para el mejor uso de estas tecnologías (Hernandez, 2021). En la Fig. 29 se observa la configuración para el uso de Babel para transformar código ES.

Fig. 29. Babel Pizza
Fuente: Propia

Ejecución de las operaciones Queries y Mutations de la API GraphQL, mediante la aplicación cliente GraphiQL.

En la Fig. 30 se realizó una consulta query de pizzas, en la que se lista los parámetros: id, ingredient con su name, y origin.

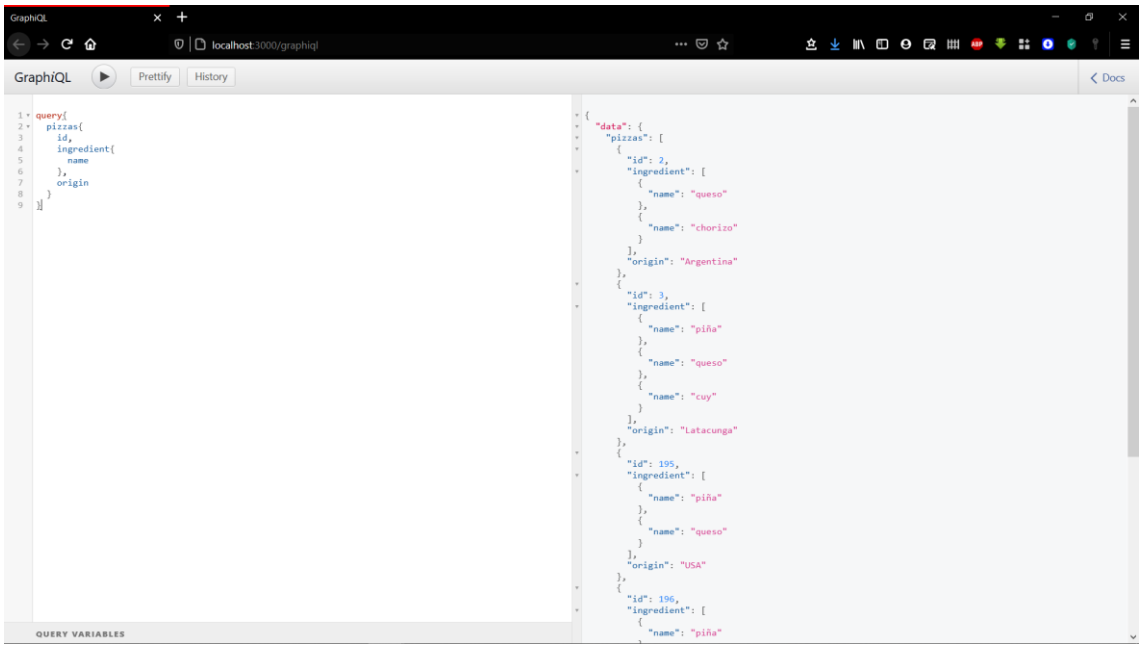


Fig. 30. Query Pizzas

En la Fig. 31 se realizó una consulta query de ingredients, en la que se lista los parámetros: id, name, y calories.

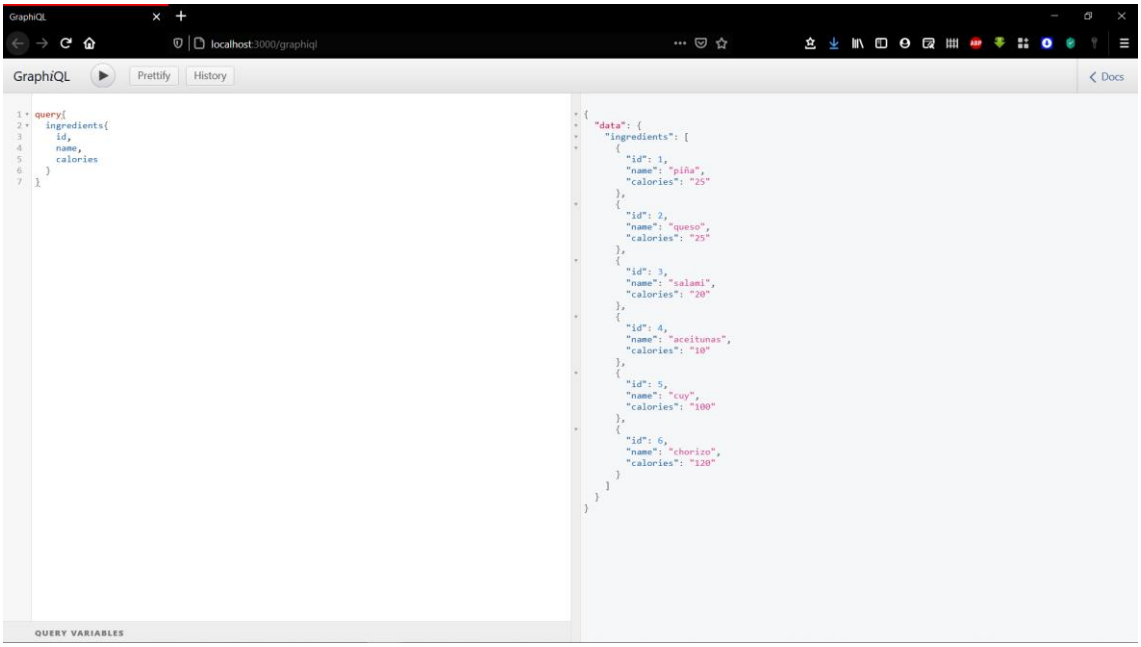


Fig. 31. Query Ingredients
Fuente: Propia

En la Fig. 32 se realizó una operación mutation de actualizar pizzas, en la que se actualiza el objeto pizza y se pide los parámetros: id, name, origin, ingredient con su parámetro id, en los datos de retorno se tendrá el id, name, origin, ingredient con su name.

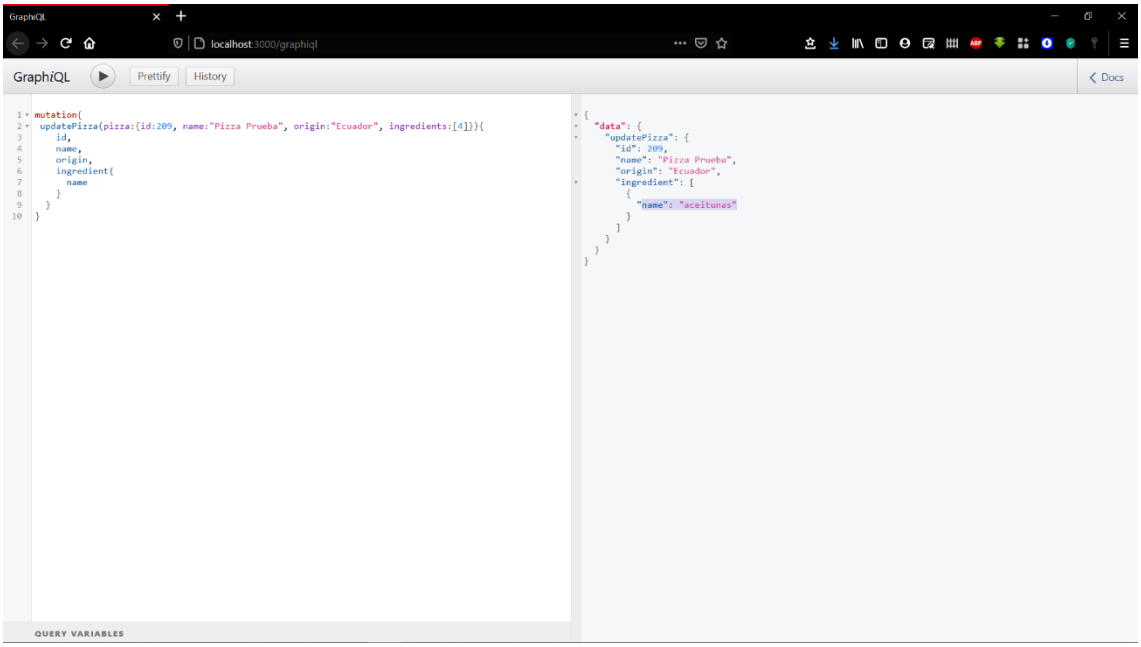


Fig. 32. Mutation Update Pizza
Fuente: Propia

En la Fig. 33 se realizó una operación mutation de actualizar ingredient, en la que se actualiza el objeto ingredient y se pide los parámetros: id, name, calories, en los datos de retorno se tendrá el id, name, calories.

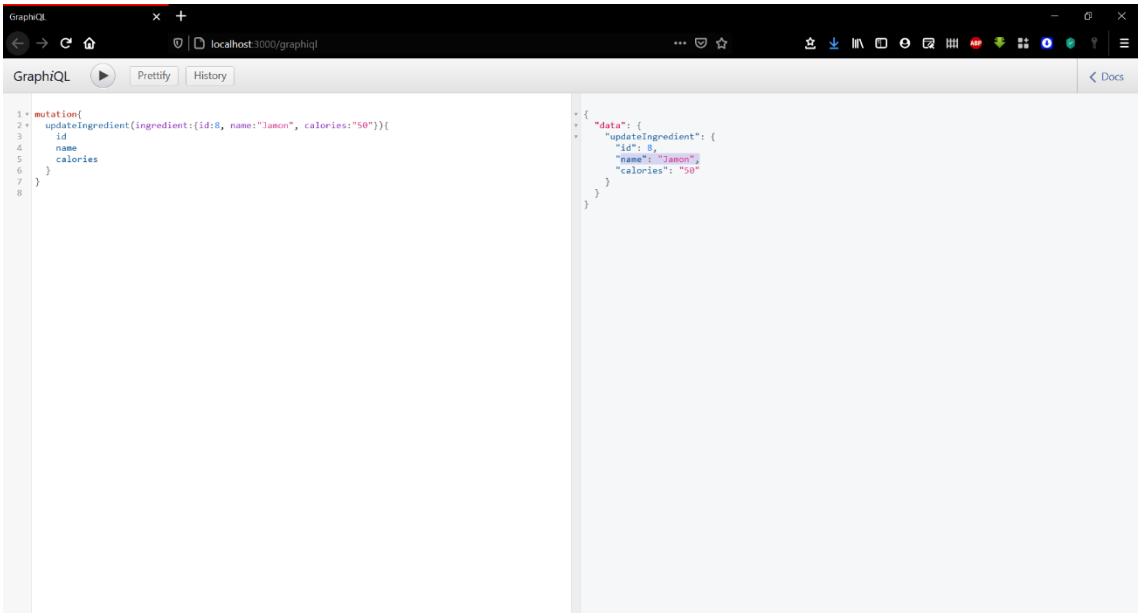


Fig. 33. Mutation Update Ingredient
Fuente: Propia

En la Fig. 34 se realizó una operación mutation de crear ingredient, en la que se pide un objeto ingredient de acuerdo con su id, name, calories, en los datos de retorno se tendrá el id, name, calories.

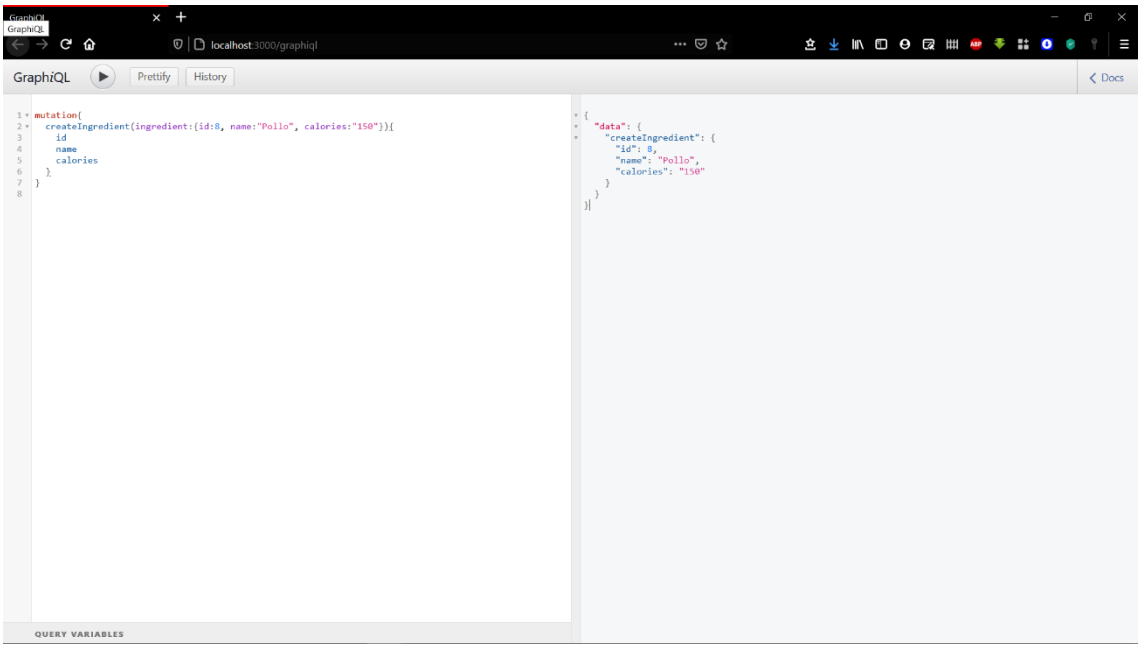


Fig. 34. Mutation Create Ingredient
Fuente: Propia

- **Pizza – REST**

En la Fig. 35 se puede observar la estructura del proyecto pizza con REST, el cual fue realizado en el lenguaje JavaScript con el framework Node JS.

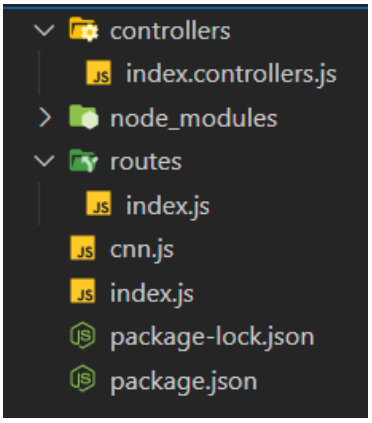


Fig. 35. Estructura del Proyecto REST
Fuente: Propia

El archivo index.js contiene las importaciones necesarias para utilizar las herramientas de express, además del uso de los servicios y métodos necesarios para lanzar el aplicativo, el código se muestra en la Fig. 36.

index.js

```
1
2   //Packages
3   const express = require('express');
4   const app = express()
5   const bodyParser = require('body-parser');
6   var cors = require('cors')
7
8   //controlador
9   const index_controller = require('./controllers/index.controllers')
10
11  //middlewears
12  app.use(cors())
13  app.use(express.json())
14  app.use(express.urlencoded({extended: true}))
15
16  //routes
17  app.use(require('./routes/index'))
18
19
20  //Execution
21      app.get('/', (req, res) => {res.send('Welcome to Pizza API-REST !')})
22  app.listen(3000)
23  console.log('Server is running in: http://localhost:3000')
24
```

Fig. 36. Index.js Pizza – REST
Fuente: Propia

En el archivo que se muestra en la Fig. 37 se muestra la conexión a la Base de Datos (BDD).

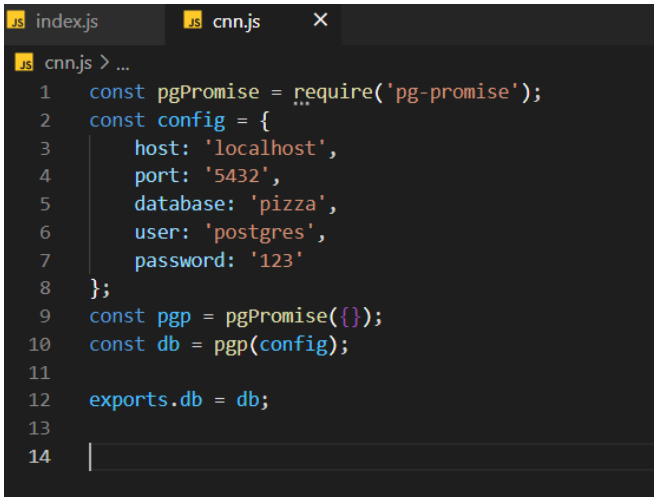


Fig. 37. Conexión Pizza – REST
Fuente: Propia

En la Fig. 38 se muestra cómo se estructuro rutas para las operaciones HTTP de pizza, ingrediets, pizza ingredients.

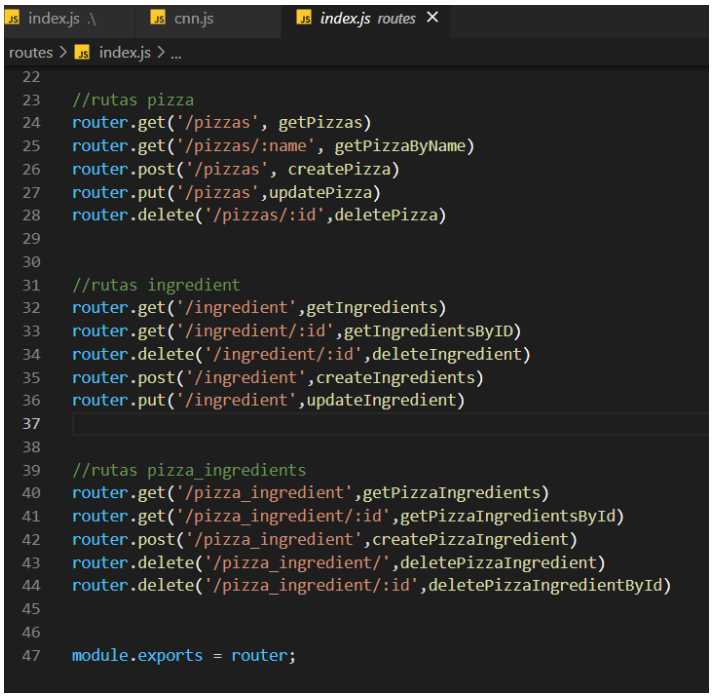


Fig. 38. Rutas Pizza – REST
Fuente: Propia

La Fig. 39 muestra el index controller en el cual se realiza los métodos de peticiones HTTP del CRUD con sus respectivas consultas a la BDD.

