

 | UNIVERSIDAD
SEÑOR DE SIPÁN

**FACULTAD DE INGENIERÍA, ARQUITECTURA Y
URBANISMO**

**ESCUELA PROFESIONAL DE INGENIERÍA DE
SISTEMAS**

TESIS

**EVALUACIÓN DE LA EFICIENCIA DE LAS
TECNOLOGÍAS GRAPHQL Y REST EN LA
IMPLEMENTACIÓN DE SERVICIOS WEB
CONSUMIDOS POR APLICACIONES ANDROID**

**PARA OPTAR EL TÍTULO PROFESIONAL DE
INGENIERO DE SISTEMAS**

Autor:

Bach. Angulo Vásquez Jorge Luis Mateo

Asesor:

Mg. Mejía Cabrera Heber Ivan

Línea de Investigación:

Infraestructura, Tecnología y Medio Ambiente

Pimentel, Perú

2020

DEDICATORIA

A mis padres que me apoyaron para poder culminar mi carrera profesional y lograr mis objetivos.

AGRADECIMIENTO

En primer, darle gracias a Dios, por brindarme sabiduría y fortaleza para culminar la etapa universitaria.

A la Universidad Señor de Sipán por brindarme a través de sus profesores, conocimientos que me sirven para desarrollarme en mi vida profesional.



RESUMEN

En el presente estudio se planteó como objetivo general evaluar la eficiencia de las tecnologías GraphQL y Rest en la implementación de servicios web consumidos desde aplicaciones móvil Android, bajo la norma ISO/IEC 25010. Para lograr ello, se ha sustentado en una investigación aplicada y cuantitativa, basándose en un diseño cuasi experimental, considerándose como muestra a las tecnologías Rest y GraphQL, empleándose como técnicas de investigación a la observación, análisis y experimentación. Como resultados se obtuvo que la arquitectura empleada en ambas tecnologías fue de tipo cliente - servidor, con servicio de multiplataforma, implementándose un API con procesos básicos tales como registrar clientes, los productos y venderlos, los cuales serán compatibles en dispositivos móviles que operan bajo el sistema Android. Los indicadores para medir el rendimiento en ambas tecnologías utilizadas fueron el tiempo (milisegundos) y recursos de RAM (megabytes), siendo el consumo de RAM por REST de 6MB en cuanto a la lista de datos y 7MB en cuanto al registro de datos, asimismo con GRAPHQL, se consume 3 MB cuando se listan datos y 5 MB cuando se registran datos; en cuanto al tiempo de respuesta, REST obtuvo 178 milisegundos en cuanto al listado de datos y 203 milisegundos en el registro de datos, por otro lado, en GraphQL el tiempo total fue de 129 milisegundos en cuanto al listado de datos y 109 milisegundos en el registro de datos. Concluyéndose que la tecnología GRAPHQL, resulta ser la mejor opción en cuanto a consumo de tiempo y recursos acorde con lo estipulado en la norma ISO/IEC 25010 para las aplicaciones que son compatibles en dispositivos Android.

Palabras claves: tecnologías, GRAPHQL, REST, aplicaciones.

ABSTRAC

In this study, the general objective was to evaluate the efficiency of GraphQL and Rest technologies in the implementation of web services consumed from Android mobile applications, under ISO / IEC 25010. To achieve this, it has been based on an applied research and quantitative, based on a quasi-experimental design, considering Rest and GraphQL technologies as a sample, using observation, analysis and experimentation as research techniques. As a result, it was obtained that the architecture used in both technologies was client-server, with multiplatform service, implementing an API with basic processes such as registering customers, products and selling them, which will be compatible in mobile devices that operate under the Android system. The indicators to measure the performance in both technologies used were time (milliseconds) and RAM resources (megabytes), with RAM consumption by REST of 6MB in terms of the data list and 7MB in terms of data logging, also with GRAPHQL, 3 MB is consumed when data is listed and 5 MB when data is recorded; Regarding the response time, REST obtained 178 milliseconds in terms of the data list and 203 milliseconds in the data register, on the other hand, in GraphQL the total time was 129 milliseconds in terms of the data list and 109 milliseconds in the data register. Concluding that GRAPHQL technology, it turns out to be the best option in terms of time and resources consumption in accordance with the stipulated in ISO / IEC 25010 for applications that are compatible on Android devices.

Keywords: technologies, GRAPHQL, REST, applications.



ÍNDICE

DEDICATORIA.....	2
AGRADECIMIENTO.....	3
RESUMEN.....	4
ABSTRAC.....	5
ÍNDICE	6
CAPÍTULO I: INTRODUCCIÓN	14
1.1. REALIDAD PROBLEMÁTICA.....	14
1.2. ANTECEDENTES DE ESTUDIO	15
<i>A nivel internacional.....</i>	<i>15</i>
<i>A nivel Nacional</i>	<i>19</i>
<i>A nivel Local.....</i>	<i>19</i>
<i>A nivel Nacional y Local.....</i>	<i>20</i>
1.3. TEORÍAS RELACIONADAS AL TEMA	20
1.3.1 <i>Arquitectura Cliente Servidor.....</i>	<i>20</i>
1.3.2. <i>Protocolos de Comunicación en la arquitectura Cliente Servidor.....</i>	<i>21</i>
1.3.3. <i>Servicios Web y API.....</i>	<i>23</i>
1.3.4. <i>Diseño de Web API.....</i>	<i>25</i>
1.3.5. <i>Tecnologías Para la implementación de un Servicio Web API.....</i>	<i>25</i>
1.3.6. <i>Eficiencia de una Web API.....</i>	<i>34</i>
1.3.7. <i>ISO/IEC 25010.....</i>	<i>34</i>
1.3.8. <i>Dimensiones de Eficiencia de ISO/IEC 25010</i>	<i>35</i>
1.4. FORMULACIÓN DEL PROBLEMA	36
1.5. JUSTIFICACIÓN E IMPORTANCIA DEL ESTUDIO	37
1.6. HIPÓTESIS	38
1.7. OBJETIVOS	38
1.7.1. <i>Objetivo General</i>	<i>38</i>
1.7.2. <i>Objetivos Específicos.....</i>	<i>38</i>
CAPÍTULO II: MATERIAL Y MÉTODO.....	40
2.1. TIPO Y DISEÑO DE INVESTIGACIÓN	40
2.1.1. <i>Tipo de Investigación.....</i>	<i>40</i>
2.1.2. <i>Diseño de Investigación.....</i>	<i>40</i>
2.2. POBLACIÓN Y MUESTRA	40
2.2.1. <i>Población.....</i>	<i>40</i>
2.2.2. <i>Muestra.....</i>	<i>40</i>



2.3. VARIABLES, OPERACIONALIZACIÓN.....	41
2.3.1. <i>Variable Independiente</i>	41
2.3.2. <i>Variable Dependiente</i>	41
2.3.3 <i>Operacionalización de las Variables</i>	42
2.4. TÉCNICAS E INSTRUMENTOS DE RECOLECCIÓN DE DATOS, VALIDEZ Y CONFIABILIDAD	43
2.5. PROCEDIMIENTO DE RECOLECCIÓN Y ANÁLISIS DE DATOS	44
2.6. CRITERIOS ÉTICOS.....	45
2.7. CRITERIOS DE RIGOR CIENTÍFICO	46
CAPÍTULO III: RESULTADOS	48
3.1. RESULTADOS EN TABLAS Y FIGURAS.....	48
3.2. DISCUSIÓN DE RESULTADOS.....	72
3.3. APORTE PRÁCTICO.....	74
3.3.1. <i>Seleccionar la arquitectura de comunicación que establecen las tecnologías GraphQL y Rest</i>	74
3.3.2. <i>Seleccionar las dimensiones e indicadores de medición de calidad de software para las tecnologías GraphQL y REST, según la norma ISO/IEC 25010</i>	76
3.3.3. <i>Seleccionar procesos CRUD se tomarán en cuenta para la medición de llamadas a la API para las tecnologías GraphQL y REST</i>	78
3.3.4. <i>Implementar un API tanto en GraphQL y REST en base a la metodología SCRUM</i>	80
3.3.5. <i>Evaluar los resultados para las tecnologías GraphQL y Rest</i>	118
CAPÍTULO IV: CONCLUSIONES Y RECOMENDACIONES	120
REFERENCIAS	123
ANEXOS.....	128



ÍNDICE DE TABLAS

Tabla 1. Cuadro de operacionalización de variables.	42
Tabla 2. Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Listado de datos.....	48
Tabla 3. Llamadas desde la API REST – Listado de datos	49
Tabla 4. Visualización del promedio de Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Listado de datos.....	49
Tabla 5. Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Listado de datos	50
Tabla 6. Llamadas desde la API GRAPHQL por consumo de RAM – Listado de datos ...	51
Tabla 7. Visualización del promedio del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Listado de datos.....	52
Tabla 8. Comparación de promedios de Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL –Listado de datos.....	53
Tabla 9. Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Registro de datos.....	54
Tabla 10. Llamadas desde la API REST por consumo de RAM – Registro de datos	55
Tabla 11. Visualización del promedio de Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Registro de datos	55
Tabla 12. Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Registro de datos	56
Tabla 13. Llamadas desde la API GRAPHQL por consumo de RAM – Registro de datos	57
Tabla 14. Visualización del promedio del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Registro de datos	58



Tabla 15. Comparación de los promedios de Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL –Registro de datos 59

Tabla 16. Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Listado de datos 60

Tabla 17. Llamadas desde la API REST por consumo de RAM – Listado de datos..... 61

Tabla 18. Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Listado de datos 61

Tabla 19. Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Listado de datos 62

Tabla 20. Llamadas desde la API GRAPHQL por tiempo de respuesta– Listado de datos 63

Tabla 21. Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Listado de datos..... 64

Tabla 22. Comparación de promedios del tiempo de respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL –Listado de datos 65

Tabla 23. Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Registro de datos..... 66

Tabla 24. Llamadas desde la API REST por tiempo de respuesta – Registro de datos..... 67

Tabla 25. Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Registro de datos 68

Tabla 26. Llamadas desde la API GRAPHQL por tiempo de respuesta – Registro de datos 69

Tabla 27. Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –registro de datos 70

Tabla 28. Comparación del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL –Registro de datos 71



Tabla 29. Cuadro comparativo entre arquitecturas de REST y GRAPHQL.	74
Tabla 30. Cuadro de valoración de tecnologías basado en la ISO 25010.....	77
Tabla 31. Cuadro de valoración y puntaje para el grado de importancia de procesos CRUD involucrados en la medición del rendimiento.....	79
Tabla 32. Cuadro de valoración y puntaje para el grado de importancia de procesos CRUD involucrados en la medición del rendimiento.....	79
Tabla 33. Roles Scrum.....	85
Tabla 34. Lista de historias de usuario	86
Tabla 35. Priorización de historias	89
Tabla 36. Primer Sprint.....	90
Tabla 37. Segundo Sprint	91
Tabla 38. Rendimiento de las tecnologías REST y GRAPHQL	118

ÍNDICE DE FIGURAS

Figura 1. Comunicación en la Arquitectura Cliente-Servidor	21
Figura 2. End Point get/users/<id> que devuelve el ID del usuario	29
Figura 3. End Point get/users/<id>/posts que devuelve las publicaciones del usuario	29
Figura 4. End Point get/users/<id>/followers que lista los seguidores del usuario	30
Figura 5. Petición GraphQL que devuelve el ID, los posts y los seguidores del usuario. Respuesta específica del servidor.	31
Figura 6. Esquema del servicio de GraphQL, se pueden visualizar los Types y sus relaciones.	33
Figura 7. Características de calidad del producto software según ISO/IEC 25010.....	35
Figura 8. Popularidad de GraphQL en los últimos 5 años, del 2014 al 2019.....	41
Figura 9: Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST -Listado de datos	48
Figura 10: Visualización del promedio de Consumo de RAM (Mb) del proceso de llamadas desde la API REST - Listado de datos	49
Figura 11: Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL - Listado de datos.....	50
Figura 12: Visualización del promedio del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Listado de datos.....	52
Figura 13: Comparación de promedios de Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL –Listado de datos.....	53
Figura 14: Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Registro de datos.....	54
Figura 15: Visualización del promedio de Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Registro de datos	55
Figura 16: Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Registro de datos	56
Figura 17: Visualización del promedio del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Registro de datos	58
Figura 18: Comparación de los promedios de Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL –Registro de datos	59



Figura 24: Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Listado de datos 60

Figura 25: Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Listado de datos 61

Figura 26: Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Listado de datos 62

Figura 27: Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Listado de datos..... 64

Figura 28: Comparación de promedios del tiempo de respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL –Listado de datos 65

Figura 19: Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Registro de datos..... 66

Figura 21: Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Registro de datos 68

Figura 22: Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –registro de datos 70

Figura 23: Comparación del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL –Registro de datos 71

Figura 29: Línea de tiempo de tecnologías para el desarrollo de aplicaciones móviles..... 76

Figura 38: Valoración para procesos CRUD. 79

Figura 29. Metodología SCRUM. 81

Figura 30. Dueño del producto 83

Figura 31. Equipo de desarrollo..... 84

Figura 32. Scrum Master 84

Figura 33. Product Backlog. 85

Figura 35. Historia de usuario. 86

Figura 34. Planificación del Sprint. 89

Figura 36. Gráfica de la distribución para arquitectura de solución..... 92

Figura 39. Modelo de datos 92

Figura 39. Diagrama de clases..... 93

Figura 40. Interfaces de datos de clientes 94

Figura 41. Interfaces de registro de clientes. 95



Figura 42. Interfaces de modificar y eliminar clientes.	96
Figura 43. Interfaces de información de productos.	97
Figura 44. Interfaces de modificar y eliminar productos.	98
Figura 45. Interfaces de registro de pedidos	99
Figura 46. Interfaz de registro de ventas	100
Figura 47. Interface de proceso productos.....	101
Figura 48. Interfaz de selección de API.....	102
Figura 49. Interfaz de resultados REST - GRAPHQL.	116
Figura 50. Gráfico de indicador de tiempo REST - GRAPHQL.	117
Figura 51. Gráfico de indicador de consumo RAM REST-GRAPHQL.	117



CAPÍTULO I: INTRODUCCIÓN

1.1. Realidad Problemática

Según Semere (2017) las aplicaciones móviles o web (que representan el lado del cliente) para que puedan realizar una interacción con el servidor requieren de mucho tiempo y esfuerzo, por esta razón, se implementa una API para efectuar la comunicación con el servidor. En ese esfuerzo, las tecnologías con las cuales se construyen las APIS se enfrentan a algunos problemas. Siendo ello así, el primer factor que se debe atender es la cantidad de datos que la tecnología de la API es capaz de manipular, en aplicaciones con grandes cantidades de datos está el problema de la latencia, es decir, existe un retraso en el tiempo de respuesta a una petición al servidor.

Aunado a ello, un estudio realizado por el investigador Pasma (2018) de la Universidad Danesa Aarhus, señala que se debe considerar la cantidad de peticiones que realiza una API o la frecuencia de las llamadas al servidor, es decir, cuando estas se efectúan con una alta frecuencia, toda la tecnología y arquitectura de la API necesitan tener una flexibilidad alta para poder servir los datos solicitados con un tiempo de respuesta relativamente pequeño o insignificante.

En ese orden de ideas, otra parte del problema es el consumo de los recursos del hardware que da soporte a la arquitectura de la aplicación. Por lo que, cada interacción de la API con el cliente (aplicación móvil) o con el servidor, implica un consumo de recursos, sin embargo, la tecnología que implementa la API, debería ser capaz de optimizar recursos para evitar sobrecargar el flujo de datos y para realizar un procesamiento rápido de los mismos (Pasma, 2018).

Asimismo, se presenta el problema de la escalabilidad, cuando se necesita que la aplicación tenga nuevas funcionalidades que involucren aumento en la frecuencia de las llamadas a la API o del mecanismo en que estas llamadas se realizaban, o cuando la naturaleza de los datos cambie y ahora existan fluctuaciones como por ejemplo en aplicaciones de tiempo real, entonces, la tecnología de la API presenta un reto de escalabilidad, para dar soporte a este nuevo orden de peticiones



y mantener la disponibilidad de los datos, conforme lo delimitan (Braubach et all., 2017).

En ese sentido, cabe recalcar que, en las aplicaciones del mundo moderno, todos los factores antes expuestos son determinantes para que la aplicación funcione óptimamente, de lo contrario, solo resultará en pérdidas de tiempo y recursos para los involucrados.

1.2. Antecedentes de Estudio

A nivel internacional

Brito, Bombach, Valente (2019) en su investigación desarrollada en la Universidad de Minas Gerais en Brasil, denominada “Migrating to GraphQL: A practical assement”, plantearon la migración de la tecnología Rest a GraphQL en las APIs que tenían implementadas, con el objetivo de mejorar la eficiencia de los sistemas en base a la reducción del tamaño de los archivos y la reducción del número de campos devueltos por las llamadas a las APIS. Para ello, realizaron un análisis comparativo de ventajas y desventajas de estas tecnologías. Este análisis requirió de la migración de 7 sistemas para utilizar APIS basadas en GraphQL en lugar de las APIS basadas en Rest que estaban ejecutadas. Las APIS que se tomaron como referencia fueron las proporcionadas por Github y arXiv. Como resultado se tuvo que, las nuevas implementaciones basadas en GraphQL demostraron que el tamaño de los archivos JSON devueltos por la API disminuyó en un 99%, así como, el número de campos devueltos por la llamada a la API se redujo en un 94%.