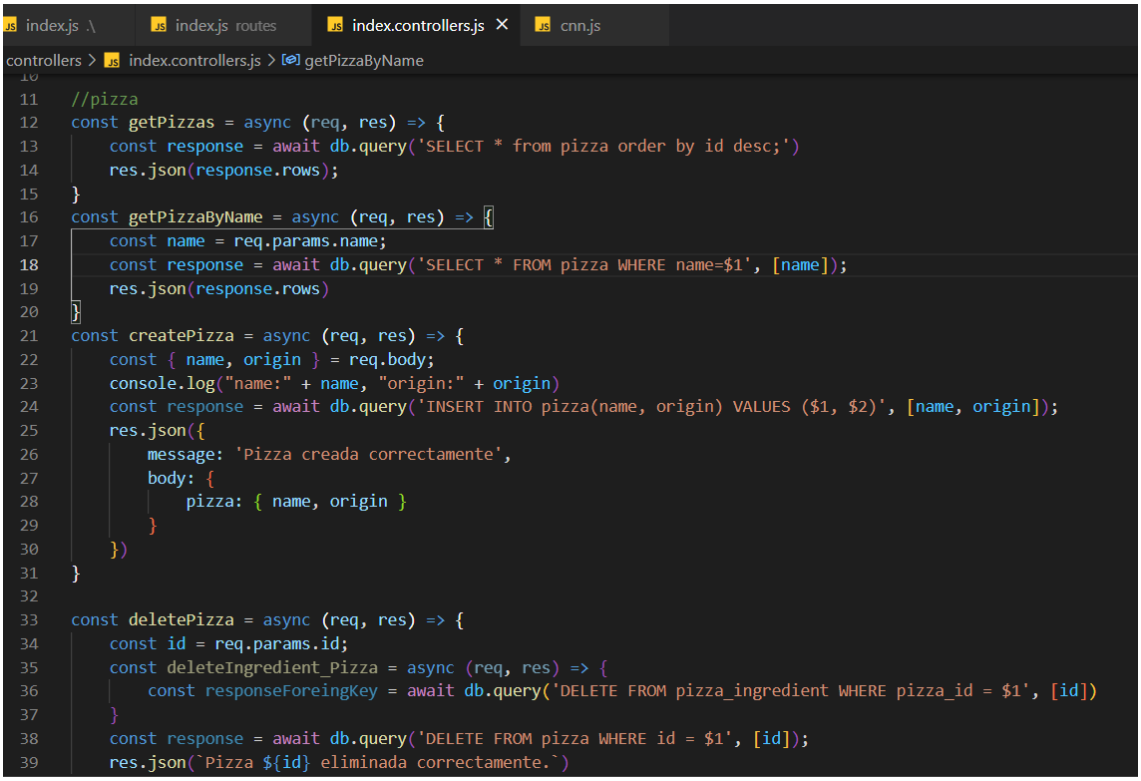


Fig. 39. Controller Pizza – REST
Fuente: Propia

Ejecución de las operaciones HTTP de la API REST, mediante la aplicación cliente Postman. En la Fig. 40 se muestra la petición Get de pizza en la que lista los parametros: id, name y origin que existan.

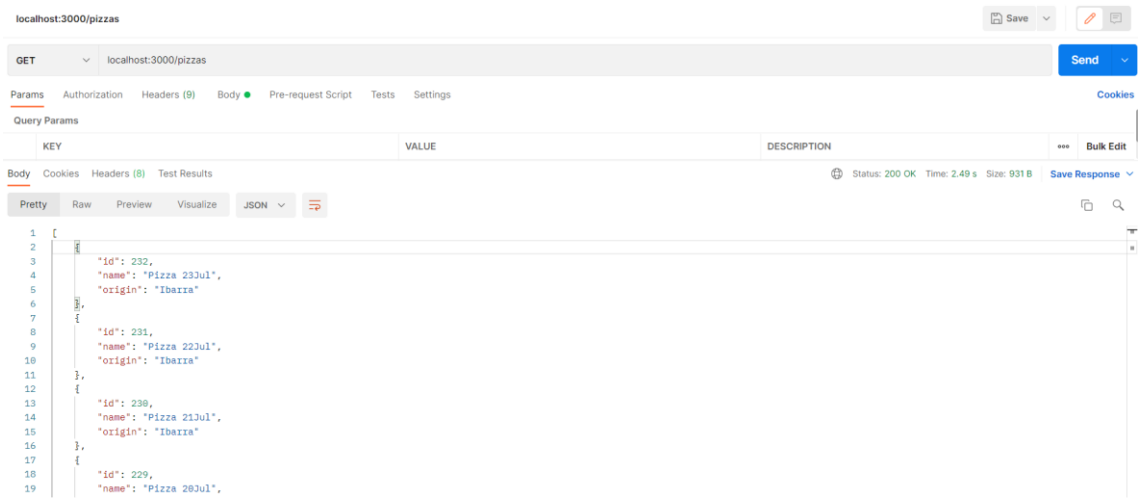


Fig. 40. Resultados Pizza GET
Fuente: Propia

En la Fig. 41 se muestra la petición POST para crear una nueva pizza, la cual requiere los parámetros: name y origin.

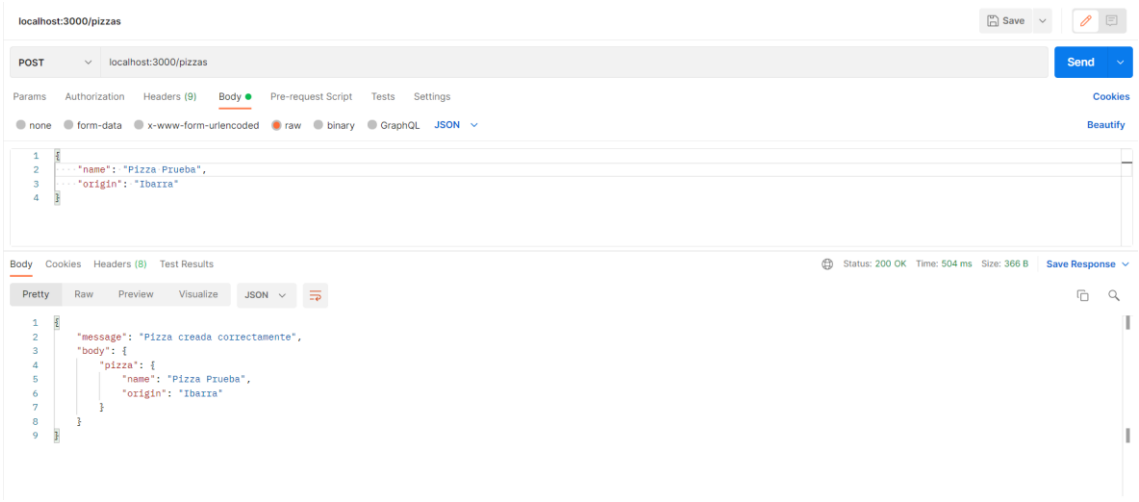


Fig. 41. Resultados Pizza POST
Fuente: Propia

En la Fig. 42 se muestra la petición PUT para actualizar una pizza, la cual requiere los parámetros: id, name, origin.

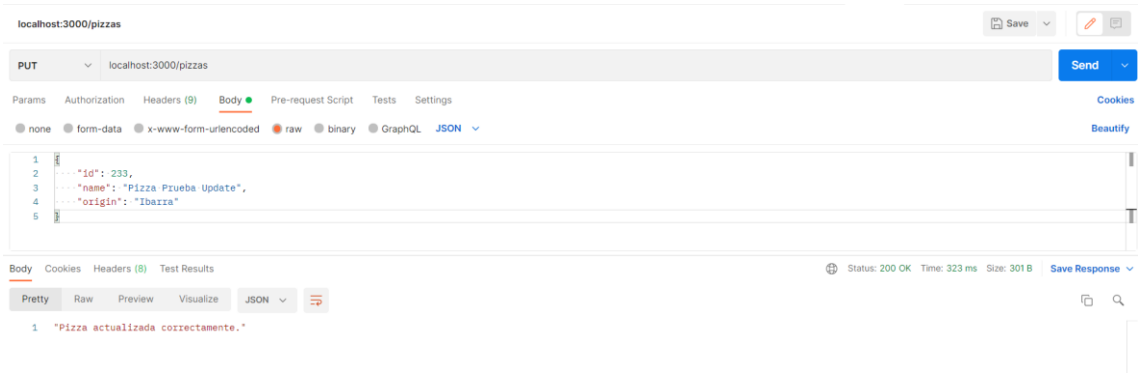


Fig. 42. Resultados Pizza – PUT
Fuente: Propia

En la Fig. 43 se muestra la petición DELETE para eliminar una pizza, la cual requiere el parámetro id.

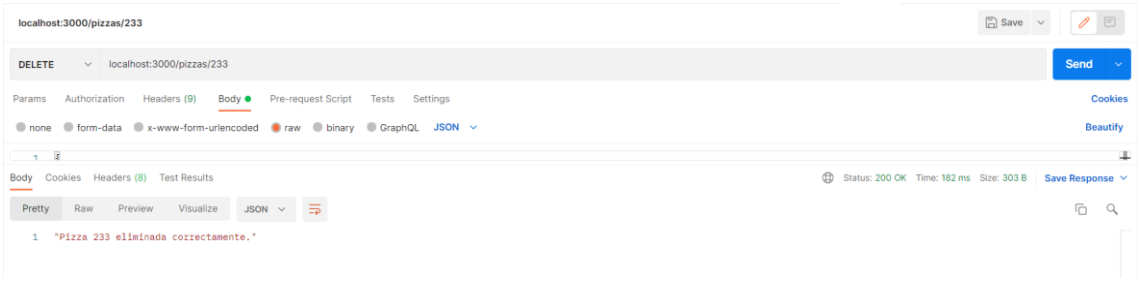


Fig. 43. Resultados Pizzas DELETE
Fuente: Propia

Segunda Fase

En la segunda fase, la fase de Practica, los sujetos del experimento desarrollaron un taller con la temática de libros, el cual consistía en realizar un CRUD de libros, los cuales a su vez tenían sus respectivos autores.

De igual forma que en la primera fase, los sujetos de prueba tuvieron acceso a un backup de la base de datos de libros para facilitar la práctica, además de un diagrama de la base de datos.

El taller tenía requerimientos a completar los cuales constaban en una hoja de Excel, en la que se insertaba capturas de pantalla por cada requerimiento completado y se añadía el tiempo de culminación de la actividad.

En la Fig. 44 se observa la estructura de la segunda fase.

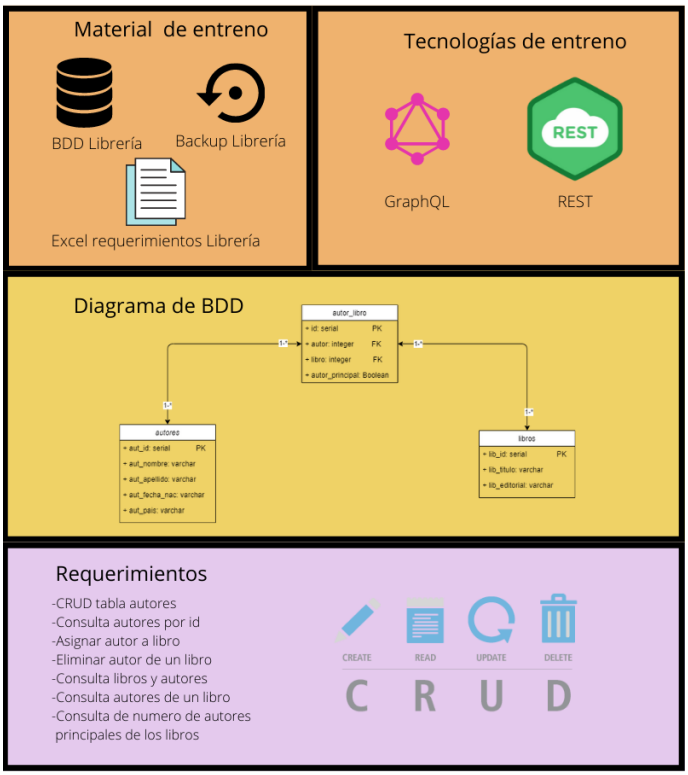


Fig. 44. Estructura Librería
Fuente: Propia

Tercera Fase

Posterior a este taller, se realizó un deber con la temática de un blog, en este cada blog debía tener una categoría, la cual contiene una publicación con autor y comentarios con me gusta.

De igual forma que en la segunda fase, los sujetos de prueba tuvieron acceso a un backup de la base de datos de blogs para facilitar la práctica, además de un diagrama de la base de datos.

El deber constaba con requerimientos a completar los cuales constaban en una hoja de Excel, en la que se insertaba capturas de pantalla por cada requerimiento completado y se añadía el tiempo de culminación de la actividad.

En la Fig. 45 se observa la estructura de la tercera fase.

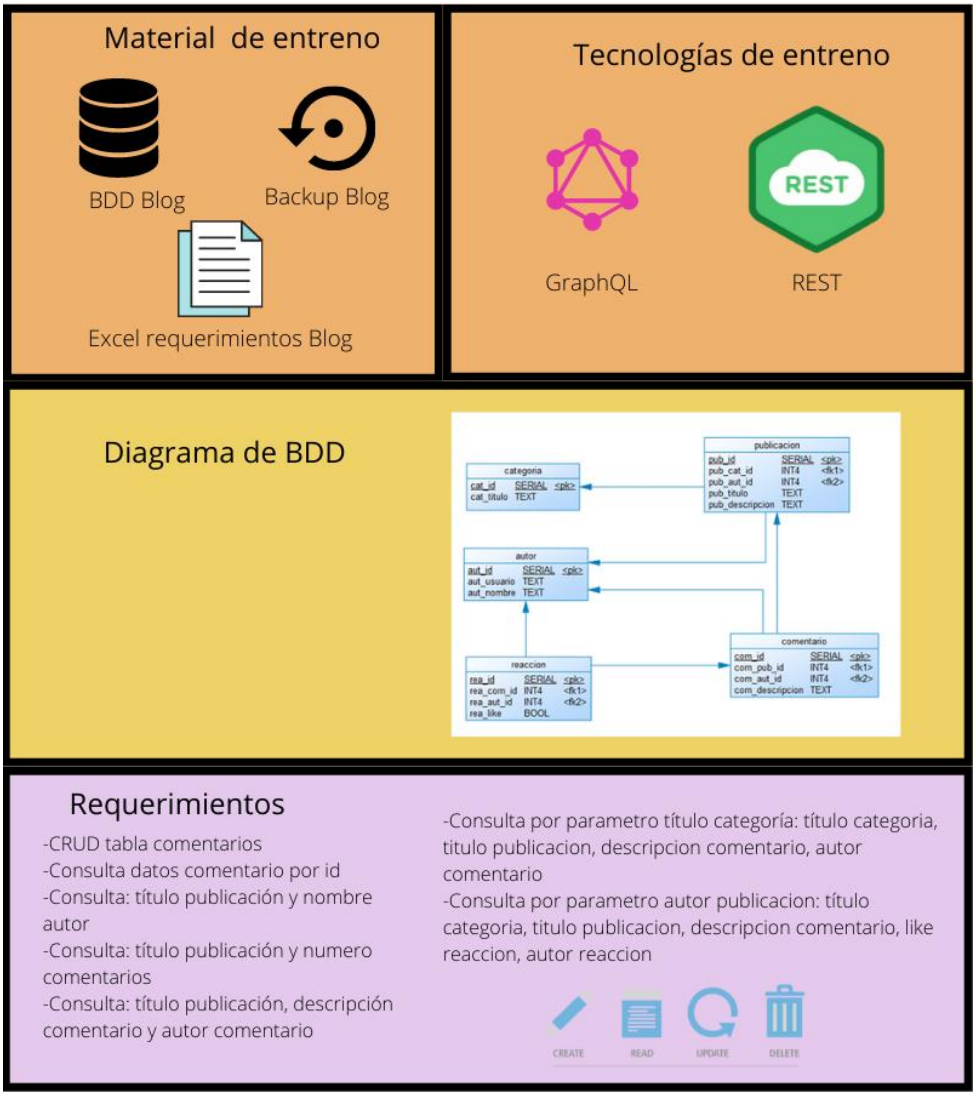


Fig. 45. Estructura Blog
Fuente: Propia

Cuarta Fase

En la cuarta fase, la fase de Evaluación o Experimentación, los sujetos realizaron de manera autónoma dos ejercicios con las dos tecnologías mencionadas anteriormente (REST y GraphQL). Se realizo ejercicios de Notas, Facturas y Eventos, en los que consistía en realizar un CRUD y realizar consultas compuestas. El tiempo que tomo por cada sesión fue de 2 horas en un mismo día.

El código fue evaluado con la métrica de facilidad de aprendizaje. Esta métrica se utiliza para evaluar el grado en el cual un producto o sistema puede ser utilizado por usuarios específicos para alcanzar los objetivos de aprendizaje, en este sentido el enfoque fue el de proporcionar una guía de usuario en la etapa de aprendizaje y practica para poder medir si se logró esta facilidad de aprendizaje.

La intención de los ejercicios era tener una dificultad similar en todas sus fases, pero en cada fase ir subiendo el nivel de dificultad en cuanto al manejo de consultas, en la fase de formación se introdujo el uso de CRUD con consulta simples, en la fase de practica se hizo algo similar en el taller, pero se añadió consultas más complejas, pero para el deber se agregó consultas compuestas entre más tablas a la base de datos, por último, la evaluación tuvo una dificultad similar al deber.

En la Fig. 46 se observa la estructura de la cuarta fase.

Fig. 46. Estructura Experimento
Fuente: Propia

2.1.6. Instrumentos

Los sujetos utilizaron los paradigmas de desarrollo de servicios REST y GraphQL, el trabajo se realizó 100% con javascript. La recopilación de datos se realizó solicitando a los estudiantes enviar los archivos de ejercicios terminados con los requisitos solicitados.

El entorno de desarrollo que se utilizó incluyo:

- Visual studio code: es un editor de código fuente ligero el cual es rico en extensiones para varios lenguajes (code.visualstudio, 2022).
- Postgresql: es una base de datos relacional open source (Postgresql.org, 2022).
- PgAdmin 4: es un administrador de base de datos para PostgreSQL la cual es open source (Pgadmin, 2022).
- Express: es un framework web, y es una librería subyacente para un gran número de otros frameworks web de Node (developer.mozilla.org, 2022).
- Nodejs: es un entorno de ejecución para JavaScript que permite crear herramientas del lado del servidor y aplicaciones(developer.mozilla.org, 2022).
- Javascript: es un lenguaje de programación web ligero (Developer.mozilla.org, 2022).

2.4. Operacionalización del experimento

2.1.7. Muestra

Se realizo con los 23 sujetos de prueba el sábado 5 de junio del 2021, los sujetos fueron estudiantes de la materia de Construcción de Software de la carrera de Ingeniería en Software de la Universidad Técnica del Norte, se realizó el experimento de manera virtual con una duración de 2 horas y un descanso de 15 minutos entre sesión.

2.1.8. Preparación

Los sujetos asistieron al experimento a tiempo. Al principio se dio instrucciones a los sujetos sobre las tareas a realizar y la preparación previa de los materiales para realizar los ejercicios.

Mas de la mitad no tenía conocimiento previo a la preparación sobre el desarrollo de API's o los lenguajes de consulta REST y GraphQL. Los sujetos han utilizado lenguajes procedimentales y orientados a objetos; y para los conocimientos impartidos en las aulas, el 100% ha trabajado con JAVA.

Realizamos el experimento de acuerdo con el cronograma establecido, Los experimentadores que realizaron el experimento atendieron consultas de los sujetos experimentales sobre aclaraciones de la actividad.

Los sujetos experimentales desarrollaron de forma totalmente autónoma los ejercicios en REST y GraphQL. Se realizo una estimación de 5 horas para la resolución de ambos ejercicios. En la TABLA 5: Ejecución Experimento se observa la ejecución del experimento.

TABLA 5: Ejecución Experimento

Hora	Pasos	Minutos
14:05	Entrega pre-requisitos BDD, diagramas (Base de datos)	5
14:10	Instalación BDD	15
14:25	Entrega de requisitos sesión 1	5
14:30	Sesión 1	120
16:30	Subir actividad	10
16:40	Entrega pre-requisitos BDD, diagramas (Base de datos)	5
16:45	Instalación BDD	5
16:50	Entrega de requisitos Sesión 2	10
16:55	Receso	5
17:00	Sesión 2	120
19:00	Subir actividad	10

Al inicio del experimento se procedió a entregar la base de datos a utilizar por los sujetos de prueba, todo esto de acuerdo al requisito que se le asigno en la entrega aleatoria de las tareas, por lo que los sujetos tenían base de datos con información pre cargada para realizar el experimento, al momento de la entrega de requisitos los sujetos de prueba solo procedían a realizar la actividad, se precautelo que todos tuvieran los pre requisitos instalados para mayor eficiencia de tiempo en el experimento.

2.5. Recolección de resultados

Al culminar la Sesión 1 del experimento, se procedió a dar un tiempo estimado de 10 minutos para subir la actividad en la que constaban los resultados tal como se muestra en la TABLA 5 sobre la ejecución del experimento. Se adjuntó el Excel de los requerimientos culminados con capturas de pantalla, y a su vez el código adjunto del proyecto, el mismo procedimiento se realizó para la Sesión 2 del experimento.

CAPITULO III

3. RESULTADOS

En este capítulo se evidenció los resultados del experimento controlado para comparar la facilidad de aprendizaje entre GraphQL y REST mediante un modelo de calidad interna basada en las normas ISO/IEC 25000. A continuación, en la Fig. 47, se muestra la estructura del contenido del capítulo.

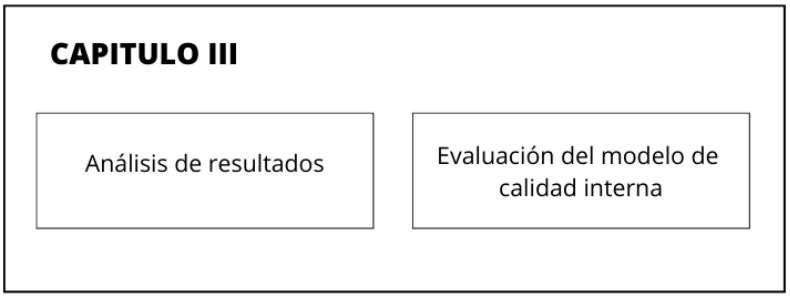


Fig. 47. Estructura Capitulo III
Fuente: Propia

3.1. Análisis de resultados

Los resultados experimentales brindan datos de sumo interés para ejecutar la fase de evaluación del modelo de calidad interna basado en la ISO/IEC 25000, y comparar la facilidad de aprendizaje de los servicios REST y GraphQL, para realizar el análisis se utilizó los resultados obtenidos por los sujetos experimentales de acuerdo con el proceso mencionado en el Capítulo 2.4 y 2.5.

3.1.1. Análisis de resultados REST

En la TABLA 6 se puede observar los resultados obtenidos del experimento, en donde, se dividió los resultados en 3 categorías: passed (completo la tarea), failed (fallo en completar la tarea) y por último not done (no completó la tarea), de esta forma se facilitó la tabulación de datos para aplicar las métricas de evaluación en los resultados del experimento.

TABLA 6: Resultados REST

Nro. Estudiante	P001-REST	P002-REST	P003-REST	P004-REST	P005-REST	P006-REST	P007-REST	P008-REST	P009-REST	P010-REST
1	passed	not done	not done	not done	not done	not done	not done	not done	not done	not done
2	passed	passed	passed	passed	passed	passed	passed	passed	passed	passed
3	passed	passed	passed	passed	passed	passed	passed	not done	not done	not done
4	failed	failed	failed	failed	failed	not done	not done	not done	not done	not done
5	failed	failed	failed	not done	not done	not done	not done	not done	not done	not done
6	passed	passed	passed	passed	passed	not done	not done	passed	not done	not done
7	passed	failed	failed	failed	failed	not done	failed	not done	not done	not done
8	passed	passed	passed	passed	passed	passed	failed	passed	passed	failed
9	passed	passed	passed	passed	passed	passed	passed	passed	passed	passed
10	passed	passed	passed	passed	failed	not done	not done	not done	not done	not done
11	not done	not done	not done	not done	not done	not done	not done	not done	not done	not done
12	failed	failed	failed	failed	failed	not done	not done	not done	not done	not done
13	passed	passed	passed	passed	passed	failed	failed	failed	not done	not done
14	failed	failed	failed	failed	failed	failed	failed	not done	not done	not done
15	passed	passed	passed	passed	failed	passed	not done	not done	not done	not done
16	passed	passed	failed	passed	passed	failed	not done	failed	failed	not done
17	failed	passed	passed	passed	passed	failed	passed	failed	failed	failed
18	passed	passed	passed	passed	passed	passed	passed	passed	failed	failed
19	passed	passed	passed	failed	failed	failed	failed	failed	failed	failed
20	passed	passed	passed	passed	passed	failed	failed	passed	failed	failed
21	passed	passed	passed	passed	passed	passed	passed	passed	not done	not done
22	passed	passed	passed	passed	passed	not done	not done	not done	not done	not done
23	passed	passed	passed	passed	passed	passed	passed	passed	passed	passed

En la Fig. 48 se puede observar los valores porcentuales de respuesta en la pregunta 1 (Insertar datos tabla) del experimento con REST, teniendo como resultados que el 4% de sujetos experimentales no realizó el requerimiento solicitado, un 22% intentó desarrollar el requerimiento, sin embargo, fallo su desarrollo, y un 74% logró realizar con éxitos el requerimiento. En este sentido, se puede observar que el 74% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

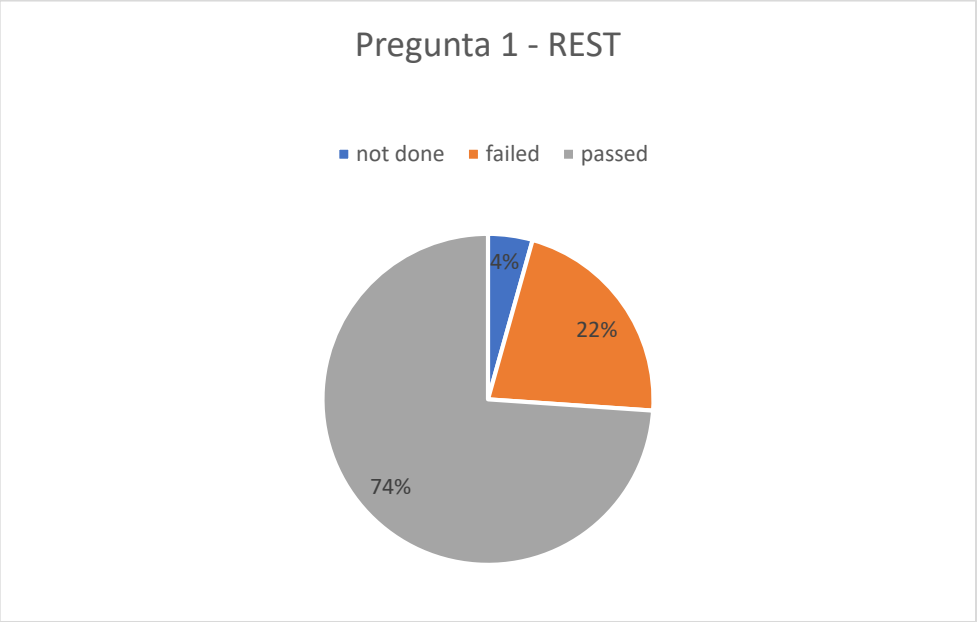


Fig. 48. Pregunta 1 - Experimento REST
Fuente: Propia

En la Fig. 49 se puede observar los valores porcentuales de respuesta en la pregunta 2 (Actualizar datos tabla) del experimento con REST, teniendo como resultados que el 9% de sujetos experimentales no pudo realizar el requerimiento presentado, un 22% fallo, y un 69% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 69% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

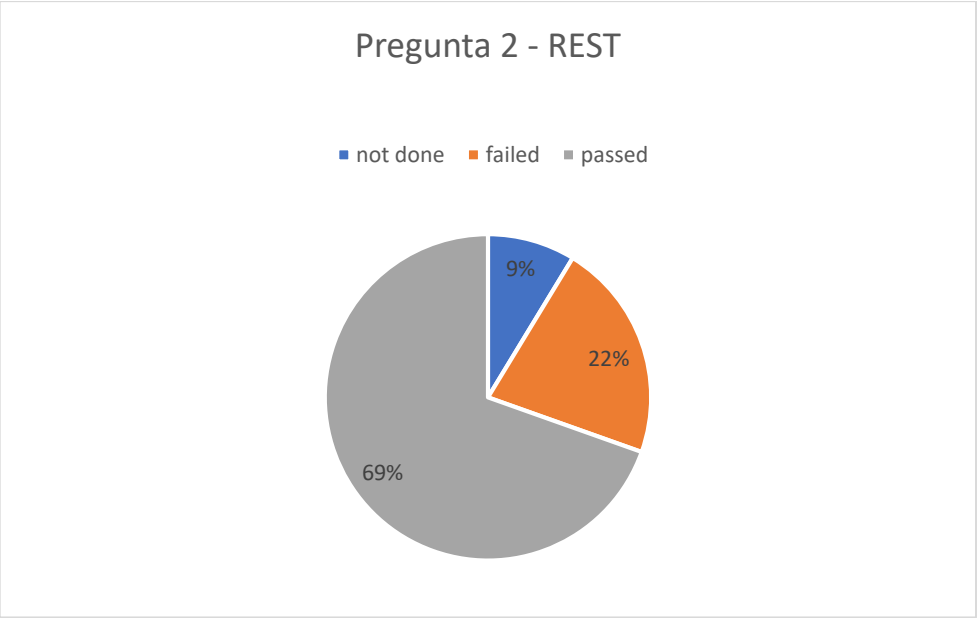


Fig. 49. Pregunta 2 - Experimento REST
Fuente: Propia

En la Fig. 50 se puede observar los valores porcentuales de respuesta en la pregunta 3 (Eliminar datos tabla) del experimento con REST, teniendo como resultados que el 9% de sujetos experimentales no pudo realizar el requerimiento presentado, un 26% fallo, y un 65% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 65% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

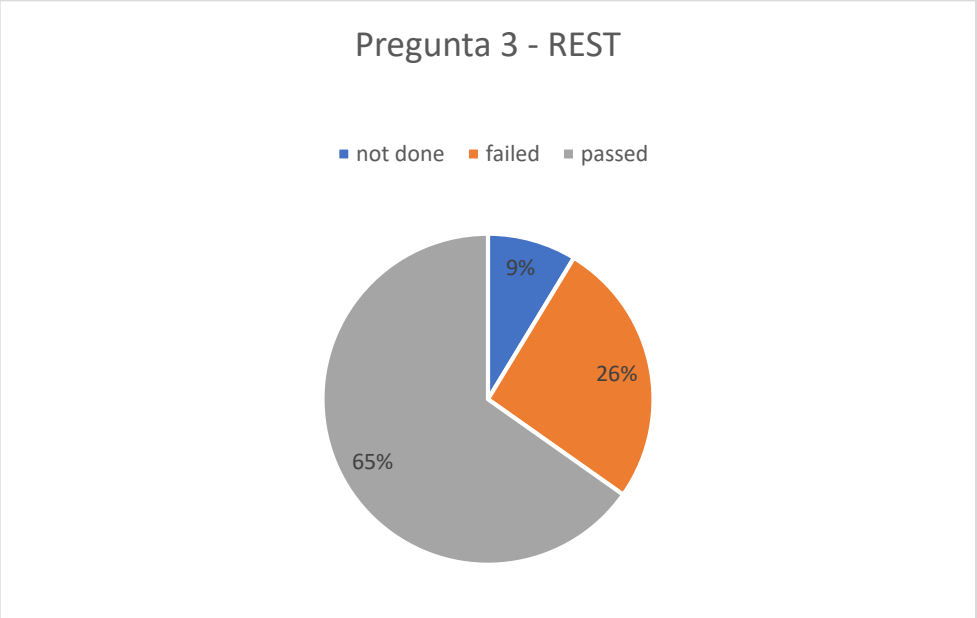


Fig. 50. Pregunta 3 - Experimento REST
Fuente: Propia

En la Fig. 51 se puede observar los valores porcentuales de respuesta en la pregunta 4 (Consultar datos tabla) del experimento con REST, teniendo como resultados que el 13% de sujetos experimentales no pudo realizar el requerimiento presentado, un 22% fallo, y un 65% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 65% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

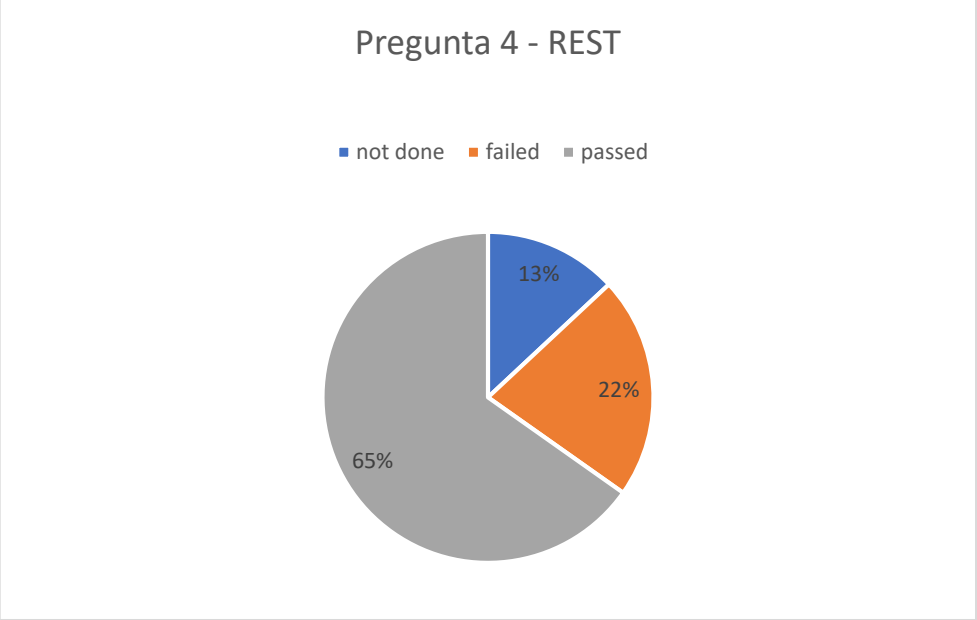


Fig. 51. Pregunta 4 - Experimento REST
Fuente: Propia

En la Fig. 52 se puede observar los valores porcentuales de respuesta en la pregunta 5 (Consultar datos tabla por id) del experimento con REST, teniendo como resultados que el 13% de sujetos experimentales no pudo realizar el requerimiento presentado, un 30% fallo, y un 57% logro realizar con éxitos el requerimiento. Los porcentajes de tareas no realizadas con respecto a las figuras anteriores del experimento desde la Fig. 48 hasta la Fig. 52 varían de forma ascendente en todas sus categorías exceptuando en algunos gráficos que mantienen valores iguales en ciertas categorías, todo esto teniendo en cuenta hasta el momento que los requerimientos del experimento fueron de realización de un CRUD básico a una sola tabla. En este sentido, se puede observar que el 57% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