Helgason (2017) realizó un trabajo de investigación en la Escuela de informática Universidad de Skoude en Suecia, denominado “Performance analysis of web services: Comparison between RestFull & GraphQL web services”, mediante el cual, expuso la comparación de rendimiento entre las tecnologías de servicios web GraphQL y Rest bajo la perspectiva del manejo de bases de datos relacionales. El método que se propuso fue implementar una base de datos relacional compleja, con claves foráneas, relaciones múltiples y realizar peticiones a esta base de datos, tanto con GraphQL como con Rest, a través de una red local ethernet para medir algunos

factores como: la latencia de entrega de los datos, el tamaño de los datos de cada respuesta del servidor. Estas pruebas se realizaron múltiples veces para obtener datos más confiables. Entre los resultados obtenidos: Los datos requeridos pudieron realizarse en 1 sola llamada desde la API implementada con GraphQL; sin embargo, se necesitaron 3 llamadas con sus respectivos End Points desde la API implementada con Rest. Respecto a la latencia se obtuvo que, en consultas rápidas y sencillas Rest es significativamente más eficiente que GraphQL; siendo que, en las consultas que involucran combinación de campos y tablas, GraphQL es más eficiente en una proporción de 1 a 4 respecto a Rest.

Gustavsson y Stenlund (2016) en su investigación desarrollada en Suecia, titulada “Efficient data communication between a webclient and a cloud environment” determinaron la diferencia de utilizar GraphQL o Rest para la implementación de una API, además desarrollaron un modelo de decisión para determinar con precisión objetiva los escenarios en los cuales es más eficiente utilizar GraphQL o Rest para construir una API. Para lograr su cometido, proporcionaron prototipos de cada API bajo el esquema de la API de Apache Axis, para facilitar las llamadas a los datos. De las llamadas a la API se obtuvieron los tiempos de respuesta para cada uno de los prototipos implementados. En base a estos valores se formuló el modelo de decisión a través de entrevistas y experiencias de los desarrolladores de software. Entre los resultados se encontró que, Rest es significativamente más flexible y abstracto que GraphQL por tener varias opciones tecnológicas para efectuar las llamadas; sin embargo, por la misma razón, desarrollar una API con GraphQL requiere de menos esfuerzo y tiempo que desarrollarla con Rest.

Pasma (2018) en su investigación desarrollada en Dinamarca, titulada “Web API Capacity Study. A comparison of REST and GraphQL”, evaluó el impacto que tiene el uso de una API en la capacidad de un sistema que ejecuta peticiones repetitivas, las tecnologías evaluadas fueron GraphQL y Rest. Para ello, se tomaron en cuenta indicadores como la tasa de uso del CPU, limitaciones en la red (ancho de banda y tipo de conexión) y el tiempo total de la transacción. La metodología para realizar las pruebas fue generar carga en el servidor que simularan las peticiones, en bloques de 50 peticiones con intervalos de 1 segundo por bloque. En los resultados,

se determinó que, el consumo de CPU fue de 37% con la tecnología GraphQL; en cambio, con la tecnología Rest el CPU mostró un consumo de 43% por cada bloque enviado. Además, se reportó que mientras los bloques de peticiones aumentan, la brecha en el consumo de CPU entre Rest y GraphQL crece exponencialmente, pero cuando el bloque de peticiones disminuye, esta brecha tiende a disminuir también. De esta manera, se concluyó que, para implementar una API que mantenga una capacidad óptima en sistemas con múltiples peticiones, la tecnología GraphQL tiene una eficiencia de más del 100% en comparación a la tecnología Rest.

Eizinger (2017) en su trabajo desarrollado en la Universidad de Ciencias Aplicadas Technikum Wien en Austria, denominada “API Design in Distributed Systems: A Comparison between GraphQL and REST”, realizó una investigación con un enfoque teórico para determinar las diferencias entre GraphQL y Rest en cuanto a la eficiencia de estas tecnologías en sistemas distribuidos. Para lograr resultados fiables, se consideraron factores de medición de eficiencia tales como: Reutilización de componentes, el rendimiento en base a consumo de recursos, la sobrecarga en base a manejo de peticiones, y por último la reutilización de operaciones. Además, para los resultados, el autor planteó tomar en cuenta el tamaño y la información de la cabecera de metadatos que devuelve cada tecnología (Rest o GraphQL) frente a una petición. Al ser una comparación teórica, el investigador no demostró cuantificablemente cuál es la tecnología adecuada o correcta, sino que estableció aspectos relevantes para clasificar las tecnologías según el escenario y los requerimientos de los sistemas distribuidos.

RitzilÄ (2017) realizó un trabajo de investigación en Finlandia, llamado “GraphQL: The API Design Revolution”, con el objetivo de establecer las diferencias entre la nueva tecnología GraphQL con la arquitectura Rest. Su trabajo se basó en la recopilación teórica de factores o puntos críticos de comparación para ambas tecnologías, entre los factores se consideraron: la estructura de los end points de comunicación, la respuesta obtenida ante una petición, el porcentaje de utilización de las tecnologías, la gestión del almacenamiento en memoria caché y la experiencia de los desarrolladores. Los resultados que se obtuvieron demostraron que Rest es más fácil de entender y de manipular respecto a la especificación de los verbos HTTP

de comunicación en sus endpoints; en cambio, GraphQL puede resultar de más difícil comprensión y es menos intuitivo en este aspecto; sin embargo, por esa misma razón, GraphQL ejecuta más rápido y con menos código sus peticiones al servidor. Otro punto importante en los resultados fue que GraphQL está siendo más utilizado que Rest en sistemas distribuidos modernos que involucran gran carga y flujo de datos, debido a su excelente gestión del almacenamiento y su flexibilidad en el consumo desde aplicaciones Android.

Cechak (2017) en su investigación en República Checa, denominada “Using GraphQL for Content Delivery in Kentico Cloud”, exploró la tecnología GraphQL para la implementación de la API de entrega o delivery para el sistema CMS Kentico Cloud que inicialmente estaba funcionando con la tecnología RestFull. Para ello, realizó experimentos y pruebas de rendimiento con peticiones típicas que simulaban el tráfico propio de un CMS. Se eligieron 5 tipos de consultas de muestra que generalmente se utilizan en la API de delivery, para cada tipo de consulta se realizó 100 veces cada llamada de prueba. Luego de realizar las pruebas, se analizó el tamaño del archivo de respuesta que generaron ambas tecnologías, así como el tiempo total de respuesta del servidor. Según estas métricas, se concluyó que, la nueva API construida con GraphQL era 14.3% más eficiente que la API Restfull existente para el servicio de delivery del CMS Kentico Cloud.

Eeda (2018) realizó un trabajo de investigación en la Universidad de Western Ontario Canadá, denominado “Rendering real-time dashboards using a GraphQLbased UI Architecture”, mediante el cual se propuso implementar una API basada en GraphQL que se pueda integrar con el sistema basado en la nube BlueMix, para mejorar la eficiencia y rendimiento en el flujo de datos y gráficos estadísticos en tiempo real. El enfoque que se adoptó fue tomar solo los datos específicos que solicitaba el usuario para evitar la sobrecarga del servidor, además se consideró gestionar el almacenamiento en caché y la resolución de peticiones de manera asíncrona para cargar los datos de manera rápida y fluida. Los resultados obtenidos de las pruebas, demostraron que, la implementación de la API con GraphQL procesa las peticiones con una eficiencia en cuanto a rendimiento del 245% y en cuanto a tiempo de respuesta con una tasa de 153% más rápido que el sistema existente.

A nivel Nacional

Atencio y Mamani (2017) realizaron un trabajo de investigación en Puno, denominado “Interconectividad basada en API REST en Aplicaciones de la municipalidad provincial de Lampa”, con el objetivo de desarrollar una plataforma integral que mejorara la eficiencia de comunicación entre los diferentes sistemas y aplicaciones de la referida municipalidad. Para ello, se planteó un modelo que tuvo como principal componente la implementación de un servicio web basado en la arquitectura REST. Luego, se determinó el rendimiento del desarrollo basado en REST, evaluando esta arquitectura en base a la cantidad de usuarios que utilizan las aplicaciones y el número de trámites promedio que se realizan en un mes.

Entre los resultados obtenidos están la optimización del tiempo de los trámites al utilizar la API REST, optimización de los recursos del servidor y otros recursos tecnológicos de la municipalidad, la API REST proporcionó un sistema escalable y permitió la integración de las aplicaciones en una sola interfaz de comunicaciones, mejorando significativamente el flujo de la información.