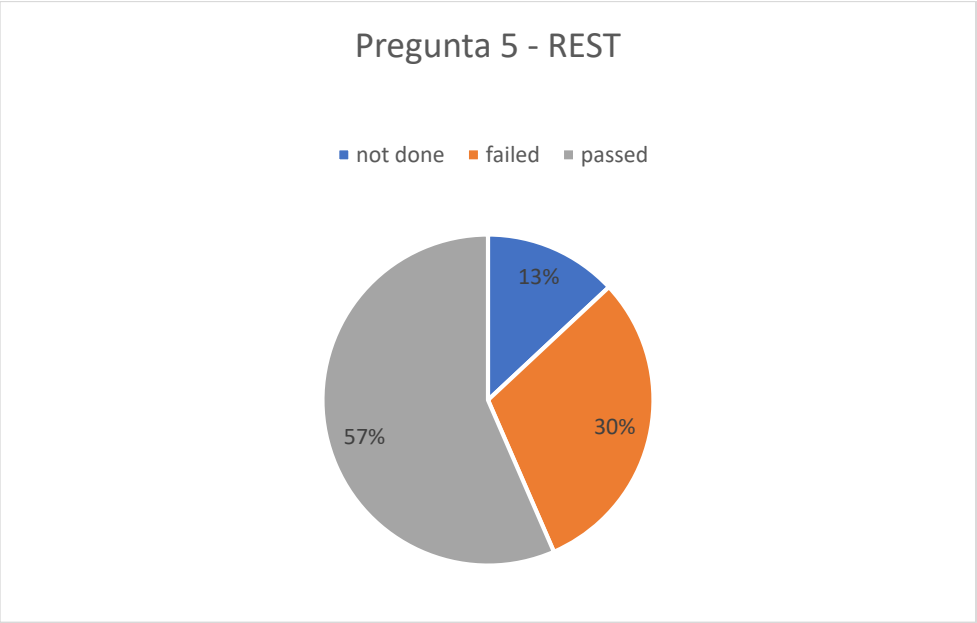


Fig. 52. Pregunta 5 - Experimento REST
Fuente: Propia

En la Fig. 53 se puede observar los valores porcentuales de respuesta en la pregunta 6 (Asignar un dato de tabla 1 a tabla 2) del experimento con REST, teniendo como resultados que el 39% de sujetos experimentales no pudo realizar el requerimiento presentado, un 26% fallo, y un 35% logro realizar con éxitos el requerimiento. Al realizar un requerimiento diferente al de un crud normal se encuentra mayor dificultad en la realización de la tarea y los porcentajes cambian de manera más abrupta, con respecto a la pregunta de la Fig. 52 el porcentaje de no realización de la tarea cambia de un 13% a un 39%, en la categoría de fallo no sucede lo mismo, al contrario se reduce de un 30% a un 26% y el porcentaje de éxito de la tarea pasa de un 57% a un 35%, evidenciando que menos sujetos de prueba pudieron realizar la tarea con una dificultad mayor a un CRUD normal. En este sentido, se puede observar que el 35% de éxito del requerimiento, se encuentra en el rango mínimamente aceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

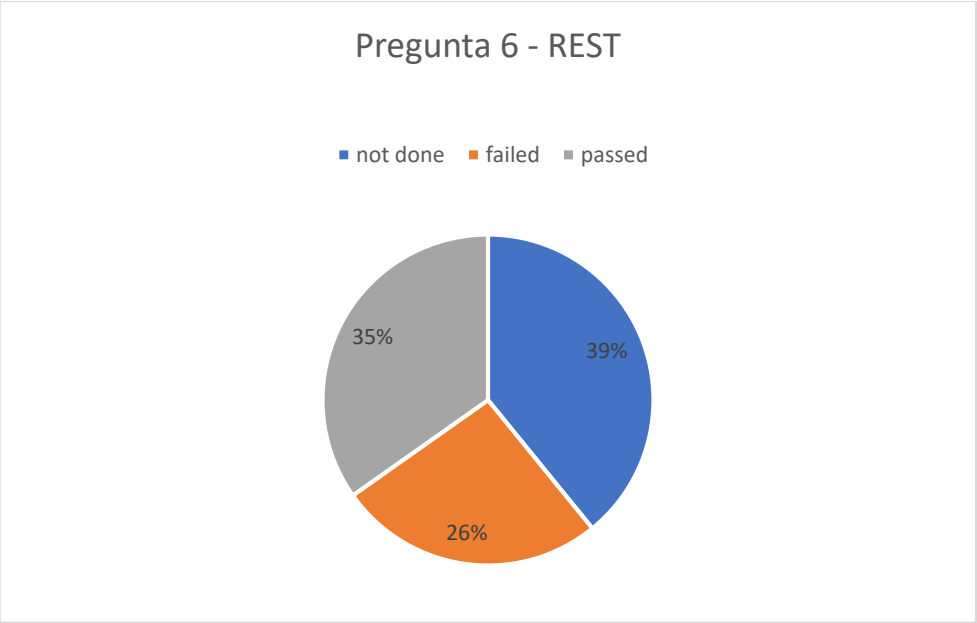


Fig. 53. Pregunta 6 - Experimento REST
Fuente: Propia

En la Fig. 54 se puede observar los valores porcentuales de respuesta en la pregunta 7 (Eliminar dato de tabla 2) del experimento con REST, teniendo como resultados que el 44% de sujetos experimentales no pudo realizar el requerimiento presentado, un 26% fallo, y un 30% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 30% de éxito del requerimiento, se encuentra en el rango mínimamente aceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

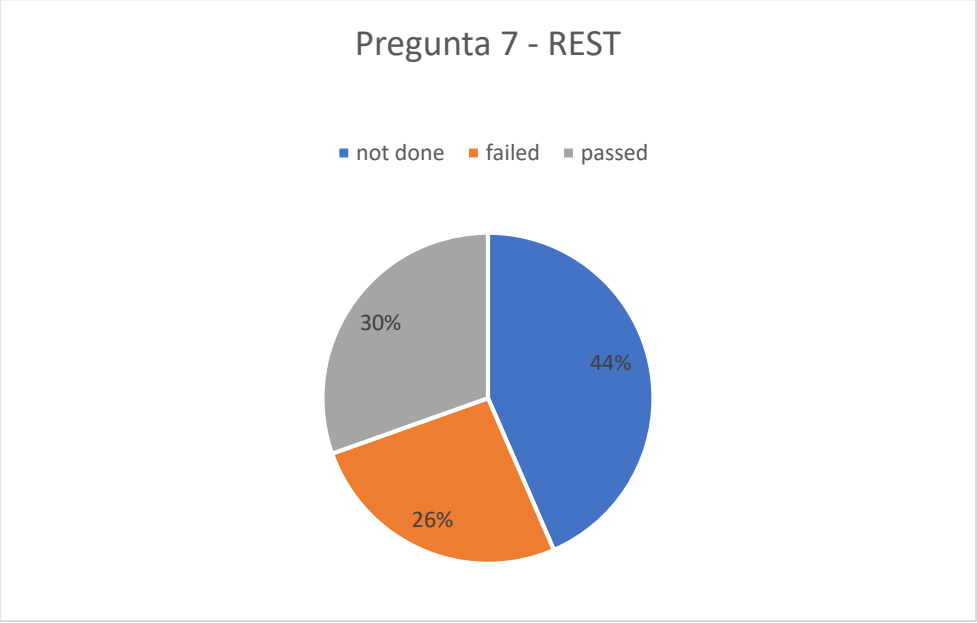


Fig. 54. Pregunta 7 - Experimento REST
Fuente: Propia

En la Fig. 55 se puede observar los valores porcentuales de respuesta en la pregunta 8 (Consulta de 2 tablas, y 4 campos) del experimento con REST, teniendo como resultados que el 48% de sujetos experimentales no pudo realizar el requerimiento presentado, un 17% fallo, y un 35% logro realizar con éxitos el requerimiento. Hasta este punto el porcentaje de sujetos experimentales que no realizan la tarea aumenta ascendentemente, como referencia se tiene la Fig. 53con un porcentaje del 39% a un 48% en la actual figura, y el resto de las variables con un cambio no muy significativo. En este sentido, se puede observar que el 35% de éxito del requerimiento, se encuentra en el rango mínimamente aceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

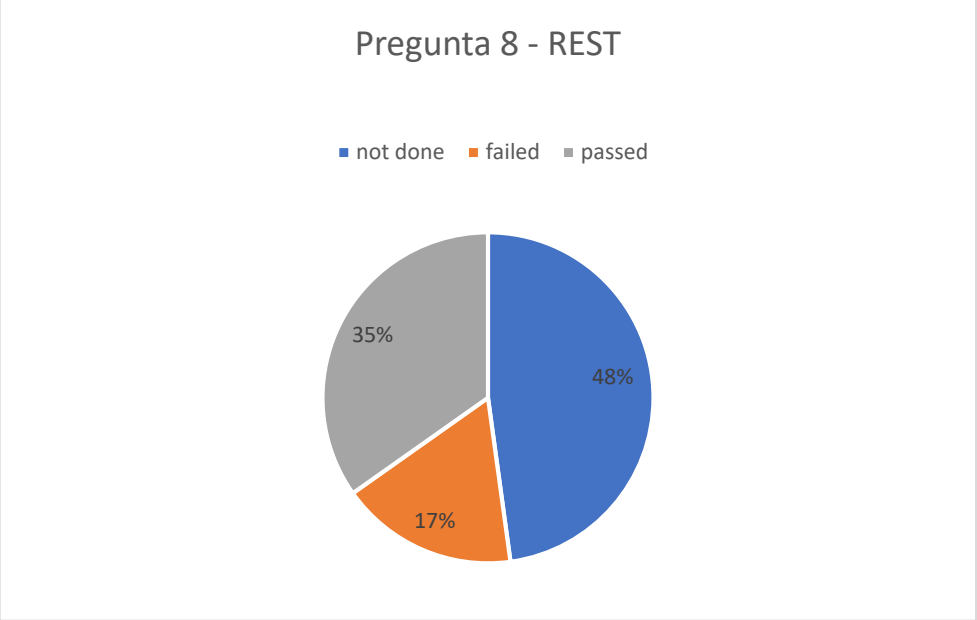


Fig. 55. Pregunta 8 - Experimento REST
Fuente: Propia

En la Fig. 56 se puede observar los valores porcentuales de respuesta en la pregunta 9 (Consulta de 2 tablas, 4 campos, por id) del experimento con REST, teniendo como resultados que el 61% de sujetos experimentales no pudo realizar el requerimiento presentado, un 22% fallo, y un 17% logro realizar con éxitos el requerimiento. El porcentaje de éxito y sujetos que no realizaron el experimento vuelve a cambiar con un rango significativo, el porcentaje de éxito de la Fig. 55 tiene un 35% y el de la actual figura un 17%, el porcentaje de sujetos que no realizaron el ejercicio tiene un 48% y el de la actual figura 61%, el cambio de dificultad en el cual la variación de una consulta de forma compuesta a la base de datos y una con parámetros, causo gran dificultad en los sujetos de prueba. En este sentido, se puede observar que el 17% de éxito del

requerimiento, se encuentra en el rango inaceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

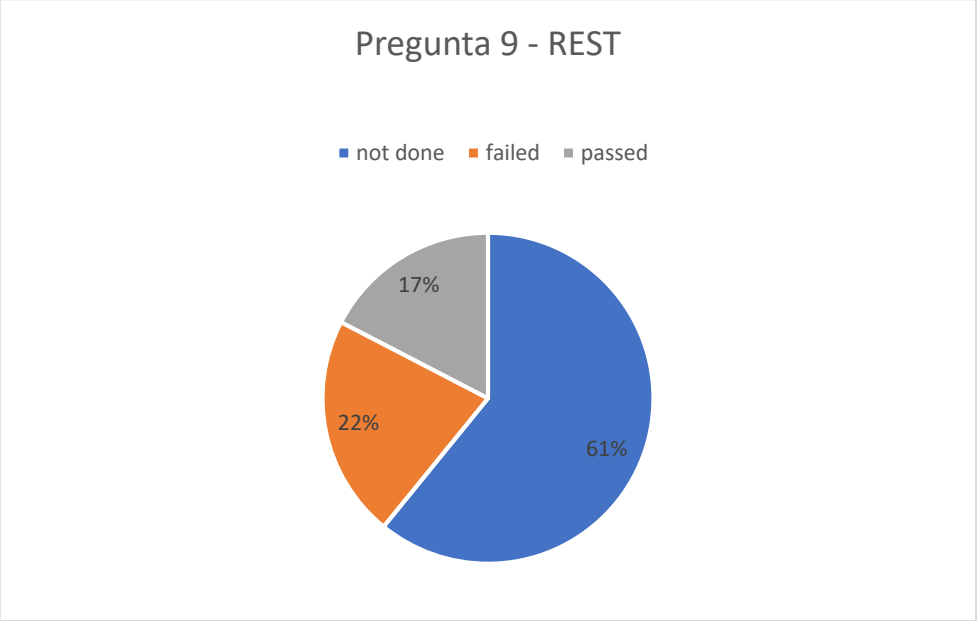


Fig. 56. Pregunta 9 - Experimento REST
Fuente: Propia

En la Fig. 57 se puede observar los valores porcentuales de respuesta en la pregunta 10 (Consulta 3 tablas, 5 campos, por id) del experimento con REST, teniendo como resultados que el 65% de sujetos experimentales no pudo realizar el requerimiento presentado, un 22% fallo, y un 13% logro realizar con éxitos el requerimiento. Al ser una pregunta similar a la Fig. 56 los porcentajes no varían demasiado. En este sentido, se puede observar que el 13% de éxito del requerimiento, se encuentra en el rango inaceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

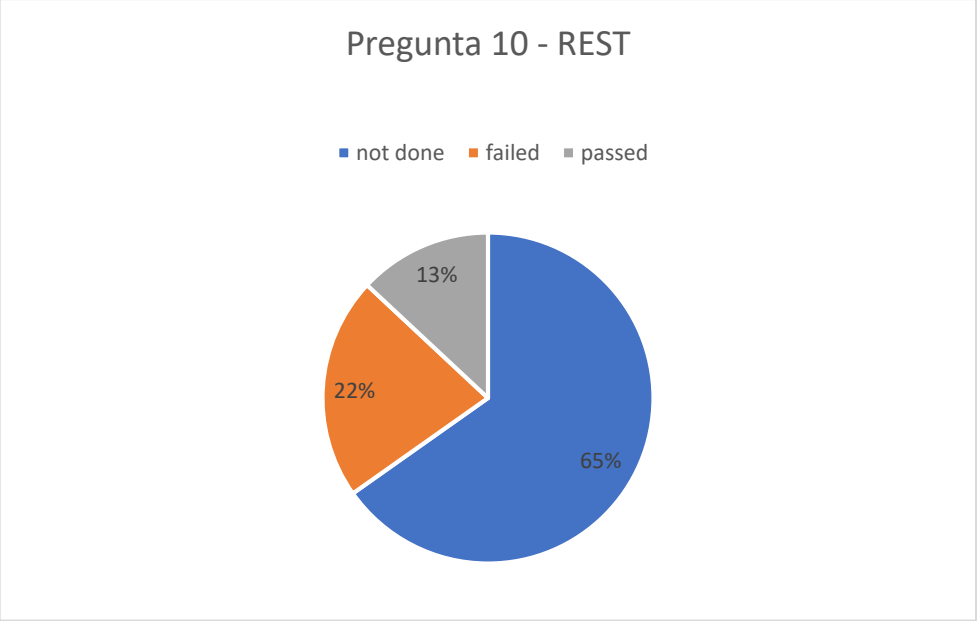


Fig. 57. Pregunta 10 - Experimento REST
Fuente: Propia

3.1.2. Análisis de resultados GraphQL

En la TABLA 7 se puede observar los resultados obtenidos del experimento, en donde, se dividió los resultados en 3 categorías: passed (completo la tarea), failed (fallo en completar la tarea) y por último not done (no completó la tarea), de esta forma se facilitó la tabulación de datos para aplicar las métricas de evaluación en los resultados del experimento.