A nivel Local

Burgos (2017) en su investigación realizada en Chiclayo, denominada “Análisis y evaluación de las arquitecturas REST y SOAP para el desarrollo de servicios web aplicados al ERP AdrisERP y su versión móvil en Android”, llevó a cabo un análisis y evaluación de rendimiento y escalabilidad de las tecnologías REST y SOAP en la implementación del servicio web. Para lograr su objetivo, se construyó dicho web service y se realizaron pruebas de carga a cada tecnología, se utilizó el simulador de peticiones Insomnia desde donde se ejecutaron las llamadas al servidor. El factor de medición de rendimiento fue el tiempo de respuesta del servidor, obteniendo un valor de 1515.7 ms para REST y un valor de 847397666.7 ms para SOAP, como se puede observar, el resultado fue que REST ofrece un rendimiento significativamente mayor al que ofrece SOAP en la implementación de servicios web.



A nivel Nacional y Local

Los trabajos previos documentados en este ámbito, están parcialmente relacionados con esta investigación.

En este sentido, solo se encontraron investigaciones que estudian y evalúan a la tecnología REST, en el trabajo de ámbito nacional la estudian no tan detalladamente, y en la investigación de ámbito local la estudian comparándola con SOAP. Así mismo, no se encontraron trabajos previos relacionados con GraphQL, toda vez que, esta es una tecnología emergente, que en las aplicaciones que se utilizan y vienen implementándose en Latinoamérica aún no se desarrolla por completo.

1.3. Teorías Relacionadas al tema

1.3.1 Arquitectura Cliente Servidor

En palabras de Albertos (2018) la terminología Cliente-Servidor se utiliza para describir un modelo informático para el desarrollo de sistemas. Este modelo se basa en la distribución de funciones entre dos tipos de procesos independientes y autónomos: Servidor y Cliente.

Siendo ello así, se entiende por cliente al proceso que solicita servicios específicos del proceso propiamente del servidor. Por otro lado, un servidor es un proceso que proporciona los servicios solicitados para el cliente.

En esa misma línea, Chandra y Kumar (2009) indican que, los procesos de Cliente y Servidor pueden residir en la misma o en diferentes computadoras vinculadas por una red. Asimismo, cuando los procesos de Cliente y Servidor residen en dos o más computadoras independientes en una red, el Servidor puede proporcionar servicios para más de un Cliente. Además, un cliente puede solicitar servicios de varios servidores en la red sin tener en cuenta la ubicación o las características físicas de la computadora en la que reside el proceso del Servidor.

En ese orden de ideas, a continuación, se presenta la Figura 1 mediante la cual se puede advertir cómo opera la arquitectura cliente-servidor en el flujo de la información en un sistema informático:

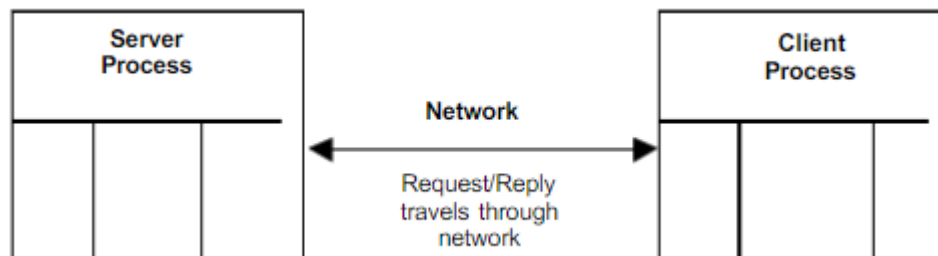


Figura 1. Comunicación en la Arquitectura Cliente-Servidor

Fuente: Chandra, S., & Kumar, S. (2009). An Introduction to Client /Server Computing [Figura]. Recuperado de

https://www.academia.edu/11334685/An_Introduction_to_Client_Server_Computing

1.3.2. Protocolos de Comunicación en la arquitectura Cliente Servidor

Para que el cliente establezca una comunicación sin fallos con el servidor, se utilizan diferentes protocolos de red, de acuerdo a la naturaleza de los datos que se han de intercambiar, por motivos de interés para esta investigación se estudió el protocolo HTTP.

1.3.2.1. HTTP

Según Singh y Kumar (2016) HTTP (Hypertext Transfer Protocol) es el protocolo principal utilizado por la World Wide Web (WWW) para la comunicación. Se basa en el modelo cliente servidor y es el encargado de definir el formato y la manera como se transmitirán los mensajes desde que (el cliente) efectúa la petición hasta que se devuelve una respuesta por parte del (servidor).

Asimismo, cabe precisar que, un navegador puede funcionar como un cliente HTTP, toda vez que se usa para enviar solicitudes al servidor. Y, posterior a la acción antes referida, es el servidor quien envía la respuesta de la información

solicitada al cliente. Siendo ello así, el puerto predeterminado por el servidor para escuchar la solicitud es el puerto 80.

Es así como, Gustavsson y Stenlund (2016) consideran que la función principal de HTTP es transmitir recursos a través de internet. Un recurso puede ser un archivo, un script o un documento escrito en cualquier formato.

Por otro lado, Singh (2016) señala que las características principales de HTTP son las siguientes:

- a) **HTTP es un protocolo sin conexión.** Significa que el cliente realiza una solicitud HTTP y luego se desconecta del servidor y espera la respuesta del mismo. El servidor después de procesar la solicitud envía respuesta al cliente.
- b) **HTTP es un protocolo independiente de los medios.** Significa que HTTP puede enviar cualquier tipo de datos.
- c) **HTTP es un protocolo sin estado.** Significa que cada transferencia de datos implica una conexión independiente de la anterior.

Además de lo antes descrito, es relevante desarrollar en esta investigación, los métodos que maneja HTTP para establecer la comunicación sin fallas, siendo los siguientes:

1.3.2.2. Métodos HTTP para la comunicación

Es importante señalar que, cuando se realiza una petición al servidor utilizando el protocolo HTTP se debe efectuar mediante métodos o verbos HTTP, según lo referido por Fielding et. Al (2000) los principales métodos son:

GET

Es el método más común utilizado por HTTP. Se utiliza para recuperar la información solicitada, mediante un enlace debidamente estructurado denominado URI.

HEAD

Este método es similar al método antes referido, sin embargo, no devuelve los datos solicitados. Sino que se utiliza para obtener metadatos, como el encabezado, línea de estado, código de respuesta del servidor, etc.

POST

También se utiliza para obtener información, pero de un elemento específico indicado a través de la URI, se pueden solicitar los datos que se deseen conocer del elemento en cuestión.

PUT

Se utiliza para escribir o guardar información de un elemento específico, los metadatos que se desean escribir se pueden especificar en la URI y el servidor buscará ese elemento y guardará los datos indicados.

DELETE

Se utiliza para eliminar un elemento específico del servidor, dicho elemento es identificado a través de la URI desde el cliente.

1.3.3. Servicios Web y API

1.3.3.1. Servicios Web

Según López (2015) un servicio web ofrece una interfaz que permite a los clientes interactuar con los servidores de una manera más general que como los navegadores lo hacen. Los clientes acceden a las operaciones en la interfaz de un servicio web a través de solicitudes y respuestas con formato en XML (eXtensible Markup Language) y generalmente se transmite a través protocolos HTTP (Hypertext Transfer Protocol)

Asimismo, Castro et al. (2013) indica que estos servicios proporcionan una infraestructura para mantener de una forma idónea y más estructurada la



interoperabilidad entre clientes y servidores. En particular, permiten que aplicaciones complejas a ser desarrolladas por prestación de servicios que integren a otros servicios, no importando las tecnologías o lenguajes de programación utilizados. Es decir, los servicios web pueden actuar como componentes independientes que se integran entre sí para formar sistemas distribuidos complejos.

1.3.3.2. API

Una Application Programming Interface, es un término que se utiliza en diferentes contextos dentro del mundo de la ingeniería de software. En general, se refiere a la interfaz de un elemento de software que puede ser llamado o ejecutado. (Eizinger, 2017).

Asimismo, en palabras de Wittern et al. (2017) los proveedores de terceros necesitan poder escribir programas y que los mismos puedan interactuar con otros; en mérito a ello, se requiere utilizar la API para emplear cualquier método de comunicación. Una de las diferencias entre API y Servicio Web es que la primera es capaz de definir con total exactitud el modo, el método o métodos que un programa usará para comunicarse con otros.

Por otro lado, Bush (2019) indica que, como característica común de la API, es que, lleve a cabo sus funciones desde el interior de un programa de software. Siendo ello así, las APIs son un conjunto de especificaciones y reglas que permiten que un programa se comunique con otro. Cuando la API debe enviar datos a través de una red, entra en escena el Servicio Web; en consecuencia, de ello se advierte que un Servicio Web es una API que se comunica mediante HTTP.