TABLA 7: Resultados GraphQL

Nro. Estudiante	P001-GraphQL	P002-GraphQL	P003-GraphQL	P004-GraphQL	P005-GraphQL	P006-GraphQL	P007-GraphQL	P008-GraphQL	P009-GraphQL	P010-GraphQL
1	not done	not done	not done	not done	not done	not done	not done	not done	not done	not done
2	passed	passed	passed	not done	not done	not done	not done	not done	not done	not done
3	passed	passed	passed	passed	passed	failed	failed	passed	passed	failed
4	passed	passed	passed	failed	failed	failed	failed	failed	failed	failed
5	failed	failed	failed	failed	failed	failed	failed	failed	failed	not done
6	passed	passed	passed	passed	failed	not done	not done	not done	not done	not done
7	passed	passed	passed	passed	passed	passed	passed	passed	not done	not done
8	passed	passed	passed	passed	passed	passed	passed	passed	passed	passed
9	passed	passed	passed	passed	passed	passed	passed	passed	passed	passed
10	passed	passed	passed	passed	failed	not done	not done	failed	failed	not done
11	passed	passed	passed	passed	passed	passed	passed	passed	failed	failed
12	passed	passed	passed	passed	not done	not done	not done	not done	not done	not done
13	passed	failed	passed	passed	passed	failed	failed	passed	failed	failed
14	failed	failed	failed	passed	passed	failed	failed	passed	failed	failed
15	passed	passed	passed	passed	passed	passed	passed	passed	failed	not done
16	failed	failed	failed	passed	passed	not done	not done	not done	not done	not done
17	passed	passed	passed	passed	passed	passed	passed	not done	not done	not done
18	passed	passed	passed	passed	passed	passed	passed	passed	passed	failed
19	passed	passed	passed	passed	passed	passed	failed	failed	failed	failed
20	failed	failed	failed	passed	passed	not done	not done	failed	not done	not done
21	passed	passed	passed	passed	passed	passed	passed	passed	passed	passed
22	not done	not done	not done	not done	not done	not done	not done	not done	not done	not done
23	passed	passed	passed	passed	passed	not done	not done	passed	passed	passed

En la Fig. 58 se puede observar los valores porcentuales de respuesta en la pregunta 1 (Insertar datos tabla) del experimento con GraphQL, teniendo como resultados que el 9% de sujetos experimentales no pudo realizar el requerimiento presentado, un 17% fallo, y un 74% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 74% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

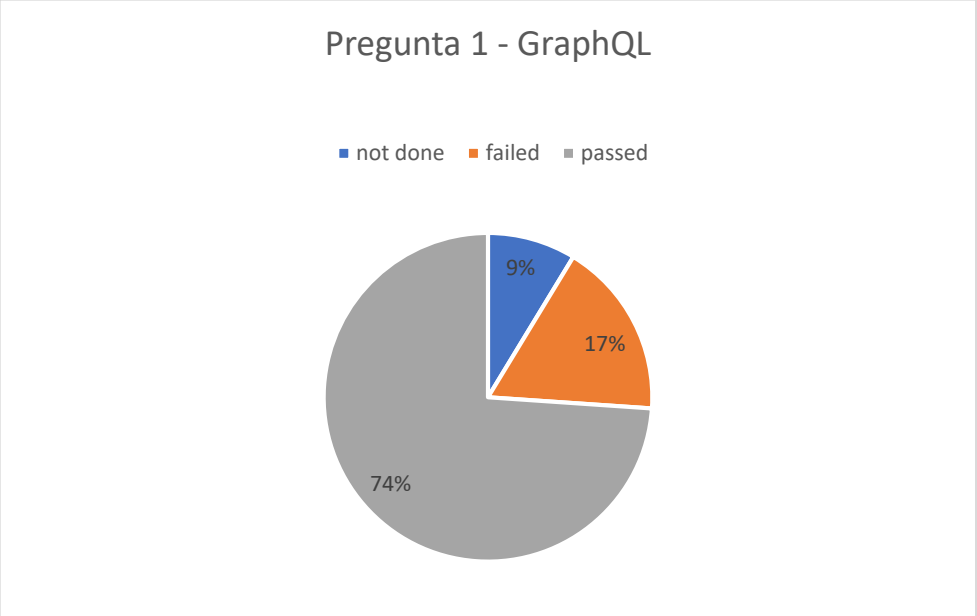


Fig. 58. Pregunta 1 - Experimento GraphQL
Fuente: Propia

En la Fig. 59 se puede observar los valores porcentuales de respuesta en la pregunta 2 (Actualizar datos tabla) del experimento con GraphQL, teniendo como resultados que el 9% de sujetos experimentales no pudo realizar el requerimiento presentado, un 22% fallo, y un 69% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 69% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

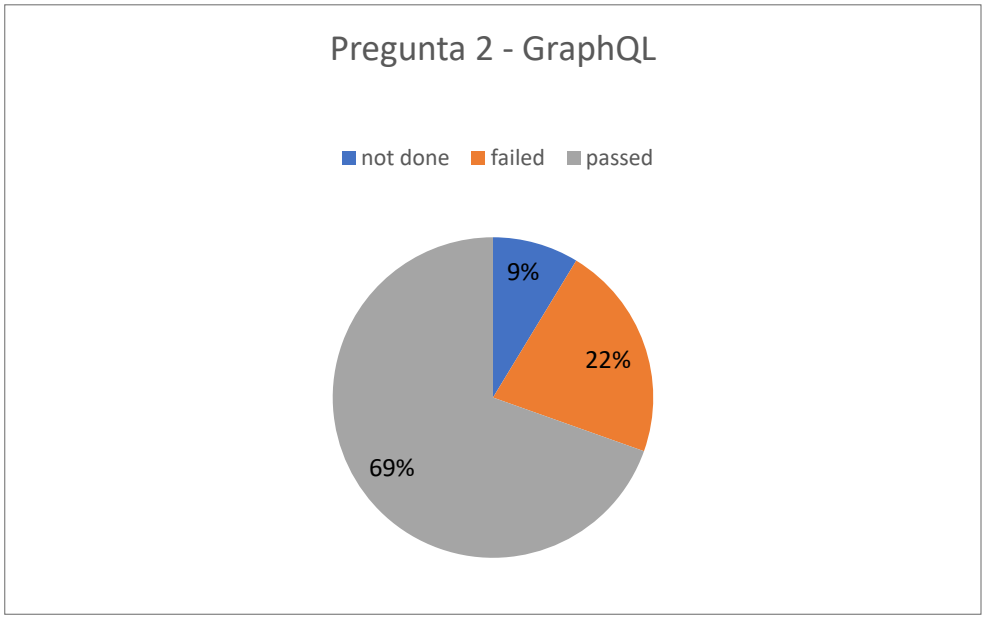


Fig. 59. Pregunta 2 - Experimento GraphQL
Fuente: Propia

En la Fig. 60 se puede observar los valores porcentuales de respuesta en la pregunta 3 (Eliminar datos tabla) del experimento con GraphQL, teniendo como resultados que el 9% de sujetos experimentales no pudo realizar el requerimiento presentado, un 17% fallo, y un 74% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 74% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

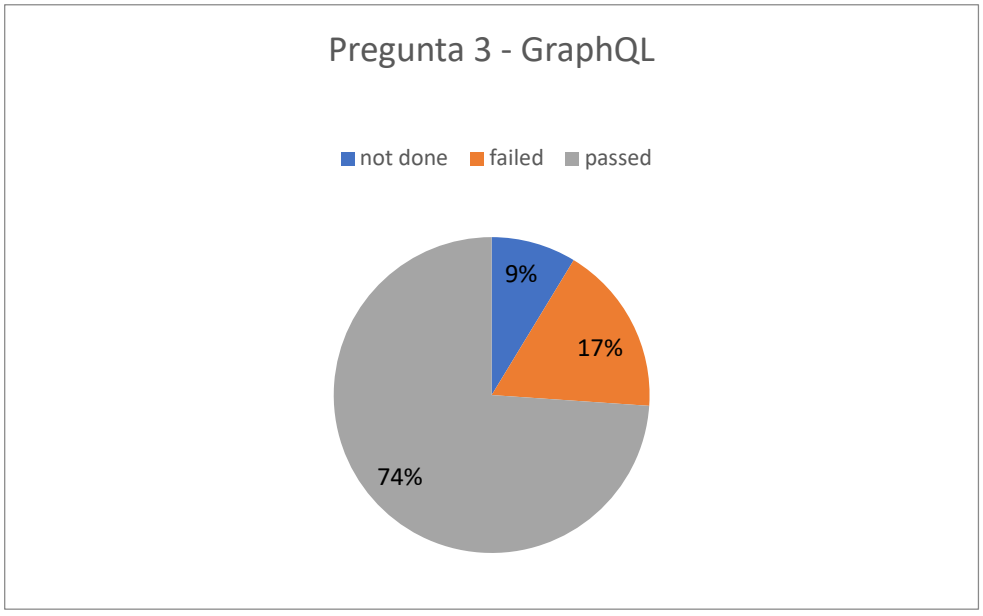


Fig. 60. Pregunta 3 - Experimento GraphQL
Fuente: Propia

En la Fig. 61 se puede observar los valores porcentuales de respuesta en la pregunta 4 (Consultar datos tabla) del experimento con GraphQL, teniendo como resultados que el 13% de sujetos experimentales no pudo realizar el requerimiento presentado, un 9% fallo, y un 78% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 78% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

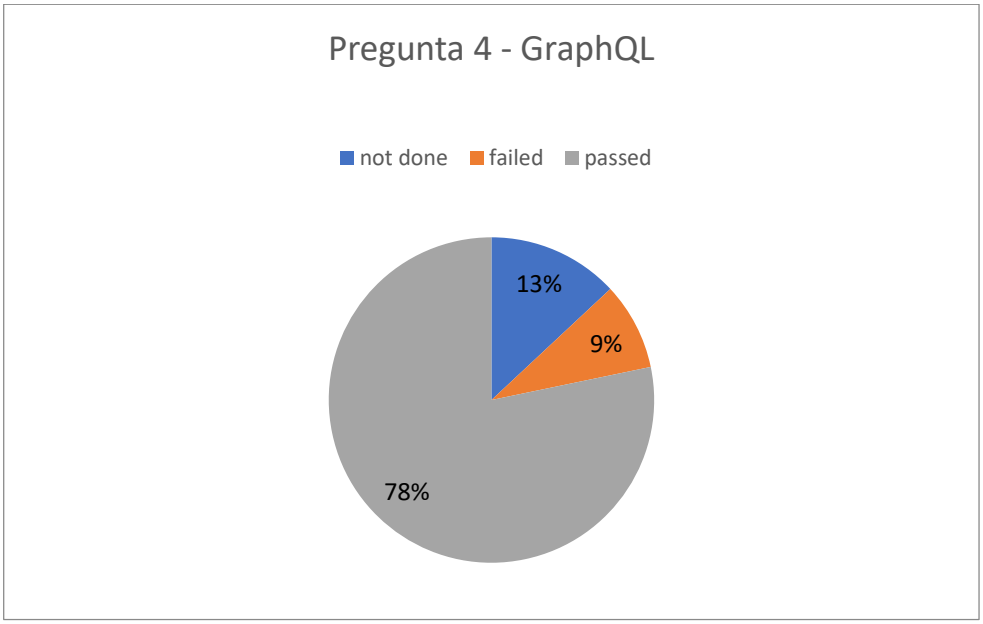


Fig. 61. Pregunta 4 - Experimento GraphQL
Fuente: Propia

En la Fig. 62 se puede observar los valores porcentuales de respuesta en la pregunta 5 (Consultar datos tabla por id) del experimento con GraphQL, teniendo como resultados que el 18% de sujetos experimentales no pudo realizar el requerimiento presentado, un 17% fallo, y un 65% logro realizar con éxitos el requerimiento. Los porcentajes de tareas no realizadas con respecto a las figuras anteriores del experimento como en la Fig. 58 y Fig. 59, no varían teniendo un porcentaje de 9%, y desde la Fig. 60 a la Fig. 62 varían de forma ascendente de un 13% a un 18%, en la categoría de requerimientos fallidos en cambio las figuras Fig. 61 y Fig. 59 tienen valores de 22% y 9% respectivamente, y las figuras Fig. 58, Fig. 60 y Fig. 62 se mantienen con un 17, todo esto teniendo en cuenta hasta el momento que los requerimientos del experimento fueron de realización de un CRUD básico a una sola tabla. En este sentido, se puede observar que el 65% de éxito del requerimiento, se encuentra en el rango objetivo (Satisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

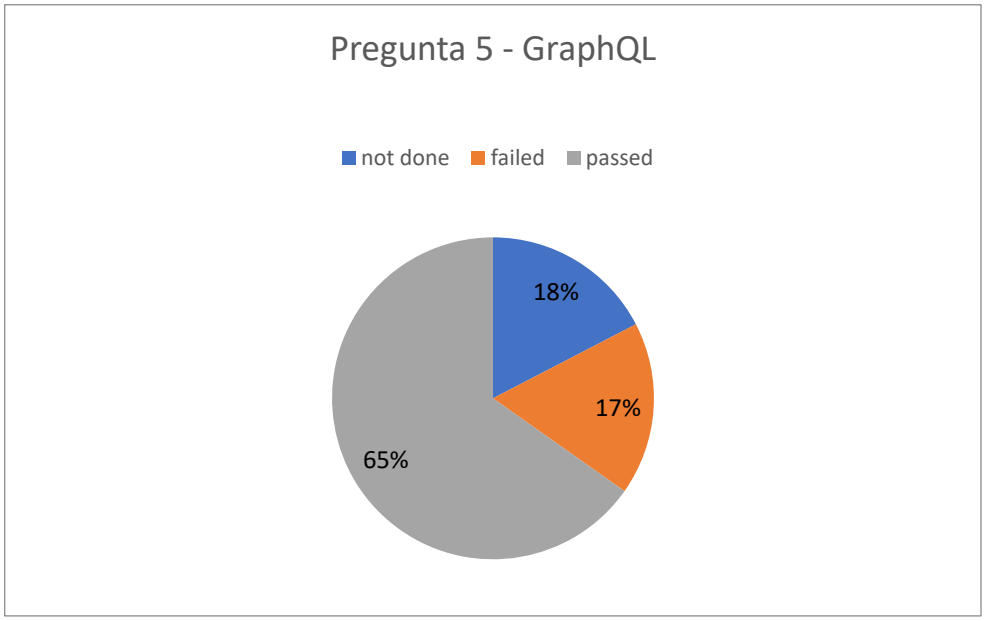


Fig. 62. Pregunta 5 - Experimento GraphQL
Fuente: Propia

En la Fig. 63 se puede observar los valores porcentuales de respuesta en la pregunta 6 (Asignar un dato de tabla 1 a tabla 2) del experimento con GraphQL, teniendo como resultados que el 39% de sujetos experimentales no pudo realizar el requerimiento presentado, un 22% fallo, y un 39% logro realizar con éxitos el requerimiento. Al realizar un requerimiento diferente al de un CRUD normal se encuentra mayor dificultad en la realización de la tarea y los porcentajes cambian de manera más abrupta, con respecto a la pregunta de la Fig. 62, en la que el porcentaje de no realización de la tarea cambia de un 18% a un 39%, en la categoría de fallo no cambia tan drásticamente pues pasa de un 17% a un 22%, y con respecto al porcentaje de éxito del requerimiento se reduce de un 65% a un 39%, evidenciando que menos sujetos de prueba pudieron realizar la tarea con una dificultad mayor a un CRUD normal. En este sentido, se puede observar que el 39% de éxito del requerimiento, se encuentra en el rango mínimamente aceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

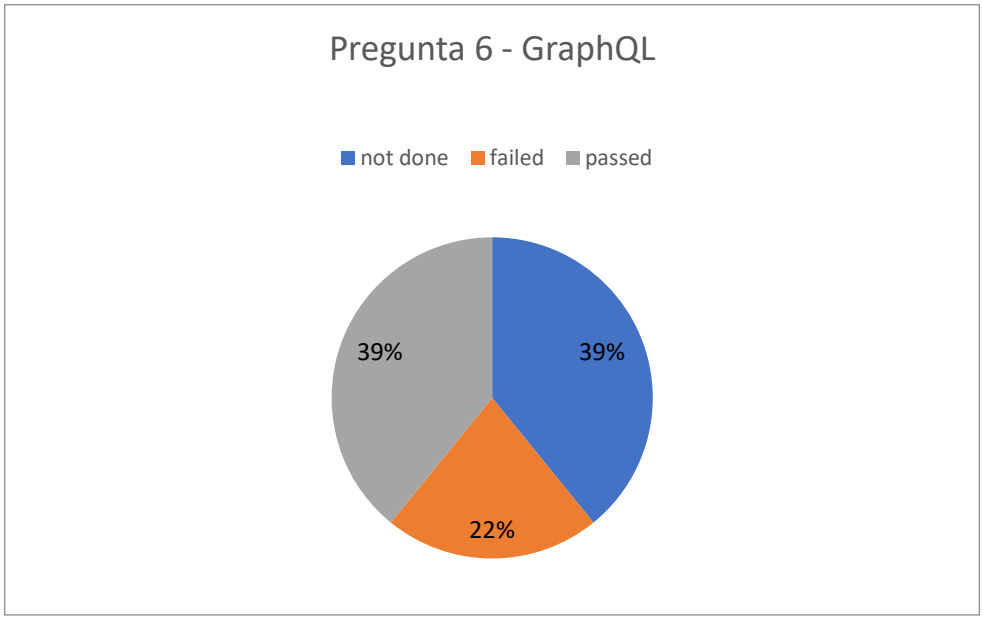


Fig. 63. Pregunta 6 - Experimento GraphQL
Fuente: Propia

En la Fig. 64 se puede observar los valores porcentuales de respuesta en la pregunta 7 (Eliminar dato de tabla 2) del experimento con REST, teniendo como resultados que el 39% de sujetos experimentales no pudo realizar el requerimiento presentado, un 26% fallo, y un 35% logro realizar con éxitos el requerimiento. En este sentido, se puede observar que el 35% de éxito del requerimiento, se encuentra en el rango mínimamente aceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

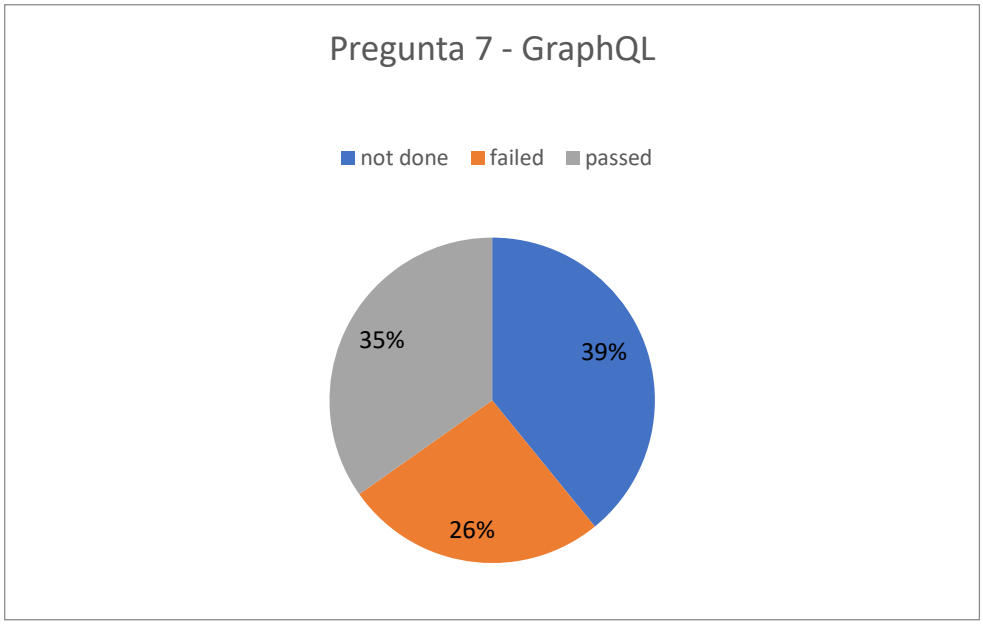


Fig. 64. Pregunta 7 - Experimento GraphQL
Fuente: Propia

En la Fig. 65 se puede observar los valores porcentuales de respuesta en la pregunta 8 (Consulta de 2 tablas, y 4 campos) del experimento con REST, teniendo como resultados que el 30% de sujetos experimentales no pudo realizar el requerimiento presentado, un 22% fallo, y un 48% logro realizar con éxitos el requerimiento. Hasta este punto el porcentaje de sujetos experimentales que no realizan la tarea no varían mucho, teniendo en la figura Fig. 64 un 39% a comparación del 30% de la Fig. 65, y el porcentaje de fallo de las tareas tiene un 26% y 22%, por último, el porcentaje de éxito tiene 35% y sube a un 48% en la figura actual, la variación más significativa es la del porcentaje de éxito. En este sentido, se puede observar que el 48% de éxito del requerimiento, se encuentra en el rango mínimamente aceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

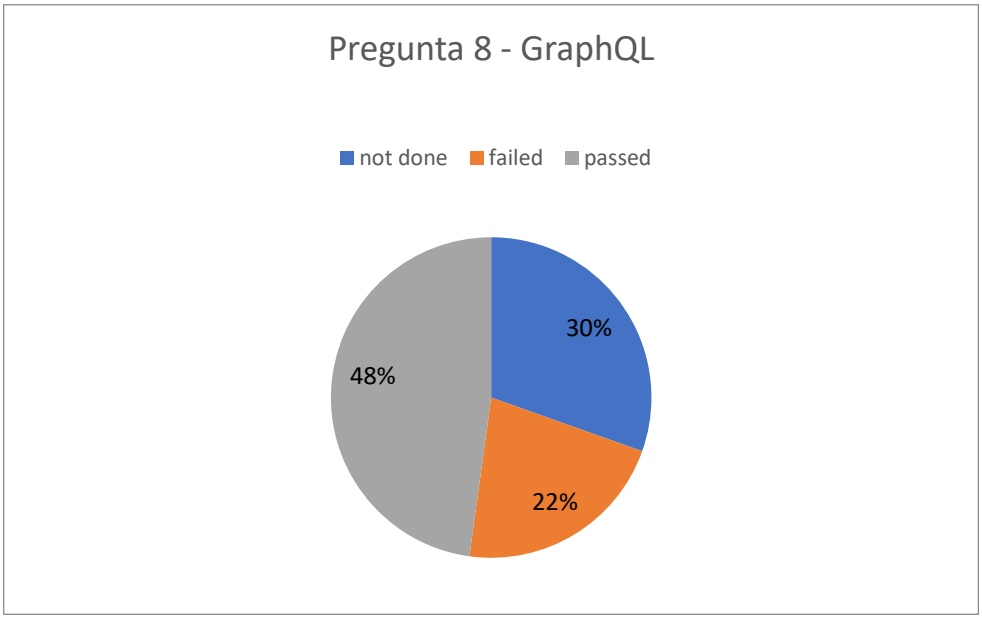


Fig. 65. Pregunta 8 - Experimento GraphQL
Fuente: Propia

En la Fig. 66 se puede observar los valores porcentuales de respuesta en la pregunta 9 (Consulta de 2 tablas, 4 campos, por id) del experimento con GraphQL, teniendo como resultados que el 39% de sujetos experimentales no pudo realizar el requerimiento presentado, un 35% fallo, y un 26% logro realizar con éxitos el requerimiento. El porcentaje de éxito y sujetos que no realizaron el experimento vuelve a cambiar con un rango significativo, el porcentaje de éxito de la Fig. 65 tiene un 48% y el de la actual figura un 39%, el porcentaje de fallo tiene un 22% y en la actual figura un 35%, teniendo más intentos que la anterior, el porcentaje de sujetos que no realizaron el ejercicio tiene un 30% y el de la actual figura 26%, el cambio de dificultad en el cual la variación de una consulta de forma compuesta a la base de datos y una con parámetros, no causo gran dificultad en los sujetos de prueba en el experimento con GraphQL. En este sentido, se puede observar que el 26% de éxito del requerimiento, se encuentra en el rango inaceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

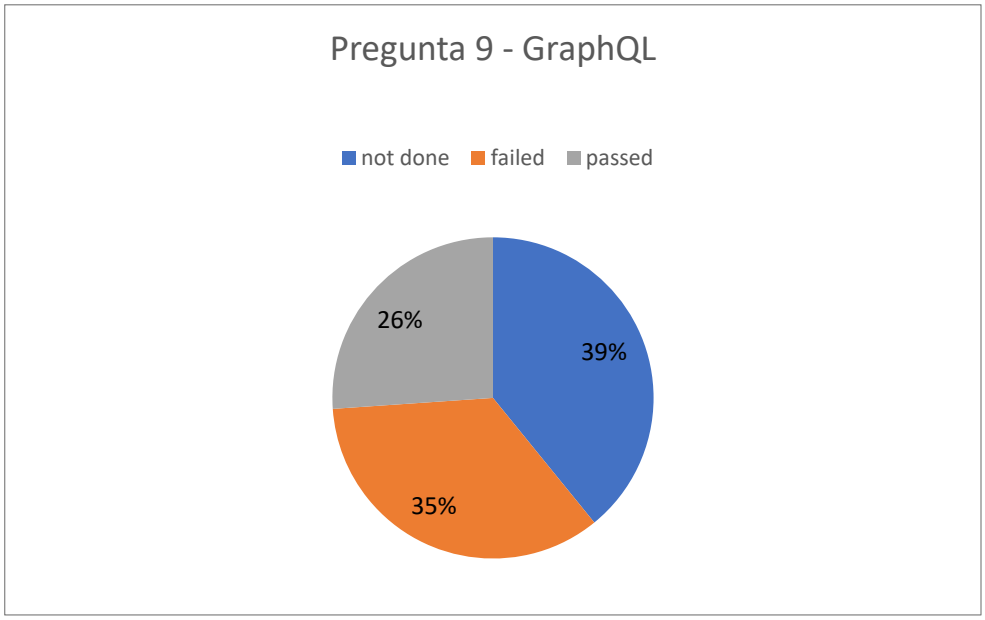


Fig. 66. Pregunta 9 - Experimento GraphQL
Fuente: Propia

En la Fig. 67 se puede observar los valores porcentuales de respuesta en la pregunta 10 (Consulta 3 tablas, 5 campos, por id) del experimento con GraphQL, teniendo como resultados que el 52% de sujetos experimentales no pudo realizar el requerimiento presentado, un 31% fallo, y un 17% logro realizar con éxitos el requerimiento. Al ser una pregunta similar a la figura 31 los porcentajes no deberían variar demasiado, pero el porcentaje de éxito se redujo de un 26% de la Fig. 66 a un 17% en la actual, teniendo un 52% de sujetos de prueba que no realizaron el ejercicio en la actual figura con un 39% de la anterior, y el único porcentaje que no varía significativamente es el de fallo, de un 35% en la anterior a un 31% en la actual figura. En este sentido, se puede observar que el 17% de éxito del requerimiento, se encuentra en el rango mínimamente aceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

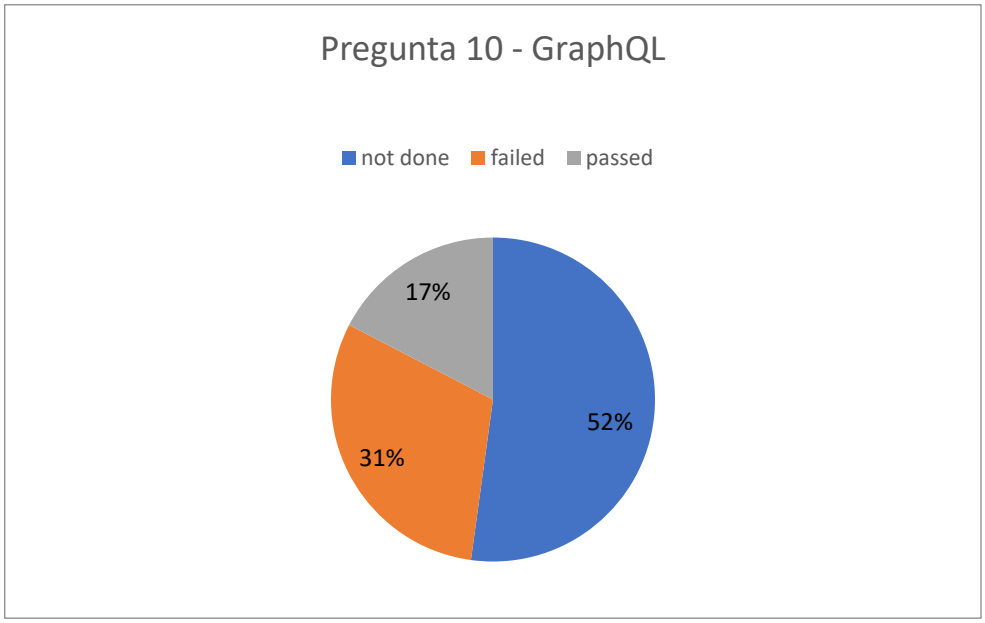


Fig. 67. Pregunta 10 - Experimento GraphQL
Fuente: Propia

En la Fig. 68 se puede apreciar el total de respuestas en las 3 categorías, en la cual se puede apreciar que en la categoría de not done(tareas no realizadas) en GraphQL se tiene un total de 59 y en REST 70, teniendo a REST como lenguaje de consulta en el que más tareas no se realizaron, en la categoría de failed (tareas falladas) se tiene en GraphQL unas 50 y en REST 54, sin tener una gran variación, en la categoría de passed(tareas realizadas con éxito) se tiene en GraphQL 121 y en REST 106, teniendo a GraphQL como el lenguaje de consulta en el que más tareas realizadas con éxito se tuvo. En este sentido, se puede observar que GraphQL tiene un 52.6% de éxito en los requerimientos totales, y se encuentra en el rango objetivo (Satisfactorio) y REST, tiene un 46.08% de éxito, por ende, se encuentra en el rango mínimamente aceptable (Insatisfactorio) de acuerdo con la norma ISO/IEC 14598 (ver Fig. 69).

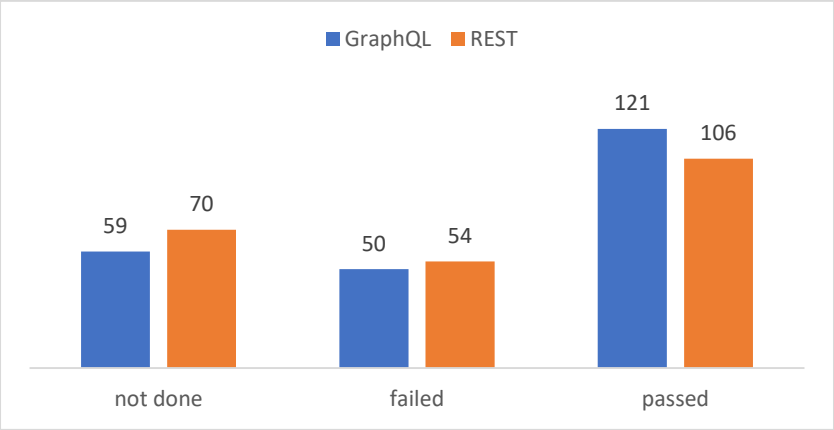


Fig. 68. Comparación total preguntas GraphQL/ REST
Fuente: Propia

3.2. Evaluación del modelo de calidad interna

En esta sección se realizó el procedimiento de evaluación para la calidad interna basada en el modelo calidad de la ISO/IEC 25023, y se documenta los resultados obtenidos con la investigación.

3.2.1. Evaluación característica Usabilidad

Sub - característica facilidad de aprendizaje GraphQL

La sub característica de facilidad de aprendizaje evalúa el grado en el que un producto o sistema puede ser utilizado por los usuarios con eficacia, eficiencia, ausencia de riesgo y satisfacción. Para obtener los resultados se tuvo que contabilizar el número de tareas completadas de los requerimientos para cada sujeto del experimento; luego se realizó una sumatoria y promedio del total de las tareas completadas de los sujetos experimentales como se observa en la TABLA 8.

TABLA 8: Tareas completadas GraphQL

Sujetos Experimentales	Tareas Completadas con éxito
1	0
2	3
3	7
4	3
5	0
6	4
7	8
8	10
9	10
10	4
11	8

12	4
13	5
14	3
15	8
16	2
17	7
18	9
19	6
20	2
21	10
22	0
23	8
Σ	121
Promedio	5.26

La métrica de la “medida de facilidad de aprendizaje” (ver TABLA 2) describe la proporción de funciones que se explica con suficiente detalle en la documentación del usuario o en la función de ayuda para que el usuario pueda aplicar, y en la TABLA 9 se observa la aplicación de tal métrica.

TABLA 9: Métrica completitud de la guía de usuario - GraphQL

Métrica	**Completitud de la guía de usuario**	
Elementos de datos	**Descripción**	**Valor**
A	Número de funciones descritas en la documentación del usuario o en la función de ayuda según sea requerido	5.26
B	Número de funciones implementadas que se requiere que sean documentadas	10

$$x = A/B; = 5.26/10 = 0.53$$

Sub - característica facilidad de aprendizaje REST

La sub característica de facilidad de aprendizaje evalúa el grado en el que un producto o sistema puede ser utilizado por los usuarios con eficacia, eficiencia, ausencia de riesgo y satisfacción. Para obtener los resultados se tuvo que contabilizar el número de tareas completadas de los requerimientos para cada sujeto del experimento; luego se realizó una sumatoria y promedio del total de las tareas completadas de los sujetos experimentales como se observa en la TABLA 10.

TABLA 10: Tareas completadas REST

Sujetos Experimentales	Tareas Completadas con éxito
1	1
2	10
3	7
4	0
5	0
6	6
7	1
8	8
9	10
10	4
11	0
12	0
13	5
14	0
15	5
16	4
17	5
18	8
19	3
20	6
21	8
22	5
23	10
Σ	106
Promedio	4.60

La métrica de la “medida de facilidad de aprendizaje” (ver TABLA 2) describe la proporción de funciones que se explica con suficiente detalle en la documentación del usuario o en la función de ayuda para que el usuario pueda aplicar, y en la TABLA 11 se observa la aplicación de tal métrica.

TABLA 11: Métricas completitud de la guía de usuario - REST

Métrica	**Completitud de la guía de usuario**	
Elementos de datos	**Descripción**	**Valor**
A	Número de funciones descritas en la documentación del usuario o en la función de ayuda según sea requerido	10
B	Número de funciones implementadas que se requiere que sean documentadas	4

$$x = A/B; = 4.60/10 = 0.46$$

3.2.2. Resultados de la evaluación de la característica Usabilidad

En esta sección se muestra los resultados obtenidos de la evaluación de facilidad de aprendizaje entre GraphQL y REST en base al modelo de calidad interna presentado en la TABLA 1, en las presentes TABLA 12 (GraphQL) y TABLA 13 (REST), se muestra característica (C), sub característica (SC), porcentaje de cada una y sus respectivos resultados que indican el grado de facilidad de aprendizaje del lenguaje de consulta.

Resultados GraphQL

Como se observa en la TABLA 12, el porcentaje utilizado de la característica para el modelo de calidad interna de la investigación fue del 100%, y se utilizó la sub característica de la facilidad de aprendizaje con un 100%, los resultados obtenidos fueron del 53% tanto en la característica como en la sub característica.

TABLA 12: Resultado evaluación facilidad de aprendizaje GraphQL

Característica	Sub-característica	Porcentaje Sub-característica (SC)	Porcentaje Característica (C)	Medición	Resultado (SC)	Resultado (C)
Usabilidad	Facilidad de aprendizaje	100%	100%	0.53	53%	53%

- **Resultados REST**

Como se observa en la TABLA 13, el porcentaje utilizado de la característica para el modelo de calidad interna de la investigación fue del 100%, y se utilizó la sub característica de la facilidad de aprendizaje con un 100%, los resultados obtenidos fueron del 46% tanto en la característica como en la sub característica.

TABLA 13: Resultado facilidad aprendizaje - REST

Característica	Sub-característica	Porcentaje Sub-característica (SC)	Porcentaje Característica (C)	Medición	Resultado (SC)	Resultado (C)
Usabilidad	Facilidad de aprendizaje	100%	100%	0.46	46%	46%

3.2.3. Análisis de resultados de la característica de Usabilidad

Para realizar el análisis de resultados se trabajó con la escala de medición de resultados, la cual consta de rangos los cuales van de inaceptable a un excede los requisitos, en la Fig. 69 se puede observar la escala. Las medidas de calidad interna del producto de software fueron tomadas en cuenta conforme nos dice la ISO/IEC 25040, en la que se aplicara un grado apropiado de confianza o métricas para analizar si este cumple con los requisitos de calidad.

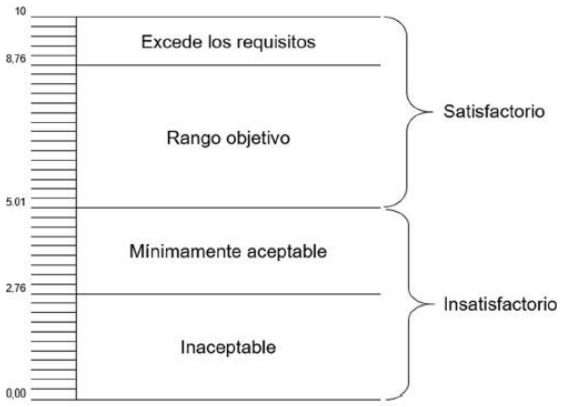


Fig. 69. Escala de medición de resultados
Fuente: Adaptada de (ISO/IEC 14598-1, 1999)

La evaluación del modelo de calidad interna de la facilidad de aprendizaje entre GraphQL y REST, como se puede observar en la TABLA 14 nos muestra que la facilidad de aprendizaje en REST fue mínimamente aceptable (Insatisfactorio) con un 4,6 y GraphQL cumple con el rango objetivo (Satisfactorio) con un 5,3. En este sentido, se puede decir que GraphQL cumple con el modelo de calidad interna basado en la ISO/IEC 25010, en el cual se basó en la métrica de facilidad de aprendizaje, y REST no cumple.