En ese orden de ideas, es necesario recalcar que, en la presente investigación, cuando se hace referencia a la API, estábamos hablando de las API web, que en su mayoría son las APIS modernas, las cuales exponen los datos y la funcionalidad de una aplicación a través de Internet.

1.3.4. Diseño de Web API

Según Gustavsson y Stenlund (2016) al consumir o desarrollar un API web, los principales desafíos están en el rendimiento, la facilidad de uso y la facilidad de mantenimiento. Para ello, su diseño debe basarse completamente en las necesidades del cliente; en consecuencia, se generan End Points para cada vista de usuario y no para cada recurso de una vista, reutilizando así componentes que pueden repetirse continuamente.

Además, un diseño optimizado de una web API debe ser escalable, es decir, cuando la cantidad de clientes aumente, los End Points construidos deben ser capaces de dar soporte a las solicitudes, y de facilitar la navegación de los usuarios.

En general, estos factores que representan desafíos para el diseño de web APIS no están aislados, sino que se relacionan entre sí, de esta forma, se tiene que, si una web API no tiene facilidad de uso ni de mantenimiento, entonces su rendimiento es bajo, por lo tanto, su escalabilidad es deficiente. (Gustavsson, 2016)

1.3.5. Tecnologías Para la implementación de un Servicio Web API

Para implementar servicios web existen diferentes tecnologías y/o arquitecturas, cada una establece un procedimiento y una forma de realizar las llamadas y la comunicación en general.

Entre ellas tenemos las siguientes:

1.3.5.1. SOAP

Simple Object Access Protocol (SOAP) se puede describir como una arquitectura para intercambiar mensajes en ambientes distribuidos, ampliamente implementado por servicios web y es una alternativa a REST y JSON. (Barry, 2013).

Asimismo, Papazoglou (2008) sostiene que, SOAP no define un transporte estándar para trasladar los datos, pero generalmente se usa el protocolo HTTP y el protocolo SMTP

Así es como esta arquitectura funciona con los dos métodos básicos: GET y POST. El primero se usa para recuperar datos del servidor, mientras que el segundo se usa para agregar o modificar estos. A pesar de su aparente simplicidad, SOAP implica alto consumo de recursos lo cual genera demoras y bajo rendimiento, por lo que, con la aparición de las aplicaciones móviles, REST empieza a ganar terreno por su ligereza y la flexibilidad de sus End Points.

1.3.5.2. REST

Transferencia de estado representacional (REST), actualmente es el estilo arquitectónico más común para diseñar web APIS.

El término "REST" se introdujo en 2000 en la tesis doctoral de Roy Fielding, uno de los principales autores de la especificación del Protocolo de transferencia de hipertexto (HTTP).

Fielding (2000) señala que REST es un estilo arquitectónico para sistemas hipermedia distribuidos y está basado en un conjunto de restricciones, tales como la escalabilidad de las interacciones, la generalidad de Las interfaces, la implementación independiente y los intermediarios de componentes para reducir la latencia de interacción, reforzar la seguridad y encapsular los sistemas heredados o involucrados.

Siendo ello así, en palabras de Haupt, Leymann, Scherer y Vukojevic-Haupt (2017) cualquier arquitectura que cumpla con estas restricciones pueden llamarse arquitectura REST (o RESTful).

En esta investigación definimos a REST como cualquier interfaz entre sistemas que utilizan protocolos de transporte como HTTP para obtener servicios y generar operaciones en todos los formatos posibles, como el Lenguaje de marcado extensible (XML) y Java Script Object Notation (JSON).

Por otro lado, Fielding (2000) siguió un enfoque basado en restricciones en el proceso de descubrimiento para REST. Por lo tanto, REST se rige por las siguientes restricciones:

Sin estado: Cuando se realiza las llamadas y se emite una respuesta entre el cliente y el servidor, está contenido el estado necesario para manejar la solicitud dentro de la solicitud en sí, sin que ello implique la necesidad que el servidor guarde información que comprometa el estado del cliente.

Interfaz uniforme: En el caso de utilizar el protocolo HTTP para API REST, la comunicación es iniciada por el cliente y consiste en una solicitud seguida de una respuesta o mensaje. Cada mensaje de solicitud junto con el identificador de recurso incluye acciones, métodos o verbos HTTP específicos (Por ejemplo, GET, PUT, POST y DELETE) que definen la operación a realizar en el recurso (Haupt et al., 2017).

Cliente-servidor: La interfaz uniforme que separa a los clientes de los servidores, permite a estos no preocuparse por los asuntos internos de los servidores. Asimismo, en el caso de estos últimos, tampoco les preocupa o muestran interés sobre la interfaz de usuario o el estado de usuario de los clientes. Por ejemplo, los clientes no están preocupados con detalles de almacenamiento de datos de cada servidor para que el rendimiento del código del cliente sea mejorado y los servidores pueden volverse más simples y escalables para no preocuparse por interfaz de usuario o estado del usuario. (Kumar, 2018)

Identificación de recursos a través de URI: Es el Identificador Universal de Recursos (URI) y representa el único identificador de cada recurso en este sistema REST. Los URI permiten acceder a la información para cambiarla o eliminarla, o, por ejemplo, para compartir su ubicación exacta con terceros.

Sistema de capas: Arquitectura jerárquica entre los componentes. Un cliente puede estar directamente conectado al servidor final o a un intermediario por el camino. Los servidores intermedios pueden mejorar la escalabilidad del sistema

al habilitar el equilibrio de carga y al proporcionar cachés compartidas (Fredrich, 2012)

Caché: Los clientes pueden almacenar en caché las respuestas. Pero las respuestas deben identificarse implícita o explícitamente como almacenables en caché o no, para evitar más solicitudes de clientes de reutilizar datos obsoletos o inapropiados en la respuesta.

Interacciones con estado a través de hipervínculos: Permite al usuario navegar por el conjunto de objetos a través de enlaces hipermedia. En el caso de una API REST, el concepto de Hypermedia explica la capacidad de una interfaz de desarrollo de aplicaciones para proporcionar el cliente y el usuario con los enlaces adecuados para ejecutar acciones específicas en los datos.

En palabras de (Haupat, 2017) para que sea genuino, las API REST deberían ser compatibles con Hypermedia como motor de Principio de estado de aplicación (HATEOAS).

En esta investigación, se sintetiza que REST es una arquitectura para implementación de web APIS, cuyos recursos son identificados mediante URIs direccionables, y típicamente interactúan con el uso de los verbos integrados de HTTP, específicamente, GET, PUT, DELETE, POST. Además, esta interacción se da según restricciones y según HATEOAS (Hypermedia As The Engine Of Application State) en la que el cliente interactúa con enlaces de hipermedia en lugar de una interfaz específica.

En las figuras 2, 3 y 4 se observan cómo se realiza una llamada con REST mediante 3 End Points, el primero para obtener el ID del usuario, el segundo para devuelve las publicaciones o Posts de los usuarios y el último devuelve la lista de seguidores del usuario.