TABLA 14: Resultados GraphQL y REST

Característica	**Sub-característica**	**Resultado GraphQL**	**Resultado REST**
Usabilidad	Facilidad de aprendizaje	5,3	4,6

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

- Con el marco teórico desarrollado en el CAPITULO I, se puede concluir que fue de gran importancia para del resto de la investigación, debido a que se estableció la base conceptual que forja el conocimiento para el diseño y ejecución de un experimento controlado, el cual se utiliza para medir el modelo de calidad interna y lograr realizar la comparación entre GraphQL y REST.
- Mediante el uso de la norma ISO/IEC 25010 permitió definir un modelo de calidad de sistema/producto de una manera sencilla y didáctica (ver capitulo II, TABLA 1), en donde, se seleccionó la característica de usabilidad, con la sub característica de facilidad de aprendizaje, la cual se ajustó a los requerimientos de la investigación. Además, con la utilización de la ISO/IEC 25023 se logró definir la métrica “Facilidad de aprendizaje” (TABLA 2) para medir de manera ordenada y cuantitativa los efectos en la calidad interna del producto de software, cuando se desarrolla con REST o GraphQL.
- El experimento controlado (ver CAPITULO II) facilitó el control directo sobre los tratamientos experimentales (GraphQL y REST), además del diseño de experimentación cruzado de 2 niveles (TABLA 4), permitiendo una mayor flexibilidad y variabilidad en cuanto a la asignación de objetos (ejercicios) a los sujetos de prueba, los cuales fueron asignados de forma aleatoria a las sesiones y tratamientos, logrando una distribución uniforme, cumpliendo con las expectativas de la investigación.
- La aplicación de la norma ISO/IEC 25040, facilitó evaluar los resultados obtenidos y permitió determinar que GraphQL (5,3) tuvo un rango objetivo satisfactorio, y REST (4,6) un rango mínimamente aceptable o insatisfactorio. Lo que quiere decir, que la calidad del producto de software desarrollado con GraphQL es mejor que el producto de software desarrollado con REST. Además, se puede decir que el desarrollo en GraphQL es más fácil que el desarrollo con REST.

Recomendaciones

- Realizar la replicación del experimento controlado en diferentes entornos como, por ejemplo, en otros centros académicos, como también en el entorno industrial. Con lo cual, se podrá aporta nuevos resultados que corroboren o refuten los resultados obtenidos en la presente investigación.
- Se recomienda utilizar la guía de Wohlin para realizar el diseño y ejecución de experimentos controlados.
- No realizar dos sesiones de prueba de la experimentación el mismo día, debido al estrés acumulado en los sujetos de prueba, ya que existe la posibilidad de que los resultados estén sesgados.
- Realizar un análisis estadístico conforme la literatura de diseño de experimentación cruzada lo indica, para controlar datos sesgados y validación de datos entregados.
- Continuar con la investigación y utilizar más características del modelo de calidad interna/externa que presenta la ISO/IEC 25010.

REFERENCIAS Y BIBLIOGRAFIA

Alarcón-Aldana, A. C., Díaz, E. L., & Callejas-Cuervo, M. (2014). Guía para la evaluación de la usabilidad en los entornos virtuales de aprendizaje (EVA). *Informacion Tecnologica*, *25*(3), 135–144. https://doi.org/10.4067/S0718-07642014000300016

Albertos, E. (2018). *Arquitecturas Software para Microservicios: Una Revisión Sistemática de la Literatura*.

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

Bryant, M. (2017). GraphQL for archival metadata: An overview of the EHRI GraphQL API. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, *2018-Janua*, 2225–2230. https://doi.org/10.1109/BigData.2017.8258173

Callejas Cuervo, M., Alarcón Aldan, A., & Álvarez Carreño, A. M. (2017). Software quality models, a state of the art. *Entramado*, *13*(1), 236–250. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1900-38032017000100236

Chauhan, V., Gupta, D. L., & Dixit, S. (2014). Role of Software Metrics to Improve Software Quality. *International Journal of Computer Science and Information Technologies*, *5*(3), 4167–4170. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.589.4656&rep=rep1&type=pdf

code.visualstudio. (2022). *Getting Started*. https://code.visualstudio.com/docs

developer.mozilla.org. (2022). *Introducción a Express/Node*. https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

Developer.mozilla.org. (2022). *JavaScript*. https://developer.mozilla.org/es/docs/Web/JavaScript

Fabijan, A., Dmitriev, P., Olsson, H. H., & Bosch, J. (2017). The benefits of controlled experimentation at scale. *Proceedings - 43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017*, 18–26. https://doi.org/10.1109/SEAA.2017.47

Fabijan, A., Dmitriev, P., Olsson, H. H., & Bosch, J. (2018). Effective online controlled experiment analysis at large scale. *Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018*, 64–67.

https://doi.org/10.1109/SEAA.2018.00020

Guevara-Vega, C., Bernardez, B., Duran, A., Quina-Mera, A., Cruz, M., & Ruiz-Cortes, A. (2021). Empirical Strategies in Software Engineering Research: A Literature Survey. *Proceedings - 2021 2nd International Conference on Information Systems and Software Technologies, ICI2ST 2021*, 120–127. https://doi.org/10.1109/ICI2ST51859.2021.00025

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

Hernandez, M. (2021). *¿ Qué es Babel ? ¿ Qué es Babel ?* https://www.freecodecamp.org/espanol/news/que-es-babel/

ISO/IEC 25010. (2015). *NTE INEN-ISO/IEC 25010 SISTEMAS E INGENIERÍA DE SOFTWARE — REQUISITOS Y EVALUACIÓN DE SISTEMAS Y CALIDAD DE SOFTWARE (SQuaRE) — MODELOS DE CALIDAD DEL SISTEMA Y SOFTWARE (ISO/IEC 25010:2011, IDT)*.

ISO/IEC 25023. (2020). *INTE/ISO/IEC 25023:2020 Ingeniería de sistemas y de software - Requisitos y evaluación de la calidad de sistemas y del software (SQuaRE) – Medición de la calidad de sistemas y productos de software*. *506*.

ISO/IEC 25040. (2010). *Systems and software engineering < Systems and software Quality Requirements and Evaluation (SQuaRE) < Evaluation process*.

Kampenes, V. B., Dybå, T., Hannay, J. E., & Sjøberg, D. I. K. (2007). A systematic review of effect size in software engineering experiments. *Information and Software Technology*, *49*(11–12), 1073–1086. https://doi.org/10.1016/j.infsof.2007.02.015

Ko, A. J., LaToza, T. D., & Burnett, M. M. (2015). A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering*, *20*(1), 110–141. https://doi.org/10.1007/s10664-013-9279-3

Kumar, D. (2018). *Best Practices for Building Restful*. Https://Www.Infosys.Com. https://www.infosys.com/digital/insights/documents/restful-web-services.pdf

Kumar, S. (2019). *RESTful API*. Medium.Com. https://medium.com/@ksarthak4ever/restful-api-1a49417729a8

Mera Paz, J. (2016). Análisis del proceso de pruebas de calidad de software. *Ingeniería Solidaria*, *12*(20), 163–176. https://doi.org/10.16925/in.v12i20.1482

Newman, S. (2015). *Building Microservices @ Squarespace*. O'Reilly Media.

https://books.google.com.ec/books?hl=en&lr=&id=jjl4BgAAQBAJ&oi=fnd&pg=PP1&dq=Newman+Sam.+Building+microservices&ots=_AKRUo6XfM&sig=8gdoi9-fLEYBzqpXNWNwhYoErvQ#v=onepage&q=Newman Sam. Building microservices&f=false

Pgadmin. (2022). *pgAdmin*. https://doi.org/10.1016/s0737-0806(97)80016-0

Postgresql.org. (2022). *PostgreSQL*. https://www.postgresql.org/about

Roa, P., Morales, C., & Gutiérrez, P. (2015). Norma ISO / IEC 25000. *Universidad Distrital Francisco Jose De Caldas*, *3*(2), 26–32. http://revistas.udistrital.edu.co/ojs/index.php/tia

Rodríguez, M., & Piattini, M. (2015). Experiencias en la industria del software: Certificación del producto con ISO/IEC 25000. *CIBSE 2015 - XVIII Ibero-American Conference on Software Engineering*, 814–827.

Santos, D., Resende, A., Junior, P. A., & Costa, H. (2017). Attributes and metrics of internal quality that impact the external quality of object-oriented software: A systematic literature review. *Proceedings of the 2016 42nd Latin American Computing Conference, CLEI 2016*. https://doi.org/10.1109/CLEI.2016.7833322

Sayago H., J., Flores C., E., & Recalde, A. (2019). Análisis Comparativo entre los Estándares Orientados a Servicios Web SOAP, REST y GRAPHQL (Comparative Analysis between Standards Oriented to Web Services SOAP, REST and GRAPHQL). *Revista Antioqueña de Las Ciencias Computacionales y La Ingeniería de Software (RACCIS)*, *9*(March 2020), 10–22. https://doi.org/10.5281/zenodo.3592004

Schafer, D., & Kuenzel, L. (2015). *Suscripciones en GraphQL y Relay*. https://graphql.org/blog/subscriptions-in-graphql-and-relay/

Snell, J., Tidwell, D., & Kulchenko, P. (2001). *Programming Web services with SOAP: Building Distributed Applications*. https://books.google.de/books?id=34JQAAAAMAAJ

Taskula, T. (2019). *Advanced Data Fetching with GraphQL : Case Bakery Service* [Aalto University]. http://urn.fi/URN:NBN:fi:aalto-201903172287%0A

The GraphQL Foundation. (2021). *Schemas and Types | GraphQL*. Https://Graphql.Org. https://graphql.org/learn/schema/

Vaca, T., & Jácome, A. (2018). Calidad de software del módulo de talento humano del sistema informático de la Universidad Técnica del Norte bajo la norma ISO/IEC

25000. *2018*, *May*, ResearchGate. https://www.researchgate.net/publication/325022337_Calidad_de_software_del_modulo_de_talento_humano_del_sistema_informatico_de_la_Universidad_Tecnica_del_Norte_bajo_la_norma_ISOIEC_25000

Vásquez, A. (2020). *EVALUACIÓN DE LA EFICIENCIA DE LAS TECNOLOGÍAS GRAPHQL Y REST EN LA IMPLEMENTACIÓN DE SERVICIOS WEB CONSUMIDOS POR APLICACIONES ANDROID* [Universidad Señor de Sipán]. https://hdl.handle.net/20.500.12802/6729

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 F132203*. https://doi.org/10.1145/3144826.3145437

Vegas, S., Apa, C., & Juristo, N. (2016). Crossover Designs in Software Engineering Experiments: Benefits and Perils. *IEEE Transactions on Software Engineering*, *42*(2), 120–135. https://doi.org/10.1109/TSE.2015.2467378

VILLANES, E. J. B. (2015). Método Para La Evaluación De Calidad De Software. *Universidad San Martin De Porres*.

Vogel, M., Weber, S., & Zirpins, C. (2018). Experiences on Migrating RESTful Web Services to GraphQL. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10797 LNCS*, 283–295. https://doi.org/10.1007/978-3-319-91764-1_23

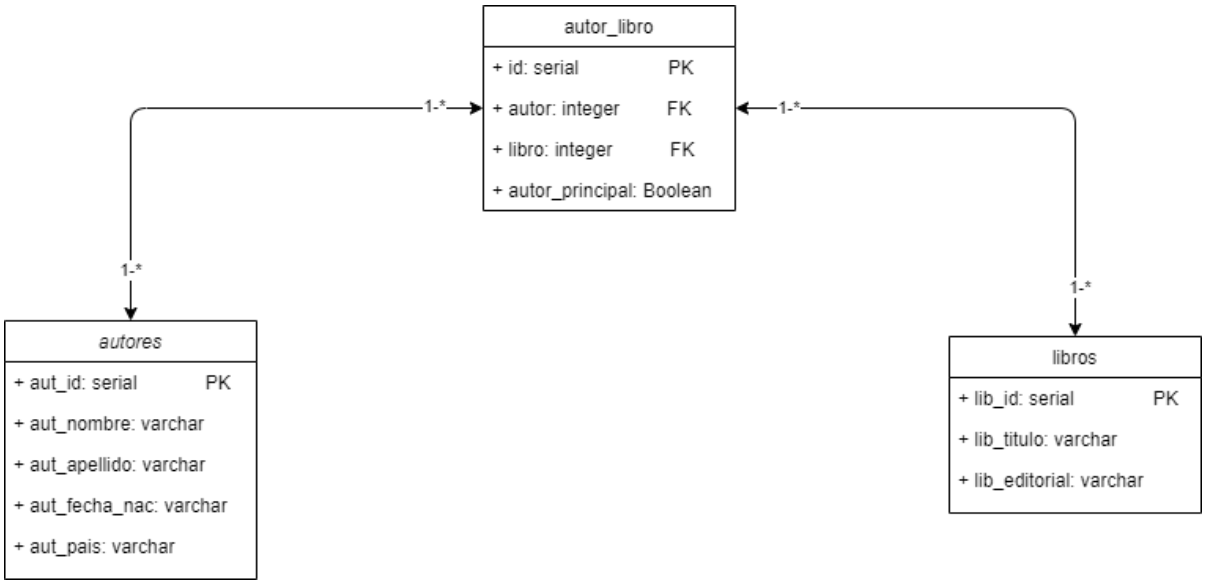
Wohlin, C., Höst, M., & Henningsson, K. (2003). Empirical research methods in software engineering. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2765*, 7–23. https://doi.org/10.1007/978-3-540-45143-3_2

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

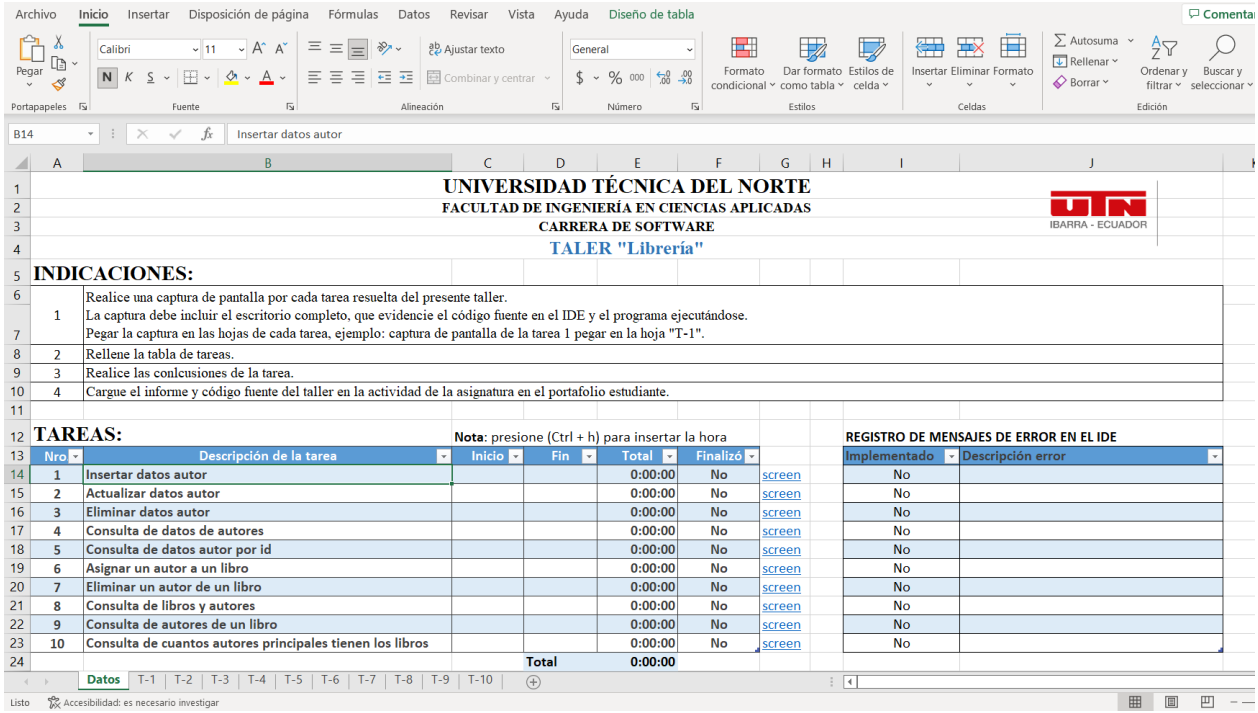
Xinyang, F., Jianjing, S., & Ying, F. (2009). REST: An alternative to RPC for web services architecture. *2009 1st International Conference on Future Information Networks, ICFIN 2009*, 7–10. https://doi.org/10.1109/ICFIN.2009.5339611

ANEXOS

Anexo A: Diagrama de base de datos de taller Librería de la segunda fase del experimento controlado.



Anexo B: Excel con requerimientos del taller Librería de la segunda fase del experimento controlado.



UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE SOFTWARE
IBARRA - ECUADOR
TALER "Librería"

INDICACIONES:

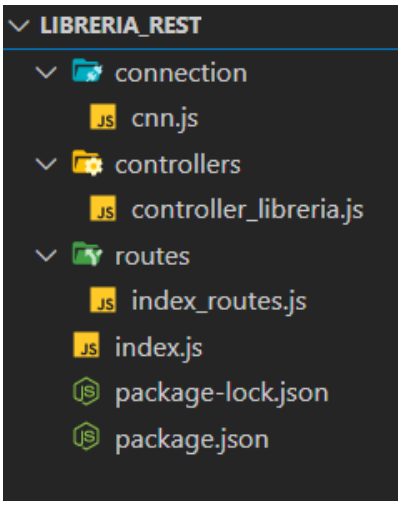
1	Realice una captura de pantalla por cada tarea resuelta del presente taller. La captura debe incluir el escritorio completo, que evidencie el código fuente en el IDE y el programa ejecutándose. Pegar la captura en las hojas de cada tarea, ejemplo: captura de pantalla de la tarea 1 pegar en la hoja "T-1".
2	Rellene la tabla de tareas.
3	Realice las conlcusiones de la tarea.
4	Cargue el informe y código fuente del taller en la actividad de la asignatura en el portafolio estudiante.