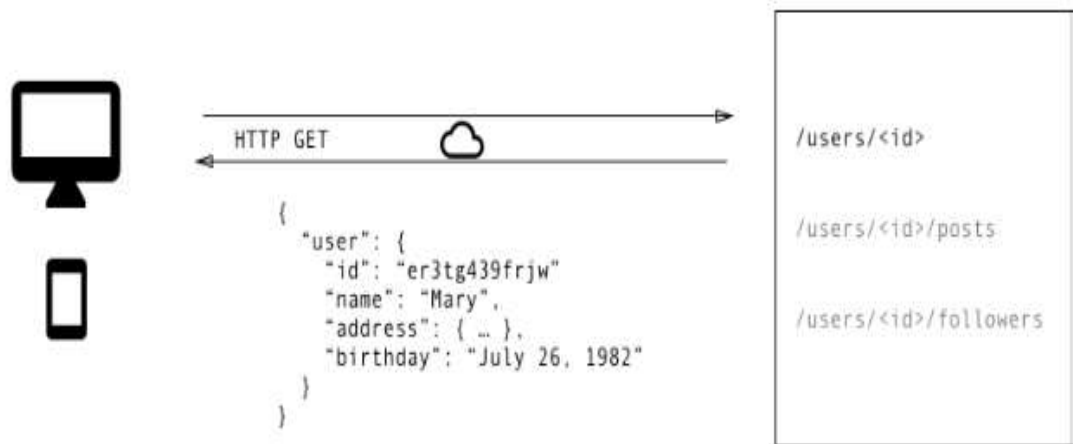


Figura 2. End Point get/users/<id> que devuelve el ID del usuario

Fuente: Recuperado de <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/>

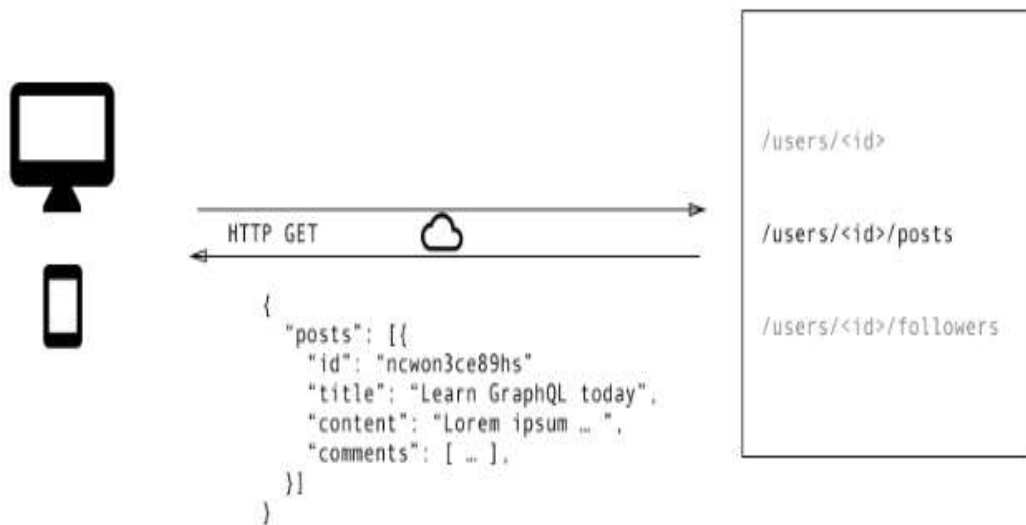


Figura 3. End Point get/users/<id>/posts que devuelve las publicaciones del usuario

Fuente: Recuperado de <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/>





Figura 4. End Point get/users/<id>/followers que lista los seguidores del usuario

Fuente: Recuperado de <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/>

Si bien REST fue una gran solución en su momento, ahora, hay algunos problemas bastante importantes en la arquitectura, toda vez que, las circunstancias actuales han dado lugar a nuevas necesidades, ahora las aplicaciones móviles y la big data; en consecuencia, se generan flujos de datos cambiantes y que demandan tecnologías rápidas y escalables.

Frente a este panorama, se ha considerado relevante estudiar la tecnología GraphQL que nace como una alternativa a REST para las aplicaciones y datos modernos.

1.3.5.3. GRAPHQL

GRAPHQL es el acrónimo de Graph Query Language, una tecnología desarrollada por Facebook en 2012 para cubrir la necesidad de obtener más flexibilidad y rapidez en el diseño e implementación de webs API, con requisitos cambiantes de los clientes y llamadas concurrentes crecientes (Burk, 2018)

En REST el cliente no tiene mucho control sobre la funcionalidad al solicitar o al recibir porque casi todo es proporcionado por el proveedor de servicios. En cambio, GraphQL fue desarrollado para hacer frente a la necesidad



de dar más responsabilidad a los clientes, así como mejorar la flexibilidad y la eficiencia. (Poniatowicz, 2019).

GraphQL proporciona una solución para las muchas limitaciones e ineficiencias experimentado por desarrolladores que interactúan con las API REST. Por ejemplo, GraphQL brinda al usuario la oportunidad de solicitar cualquier información específica que requiera en una sola petición o solicitud. Por el contrario, el usuario REST se ve obligado a realizar solicitudes adicionales para obtener la información específica necesaria. Como se puede apreciar en la figura 7 que presento a continuación:

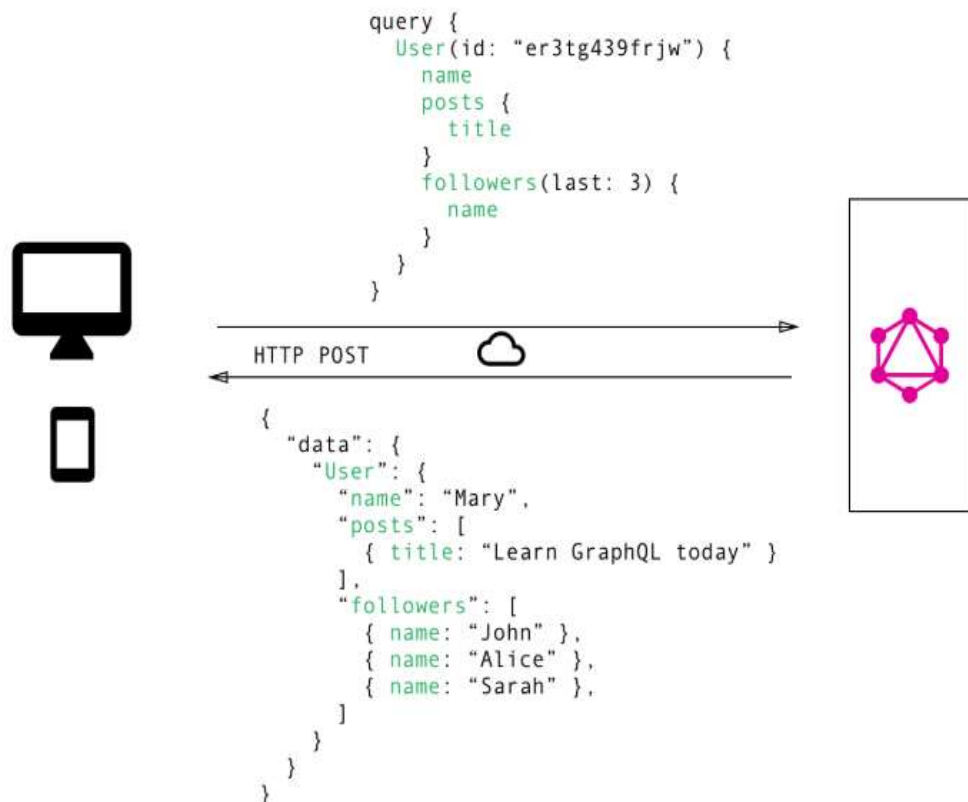


Figura 5. Petición GraphQL que devuelve el ID, los posts y los seguidores del usuario. Respuesta específica del servidor.

Fuente: Recuperado de <https://www.howtographql.com/basics/1-graphql-is-the-better-rest/>

Además, en palabras de Burk (2018) GraphQL es independiente de la base de datos y puede adaptarse bien en cualquier contexto donde Una API está involucrada.



Similar al servidor REST, un servidor GraphQL no está limitado al sistema operativo o lenguaje específico, y puede implementarse utilizando cualquier tecnología.

Es por ello, y, teniendo en cuenta lo establecido por (Eizinger, 2017) encuentro necesario describir las características más importantes de esta tecnología:

Características de GRAPHQL

Esquema: Es la columna vertebral de la tecnología GraphQL, la solicitud siempre se ejecuta de acuerdo con la estructura y el contexto del esquema. Se compone de 3 Types o puntos de entrada: query, mutation y suscripciones; asimismo se especifican las relaciones que se establecen entre estos types.

Query: Es el punto de entrada de uso común y compuesto por campos y tipos de datos. Eso se usa para realizar una consulta y obtener datos especificados al servidor.

Object Types: Estos son los componentes más básicos del esquema GraphQL que determina el tipo de objeto que el servicio GraphQL buscará, según lo especificado en el esquema. El lenguaje de esquema GraphQL admite los tipos de objetos escalares: String, Int, Float, Boolean e ID. Las Mutations y las Querys son tipos de objetos especiales, que actúan como un punto de entrada de cada consulta GraphQL.

Campos: GraphQL se trata de solicitar campos específicos en los objetos. Estos campos pueden representar tipos de datos escalares u objetos especiales. Cada campo se ejecuta de acuerdo con el Resolver correspondiente.

Argumentos: En GraphQL, cada campo y objeto anidado puede tener su propio conjunto de argumentos y esto ayuda a realizar diversas recuperaciones concurrentes de la API.

Resolvers: Estas son funciones utilizadas para obtener los datos de los campos. Cada función corresponde exactamente a un campo y resuelve las peticiones a los objetos, incluso a objetos anidados.

Mutations: Son parecidas a las Querys, pero se utilizan para cambiar datos en el servidor, y devuelven valores tal como lo hacen las consultas, por lo tanto, su importancia radica en que un cliente puede consultar datos basándose en el valor de retorno de una mutación.