TAREAS:

Nota: presione (Ctrl + h) para insertar la hora

Nro	Descripción de la tarea	Inicio	Fin	Total	Finalizó		REGISTRO DE MENSAJES DE ERROR EN EL IDE Implementado	Descripción error
1	Insertar datos autor			0:00:00	No	screen	No	
2	Actualizar datos autor			0:00:00	No	screen	No	
3	Eliminar datos autor			0:00:00	No	screen	No	
4	Consulta de datos de autores			0:00:00	No	screen	No	
5	Consulta de datos autor por id			0:00:00	No	screen	No	
6	Asignar un autor a un libro			0:00:00	No	screen	No	
7	Eliminar un autor de un libro			0:00:00	No	screen	No	
8	Consulta de libros y autores			0:00:00	No	screen	No	
9	Consulta de autores de un libro			0:00:00	No	screen	No	
10	Consulta de cuantos autores principales tienen los libros			0:00:00	No	screen	No	
			Total	0:00:00				

Anexo C: Arquitectura taller Librería con REST de la segunda fase del experimento controlado.



Anexo D: Index taller Librería.

index.js > ...

```
1   //Packages
2   const express = require('express');
3   const app = express()
4   const bodyParser = require('body-parser');
5   var cors = require('cors')
6
7   //controlador
8   // const index_controller = require('./controllers/index.controllers')
9
10  //middlewears
11  app.use(cors())
12  app.use(express.json())
13  app.use(express.urlencoded({extended: true}))
14
15  //routes
16  app.use(require('./routes/index_routes'))
17
18
19  //Execution
20  app.get('/', (req, res) => {res.send('Welcome to Libreria API-REST !')})
21  app.listen(3000)
22  console.log('Server is running in: http://localhost:3000')
23
```

Anexo E: Connection del taller Librería con REST.

connection > cnn.js > ...

```
const {Pool} = require('pg')

const bdd = new Pool({
    user: 'postgres',
    host: 'localhost',
    database: 'libros',
    password: 'qwerty123',
    port: 5432
})

module.exports = {
    bdd
}
```

Anexo F: Controller del taller Librería con REST.

controllers > controller_libreria.js > ...

```
const {bdd} = require('../connection/cnn')

// libro
const InsertarLibro = async(req, res) =>
{
    const { titulo, editorial } = req.body
    const response = await bdd.query(`INSERT INTO libro (titulo, editorial) VALUES ($1, $2)`,[titulo, editorial])
    res.json({
        message: 'Libro creada correctamente',
        body: {
            libro: { titulo, editorial }
        }
    })
}

const LeerLibro = async(req, res) =>
{
    const response = await bdd.query(`SELECT * FROM libro order by id_libro desc;`)
    res.json(response.rows);
}

const LeerLibroById = async(req, res) =>
{
    const id_libro = req.params.id_libro
    const response = await bdd.query(`SELECT * FROM libro WHERE id_libro = $1`, [id_libro]);
    res.json(response.rows)
}

const BorrarLibro = async(req, res) =>
{
    const id_libro = req.params.id_libro
    const response = await bdd.query(`DELETE FROM libro WHERE id_libro = $1`, [id_libro]);
    res.json(`Libro ${id_libro} eliminada correctamente.`)
}
```

```
async function ActualizarLibro(req, res)
{
    const {id_libro, titulo, editorial} = req.body

    const response = await bdd.query(`UPDATE libro SET titulo = $2, editorial = $3 WHERE id_libro = $1`, [
        id_libro, titulo, editorial])
    res.json('Libro actualizado correctamente.')
}

// autor
const InsertarAutor = async(req, res) =>
{
    const { nombre, apellido, fecha_nacimiento, pais } = req.body
    const response = await bdd.query(`INSERT INTO autor (nombre, apellido, fecha_nacimiento, pais) VALUES ($1, $2, $3, $4)`,
    [nombre, apellido, fecha_nacimiento, pais])
    res.json({
        message: 'Autor creada correctamente',
        body: {
            autor: { nombre, apellido, fecha_nacimiento, pais }
        }
    })
}

const LeerAutor = async(req, res) =>
{
    const response = await bdd.query(`SELECT * FROM autor order by id_autor desc;`)
    res.json(response.rows);
}

const LeerAutorById = async(req, res) =>
{
    const id_autor = req.params.id_autor
    const response = await bdd.query(`SELECT * FROM autor WHERE id_autor = $1`, [id_autor]);
    res.json(response.rows)
}
```

```
const BorrarAutor = async(req, res) =>
{
    const id_autor = req.params.id_autor
    const response = await bdd.query(`DELETE FROM autor WHERE id_autor = $1`, [id_autor]);
    res.json(`Autor ${id_autor} eliminada correctamente.`)
}

async function ActualizarAutor(req, res)
{
    const { id_autor, nombre, apellido, fecha_nacimiento, pais} = req.body

    const response = await bdd.query(`UPDATE autor SET nombre = $2, apellido = $3, fecha_nacimiento = $4, pais=$5 WHERE id_autor =
    [id_autor, nombre, apellido, fecha_nacimiento, pais])
    res.json('Autor actualizado correctamente.')
}

// autor_libro
const InsertarAutorLibro = async(req, res) =>
{
    const { autor, libro } = req.body
    const response = await bdd.query(`INSERT INTO autor_libro (autor, libro) VALUES ($1, $2)`,
    [autor, libro])
    res.json({
        message: 'Autor Libro creado correctamente',
        body: {
            autor: { autor, libro }
        }
    })
}

const LeerAutorLibro = async(req, res) =>
{
    const response = await bdd.query(`SELECT * FROM autor_libro order by id desc;`)
    res.json(response.rows);
}
```

```
const LeerAutorLibroById = async(req, res) =>
{
    const id = req.params.id
    const response = await bdd.query(`SELECT pi.id, pi.autor , pi.libro, i.titulo, i.editorial FROM
    autor_libro pi inner join libro i
    on pi.libro = i.id_libro where pi.autor = $1 order by i.id_libro ;`, [id]);
    res.json(response.rows)
}

const BorrarAutorLibro = async(req, res) =>
{
    const id = req.params.id
    const response = await bdd.query(`DELETE FROM autor_libro WHERE id = $1`, [id]);
    res.json(`Autor Libro ${id} eliminada correctamente.`)
}

async function ActualizarAutorLibro(req, res)
{
    const { id, autor, libro } = req.body

    const response = await bdd.query(`UPDATE autor_libro SET autor = $2, libro = $3 WHERE id = $1`,
    [id, autor, libro])
    res.json('Autor Libro actualizado correctamente.')
}
```

```
module.exports = {
    InsertarLibro,
    LeerLibro,
    LeerLibroById,
    BorrarLibro,
    ActualizarLibro,
    InsertarAutor,
    LeerAutor,
    LeerAutorById,
    BorrarAutor,
    ActualizarAutor,
    InsertarAutorLibro,
    LeerAutorLibro,
    LeerAutorLibroById,
    BorrarAutorLibro,
    ActualizarAutorLibro
}
```

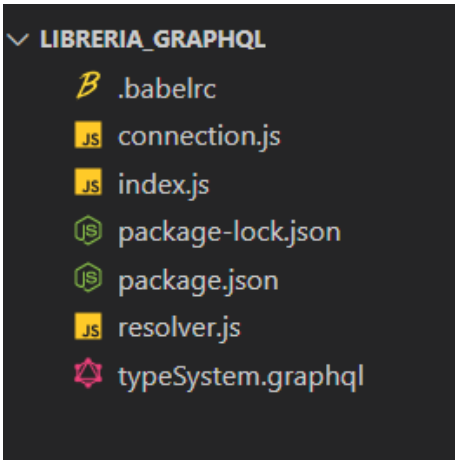
Anexo G: Routes del taller Librería con REST.

routes > index_routes.js > ...

```
const { Router } = require('express')
const router = Router()

const {
    InsertarLibro,
    LeerLibro,
    LeerLibroById,
    BorrarLibro,
    ActualizarLibro,
    InsertarAutor,
    LeerAutor,
    LeerAutorById,
    BorrarAutor,
    ActualizarAutor,
    InsertarAutorLibro,
    LeerAutorLibro,
    LeerAutorLibroById,
    BorrarAutorLibro,
    ActualizarAutorLibro
} = require('../controllers/controller_libreria')

// ruta libro
router.get('/libros', LeerLibro)
router.get('/libros/:id_libro', LeerLibroById)
router.post('/libros', InsertarLibro)
router.put('/libros', ActualizarLibro)
router.delete('/libros/:id_libro', BorrarLibro)

// ruta autor
router.get('/autor', LeerAutor)
router.get('/autor/:id_autor', LeerAutorById)
router.post('/autor', InsertarAutor)
router.put('/autor', ActualizarAutor)
router.delete('/autor/:id_autor', BorrarAutor)

```

```
// ruta autor libro
router.get('/autorlibro', LeerAutorLibro)
router.get('/autorlibro/:id', LeerAutorLibroById)
router.post('/autorlibro', InsertarAutorLibro)
router.put('/autorlibro', ActualizarAutorLibro)
router.delete('/autorlibro/:id', BorrarAutorLibro)

module.exports = router;
```

Anexo H: Arquitectura Librería con GraphQL.



Anexo I: Connection Librería con GraphQL.

```
connection.js > ...
1   const pgPromise = require('pg-promise');
2
3   const config = {
4       host: 'localhost',
5       port: '5432',
6       database: 'libros',
7       user: 'postgres',
8       password: 'qwerty123',
9   }
10
11  const pgp = pgPromise({});
12  const db = pgp(config);
13
14  exports.db = db;
```

Anexo J: Index Librería con GraphQL.

index.js > ...

```
const express = require("express");
const cors = require("cors");

const { graphqlExpress, graphiqlExpress } = require("graphql-server-express");
const bodyParser = require("body-parser");
const { importSchema } = require("graphql-import"); //Importar un schema
const { makeExecutableSchema } = require("graphql-tools");

const port = 3000;
const endPoint = "/libros";

const typeDefs = importSchema("./typeSystem.graphql");

import resolvers from "./resolver";

const schema = makeExecutableSchema({ typeDefs, resolvers });

let server = express().use(cors());

server.use(endPoint, bodyParser.json(), graphqlExpress({ schema, rootValue: resolvers }));
server.use("/graphiql", graphiqlExpress({ endpointURL: endPoint }));

server.listen(port, () => {
    console.log("Server on Port: ", port);
    console.log("Graphql localhost:" + port + endPoint);
    console.log("Graphiql: localhost:" + port + "/graphiql");
});
```

Anexo K: Resolvers Librería con GraphQL.

resolver.js > ...

```
const { db } = require("./connection");

const libroResolver = {
    Query: {
        async autores(root, args) {
            const isObjectEmpty = Object.entries(args).length == 0;
            if (isObjectEmpty) {
                return await db.any("SELECT * FROM autor ORDER BY id_autor  DESC");
            } else {
                return await db.any("SELECT * FROM autor WHERE id_autor =$1", [args.id_autor]);
            }
        },
        async libros(root, args) {
            const isObjectEmpty = Object.entries(args).length == 0;
            if (isObjectEmpty) {
                return await db.any("SELECT * FROM libro ORDER BY id_libro  DESC");
            } else {
                return await db.any("SELECT * FROM libro WHERE id_libro =$1", [args.id_libro]);
            }
        },
        async autoresLibros(root, { id_autor }) {
            const query = `SELECT i.id_libro, i.titulo, i.editorial FROM
      autor_libro pi inner join libro i
      on pi.libro = i.id_libro where pi.autor = $1 order by i.id_libro ;`;
            return await db.any(query, id_autor);
        }
    },
```

```
Autor: {
    libros(autor) {

        console.log(autor);
        return db.any(
            `SELECT libro.* FROM autor_libro,
          libro WHERE autor_libro.libro=libro.id_libro and
           autor_libro.autor=$1;`,
            [autor.id_autor]
        );
    },
},
Mutation: {
    async createAutor(root, { autor }) {
        if (autor === undefined) {
            return null;
        }
        console.log(autor);
        const query = `INSERT INTO public.autor(nombre, apellido, fecha_nacimiento, pais) VALUES ($1, $2, $3, $4) returning *`;
        let result = await db.one(query, [autor.nombre, autor.apellido, autor.fecha_nacimiento, autor.pais]);
        console.log(result);
        if (autor.libroIds && autor.libroIds.length > 0) {
            const query1 = `INSERT INTO autor_libro(autor, libro) VALUES ($1, $2)
             returning *`;

            for (const libroID of autor.libroIds) {
                await db.one(query1, [result.id, libroID]);
            }
        }

        return await db.any("SELECT * FROM autor ORDER BY id_autor DESC");;
    },
```

```
    async updateAutor(root, args) {
        const query = `UPDATE autor SET nombre=$1, apellido=$2, fecha_nacimiento=$3, pais=$4 where id_autor=$5 returning *;`;
        const autor = args.AutorU;

        let result = await db.one(query, [autor.nombre, autor.apellido, autor.fecha_nacimiento, autor.pais, args.idLast]);

        return await db.any("SELECT * FROM autor ORDER BY id_autor DESC");
    },

    async deleteAutor(root, { id }) {
        const query = `DELETE FROM autor WHERE id_autor=$1;`;
        const query2 = `DELETE FROM autor_libro WHERE autor = $1;`
        const query3 = `SELECT * FROM autor ORDER BY id_autor DESC;`
        await db.none(query2, id);
        await db.none(query, id);
        let result = await db.any(query3);
        return result;
    },
    async createLibro(root, { libro }) {
        const query = `INSERT INTO libro ( titulo, editorial ) VALUES($1, $2);`;
        await db.none(query, [libro.titulo, libro.editorial]);
        return await db.any("SELECT * FROM libro ORDER BY id_libro DESC");
    },
    async updateLibro(root, args) {
        const libro = args.libroU;
        const query = `UPDATE libro SET titulo=$1, editorial=$2 WHERE id_libro=$3;`;
        await db.none(query, [libro.titulo, libro.editorial, args.idLast]);
        return await db.any("SELECT * FROM libro ORDER BY id_libro DESC");
    },
    async deleteLibro(root, { id }) {
        const query = `DELETE FROM libro WHERE id_libro = $1`;
        const query2 = `DELETE FROM autor_libro WHERE libro = $1;`
        await db.none(query2, id);
        await db.none(query, id);
        return await db.any("SELECT * FROM libro ORDER BY id_libro DESC");
    },
```

```
        async createAutorLibro(root, { idAutor, idLibro }) {
            const query = `INSERT INTO autor_libro (autor, libro)
                    VALUES($1,$2);`;

            const query3 = `SELECT * FROM autor_libro WHERE
                    autor = $1 and libro = $2;`;
            const query4 = `SELECT i.id_libro , i.titulo, i.editorial FROM
                    autor_libro pi inner join libro i
                    on pi.libro = i.id_libro where pi.autor = $1 order by i.id_libro ;`;

            const findRecord = await db.oneOrNone(query3, [idAutor, idLibro]);

            if (!findRecord) {
                await db.none(query, [idAutor, idLibro]);
                return await db.any(query4, idAutor);
            } else {
                return await db.any(query4, idAutor);
            }

        },
        async deleteAutorLibro(root, { idAutor, idLibro }) {
            const query = `DELETE FROM autor_libro WHERE autor = $1 AND libro = $2;`;
            const query2 = `SELECT i.id_libro, i.titulo, i.editorial FROM
            autor_libro pi inner join libro i
            on pi.libro = i.id_libro where pi.autor = $1 order by i.id_libro ;`;
            await db.none(query, [idAutor, idLibro]);
            return await db.any(query2, idAutor);
        }
    },

};

export default libroResolver;
```

Anexo L: TypeSystem Librería con GraphQL.

typeSystem.graphql

```graphql
#Scalars: Int, Floatm String,Boolean, ID
#Querys
type Query {
  autores(id_autor: Int): [Autor]
  libros(id_libro: Int): [Libro]
  autoresLibros(id_autor: Int):[Libro]
}

#This is my types:
type Autor {
  id_autor: Int! #! indica que es un campo obligatorio
  nombre: String
  apellido: String
  fecha_nacimiento: String!
  pais: String
  libros: [Libro] #Arreglo de tipo libro
}

type Libro {
  id_libro: Int!
  titulo: String!
  editorial: String
}

type Mutation {
  createAutor(autor: AutorInput): [Autor]
  updateAutor(idLast: Int, AutorU: AutorInput): [Autor]
  deleteAutor(id: Int): [Autor]

  createLibro(libro: LibroInput):[Libro]
  updateLibro(idLast: Int, LibroU: LibroInput): [Libro]
  deleteLibro(id: Int): [Libro]

  createAutorLibro(idAutor: Int!, idLibro: Int!):[Libro]
  deleteAutorLibro(idAutor: Int!, idLibro: Int!):[Libro]
}
```

```graphql
input AutorInput {
  nombre: String!
  apellido: String!
  fecha_nacimiento: String!
  pais: String!
  libroIds: [Int]
}

input LibroInput{
  titulo: String!
  editorial: String!
}

input AutorInputUpdated {
  idLast: Int!
  nombre: String!
  apellido: String!
  fecha_nacimiento: String!
  pais: String!
  libroIds: [Int]
}
```

Anexo L: Babel Librería con GraphQL.

```
.babelrc > ...
1  {
2      "presets": [
3          "@babel/preset-env"
4      ]
5  }
```

Anexo G: Ejecución virtual del experimento controlado debido a la pandemia Covid-19, con estudiantes de la materia d Construcción de Software de la carrera de Ingeniería de Software de la Universidad Técnica del Norte.