Siendo ello así, en la figura 8, se puede observar un ejemplo de esquema que utiliza GraphQL, se puede visualizar los Types User y Post; así como un Query y una Mutation.

```

1  schema {
2    query: Query
3    mutation: Mutation
4  }
5
6  type User {
7    id: ID!
8    nickName: String
9    posts: [Post]!
10   followees: [User]!
11   followers: [User]!
12 }
13
14 type Post {
15   id: ID!
16   author: User
17   content: String
18   # The ISO representation of the date when the post was created.
19   createdAt: String
20   replies: [Post]!
21   replyTo: Post
22 }
23
24 type Query {
25   # Retrieve the timeline of a user identified by their nickname.
26   timeline(of: String): [Post]!
27   user(nickName: String): User
28   users: [User]!
29 }
30
31 type Mutation {
32   writePost(authorNick: String, content: String, replyTo: String = null): Post
33   newUser(nickName: String): User
34   followUser(me: String, other: String): User
35 }

```

Figura 6. Esquema del servicio de GraphQL, se pueden visualizar los Types y sus relaciones.

Fuente: Eizinger, T. (2017). API Design in Distributed Systems: A Comparison between GraphQL and REST.



1.3.6. Eficiencia de una Web API

Desde el punto de vista del usuario, una de las cosas más importantes cuando se consume una web API desde un cliente móvil o web, es que el proceso y la experiencia, desde la primera llamada y la navegación completa que ofrece esta web hasta la obtención final de los datos deseados, deben ser óptimos. (Gustavsson, 2016)

Esto involucra una exigencia de rendimiento en las tecnologías que dan soporte a la web API y a toda la arquitectura de la aplicación en cuestión.

Para conocer los estándares de evaluación de eficiencia de una web API o servicio web API, es que en este trabajo se consideró importante el fundamento teórico que proporciona la norma internacional ISO/IEC 25010.

1.3.7. ISO/IEC 25010

Esta norma hace referencia al modelo de calidad para el producto software y a la calidad en el uso del mismo, de esta manera, determina las características y factores de evaluación de las propiedades de un producto software especificado.

Según la Organización Internacional de Normalización (ISO), la calidad de un producto software queda establecida por la medida en que el producto cumple con los requisitos especificados en la documentación, estos requisitos se clasifican, categorizan y estandarizan para conformar el modelo de calidad del producto software. (ISO, 2019)

En este orden de ideas, el modelo establecido por esta norma, está conformado por ocho dimensiones o características, que a su vez presentan factores que hacen posible la evaluación de la calidad de un producto software, tal como se puede observar en la siguiente figura:



Figura 7. Características de calidad del producto software según ISO/IEC 25010

Fuente: Certificación de la Mantenibilidad del Producto Software: Un Caso Práctico - Scientific Figure on ResearchGate. Recuperado de: https://www.researchgate.net/figure/Modelo-de-calidad-del-producto-software-segun-la-ISO-IEC-25010_fig1_283699208 [accessed 1 Oct, 2019]

A continuación, se estudiaron estas dimensiones o características de calidad haciendo hincapié en la característica de Rendimiento que es la que se tomó en cuenta para lograr el objetivo de esta investigación.

1.3.8. Dimensiones de Eficiencia de ISO/IEC 25010

Según ISO (2019), las dimensiones para evaluar la calidad del producto software y la calidad en el uso funcional, son las siguientes:

- **Adecuación Funcional:** Capacidad del producto software para realizar todas las funciones establecidas en la documentación, satisfaciendo las necesidades y requerimientos funcionales de los usuarios
- **Rendimiento o Eficiencia de Desempeño:** Rendimiento del producto software en base a la cantidad de recursos utilizados o consumidos en un escenario o situación determinada. Presenta la siguiente clasificación en subdimensiones:



- **Comportamiento en el Tiempo:** Hace referencia a los valores de tiempos de respuesta y velocidad de procesamiento, además de las tasas de éxito en la entrega de mensajes (throughput).
- **Utilización de Recursos:** Representado por el tipo y la cantidad de recursos empleados, en el mismo intervalo de tiempo en que el producto software está desempeñando su función.
- **Capacidad:** Es la medida en que el producto software puede cumplir óptimamente con su función, y a partir de la cual el producto ofrece una función deficiente.
- **Compatibilidad:** Capacidad del producto para interconectarse o coexistir con otro sistema de software sin afectar sus funciones.
- **Usabilidad:** Medida en que el producto software es entendido por el usuario, además de ser intuitivo y atractivo en su uso
- **Fiabilidad:** Es la capacidad del producto de desempeñar adecuadamente sus funciones en el tiempo o cuando se alteren las condiciones normales.
- **Seguridad:** Medida en la que un sistema gestiona la protección de la información que contiene.
- **Mantenibilidad:** Medida de la facilidad con la que un producto software es modificado producto de una evolución en el tiempo.
- **Portabilidad:** Capacidad de un producto software para ser transferido o migrado a otras tecnologías o entornos diferentes de los actuales.

De estas dimensiones, la que se consideró para el desarrollo de esta investigación fue la de Rendimiento o eficiencia de desempeño, que ofrece subcaracterísticas de las cuales se pueden obtener métricas o factores de medición para llevar a cabo la evaluación de rendimiento de REST y GRAPHQL

1.4. Formulación del problema

Teniendo en cuenta los factores anteriormente referidos en líneas precedentes, es necesario establecer cuál es la tecnología más adecuada para la Implementación de una web API que sea capaz de mantener el flujo rápido de la información, así como la optimización de los recursos de hardware. Considerando las tecnologías

GraphQL y Rest para esta investigación, se planteó la siguiente interrogante: ¿Qué tecnología es la más eficiente para implementar servicios web consumidos por aplicaciones Android?

1.5. Justificación e importancia del estudio

Justificación Teórica

Este trabajo utilizó los lineamientos teóricos que establece la norma ISO/IEC 25010 para medir la eficiencia en cuanto al rendimiento, de las tecnologías GraphQL y Rest. Así mismo, esta investigación permite contrastar que las métricas, indicadores y el procedimiento seguido para dicha evaluación son aplicables en la realidad, específicamente en la implementación de servicios web consumidos por aplicaciones Android.

Justificación Práctica

Una de las razones por las que resultó importante determinar cuáles son las tecnologías más adecuadas para la implementación de un servicio web API, es que la elección de dichas tecnologías influirá directamente en los recursos humanos, económicos, de tiempo, etc. que se han de utilizar para hacer posible el flujo de datos y la disponibilidad de la información a través de internet, tal como lo mencionan (Braubach *et al.*, 2017). La eficiencia con que la API ejecute la comunicación de datos se traducirá finalmente en ahorro o pérdida de tiempo, dinero y otros recursos.

Teoría practica

Otro de los puntos relevantes por los cuales se realizó esta investigación, es el aumento de aplicaciones móviles en las tiendas virtuales. Según el reporte estadístico de datos del mercado tecnológico brindado por statista.com, se sabe que “En 2017, se descargaron 178.1 mil millones de aplicaciones desde dispositivos móviles, y se pronostica que para 2022 esta cifra será de 258.2 mil millones” (Statista, 2019). Por lo tanto, el incremento exponencial de aplicaciones móviles lleva consigo un incremento proporcional de APIS para realizar las transacciones y manejo de datos de estas aplicaciones, en consecuencia, al ser tan necesarias las APIS, se hace



también muy útil conocer el rendimiento que pueden ofrecer las tecnologías para implementar dichas APIS. Según un investigador de la universidad danesa Aarhus, en su estudio “Web API Capacity Study”, manifiesta que, si un servicio web es eficiente entonces servirá la información en el tiempo óptimo, sin errores, y el consumo del hardware estará en los niveles aceptables, esto repercute en el flujo dinámico de la información (Pasma, 2018, p. 4-8).

Justificación Metodológica

Esta investigación guarda una utilidad metodológica, porque a partir del procedimiento y metodología seguidos para la ejecución de las pruebas, y de los resultados objetivos y confiables que se obtuvieron, se contribuye en la reducción de la brecha del conocimiento en tecnologías de servicios web; asimismo, sirve como referente para el desarrollo de futuras investigaciones en este campo.

1.6. Hipótesis

La tecnología GraphQL es significativamente más eficiente que la tecnología Rest en la implementación de servicios web consumidos por aplicaciones Android.

1.7. Objetivos

1.7.1. Objetivo General

Evaluar la eficiencia de las tecnologías GraphQL y Rest en la implementación de servicios web consumidos desde aplicaciones móvil Android, bajo la norma ISO/IEC 25010.

1.7.2. Objetivos Específicos

- Seleccionar la arquitectura de comunicación que establecen las tecnologías GraphQL y Rest
- Seleccionar las dimensiones e indicadores de medición de calidad de software para las tecnologías GraphQL y REST, según la norma ISO/IEC 25010

- Seleccionar los procesos CRUD que se tomarán en cuenta para la medición de llamadas a la API para las tecnologías GraphQL y REST
- Implementar un API tanto en GraphQL y REST en base a la metodología SCRUM.
- Evaluar los resultados para las tecnologías GraphQL y Rest



CAPÍTULO II: MATERIAL Y MÉTODO

2.1. Tipo y Diseño de Investigación

2.1.1. Tipo de Investigación

Esta investigación es de tipo aplicada, debido a que la base teórica y de conocimientos que aquí se presentan se aplican directamente en tecnología real para alcanzar los objetivos planteados, contribuyendo positivamente en este campo de investigación. Además, este estudio se basó en la metodología cuantitativa, ya que las métricas e indicadores se expresan en valores numéricos o cantidades cuantificables.

2.1.2. Diseño de Investigación

La investigación corresponde a un diseño cuasi experimental, por lo mismo que, la población y muestra del estudio han sido previamente determinadas por el investigador.

2.2. Población y Muestra

2.2.1. Población

Helgason (2017) menciona que al día de hoy existen 3 tecnologías para implementar servicios web que pueden ser consumidos desde aplicaciones Android: SOAP, REST y GraphQL. Por lo tanto, la población quedó determinada por estas 3 tecnologías.

2.2.2. Muestra

La muestra para esta investigación ha sido establecida mediante el método no estadístico de muestra por conveniencia, para ello, se consideró la investigación hecha por Gustavson y Stenlund (2016) para determinar las tecnologías de servicios web más importantes en el desarrollo de APIS y más utilizadas en la actualidad, documentándose a Rest como la tecnología más utilizada y a GraphQL como la tecnología emergente y con mayor crecimiento en los últimos años. Para respaldar esta última afirmación, se documentó la estadística brindada por Google Trends donde se puede evidenciar el

interés y popularidad de GraphQL en los últimos 5 años, desde el 2014 hasta el 2019, tal como se puede observar en el siguiente gráfico.

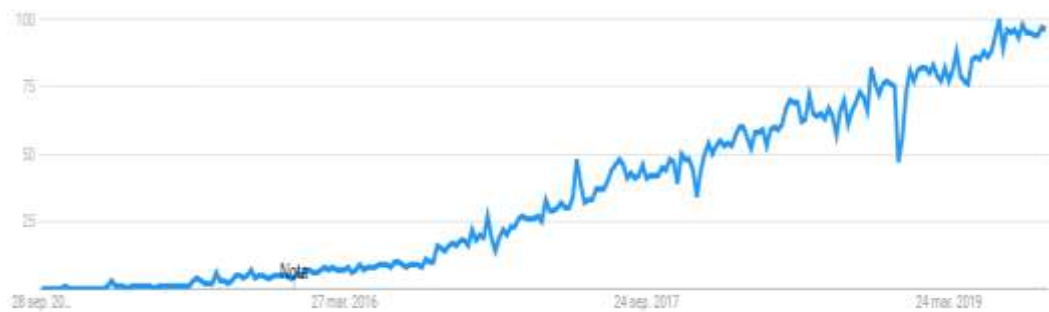


Figura 8. Popularidad de GraphQL en los últimos 5 años, del 2014 al 2019.

Fuente: Google Trends (2019)

Además, en palabras de Doerrfeld, Wood, Sandoval y Art (2016), actualmente Rest es el estándar aceptado para la implementación de servicios web; sin embargo, GraphQL está surgiendo rápidamente a tal punto que podría reemplazar a Rest.

Bajo estos argumentos, es que, en esta investigación se aplicó el método no estadístico de muestra por conveniencia. Por lo que, la muestra quedó determinada por las tecnologías Rest y GraphQL; ello en mérito a la relevancia que conllevan, y con la finalidad de obtener una mayor objetividad y utilidad en los resultados de la investigación.

2.3. Variables, operacionalización

2.3.1. Variable Independiente

Las tecnologías GraphQL y Rest

2.3.2. Variable Dependiente

Implementación de Servicios Web consumidos por aplicaciones Android



2.3.3 Operacionalización de las Variables

Tabla 1.
Cuadro de operacionalización de variables.

Variable Independiente	Dimensiones	Indicadores	Descripción	Unidad de medida	Técnicas
Tecnologías GraphQL y Rest	Utilización de los usuarios	Popularidad	Medida en la que la tecnología es conocida y/o utilizada	Porcentaje (%)	Observación Análisis
Variable Dependiente	Dimensiones	Indicadores	Descripción	Unidad de medida	Técnicas
Implementación de Servicios Web consumidos por aplicaciones Android	Rendimiento	Tiempo de respuesta	Tiempo total de la operación, desde que el dispositivo móvil realiza la llamada a la API hasta que la API devuelve la llamada al dispositivo Android.	Milisegundos (ms)	Observación
		Consumo de CPU	Fracción de CPU que utiliza el dispositivo móvil Android al realizar la llamada a la API	Porcentaje (%)	Análisis Experimentación
		Consumo de RAM	Fracción de memoria RAM que utiliza el dispositivo móvil Android al realizar la llamada a la API	Porcentaje (%)	

Fuente: Elaboración propia



2.4. Técnicas e instrumentos de recolección de datos, validez y confiabilidad

Las técnicas e instrumentos de recolección de datos que se utilizaron en esta investigación, se describen a continuación:

- Observación

Se aplicó mediante el registro visual y posterior documentación de la manera en la que operan las tecnologías GraphQL y Rest en la implementación de servicios web. Es decir, se observó las características y comportamientos reales de dichas tecnologías en el instante en el que están siendo consumidas desde aplicaciones Android.

- Análisis

Se utilizó este método para extraer la información de los diferentes documentos, tales como: libros, papers, revistas, etc. En los cuales, estuvieron contenidos los estándares, teorías, fórmulas y procedimientos, para conocer la arquitectura de comunicación con la que se realizan las llamadas, y se devuelven los datos, tanto en GraphQL como en Rest, además sirvió para conocer los lineamientos de eficiencia que brinda la norma ISO/IEC 25010.

- Experimentación

La investigación se ejecutó a partir de un problema real, empezando con la implementación de servicios web API tanto con la tecnología Rest como con la tecnología GraphQL, para luego consumir dicha API desde una aplicación móvil. Mediante este experimento se buscó medir la eficiencia de las tecnologías en cuestión, bajo los lineamientos del estándar ISO/IEC 25010, específicamente en la dimensión de rendimiento que esta norma establece. Para ello, se midieron indicadores como consumo de CPU, consumo de memoria RAM y tiempo de respuesta, desde el aplicativo Android. De esta manera se fundamentó la elaboración y verificación de la hipótesis planteada.



Por otra parte, los instrumentos para la recolección de datos que se utilizaron en esta investigación, se detallan a continuación:

Se emplearon formatos de recopilación de datos, elaborados a partir de una síntesis de tablas utilizados por otros investigadores, además de la información brindada por las páginas oficiales de las tecnologías investigadas y de sitios web de una importancia significativa como goodapi.co y howtographql.com (*los formatos se pueden observar en el anexo N°1*).

También se utilizaron guías de observación para el registro de los datos obtenidos del experimento o propuesta de investigación, estas guías fueron elaboradas en base a las métricas e indicadores de eficiencia en cuanto al rendimiento brindado por la especificación ISO/IEC 25010. (*Anexo N°2*).

2.5. Procedimiento de recolección y análisis de datos

a) Análisis Documental

El proceso efectuado para realizar el análisis documental de datos fue el siguiente:

a.1) Recopilar y analizar las especificaciones y características de las tecnologías GraphQL y Rest en la implementación de servicios web para ser consumidos por aplicaciones Android, tales como: cantidad de peticiones que procesan por unidad de tiempo, tipo de archivos que se devuelven como respuesta, constitución de las cadenas de verbos para las llamadas al servidor, etc. Esta información fue obtenida de los sitios web oficiales y sitios web que brindaron información significativa sobre las tecnologías investigadas, además, también se tomaron en cuenta las investigaciones previas documentadas.

a.2) Recopilar y analizar las técnicas y métodos para determinar la eficiencia de las tecnologías seleccionadas para este estudio, las cuales han sido desarrolladas y propuestas en investigaciones anteriormente realizadas.

a.3) Recopilar y analizar las métricas e indicadores para evaluación de calidad y eficiencia de tecnologías que sirvan para implementar servicios web API, las cuales se obtuvieron de la revisión de las normas internacionales de calidad de software ISO/IEC 25010.

b) Observación

Se observó y documentó el comportamiento de cada una de las tecnologías de servicios web, luego de que las API GraphQL y Rest fueron implementadas, mediante el siguiente proceso:

b.1) Registrar el porcentaje de CPU utilizado en el dispositivo Android, mientras la aplicación está consumiendo el servicio web tanto en GraphQL como en Rest.

b.2) Registrar el tiempo de respuesta a las peticiones para cada una de las tecnologías evaluadas.

b.3) Registrar el porcentaje de memoria RAM utilizada en el dispositivo Android, mientras la aplicación está consumiendo el servicio web tanto en GraphQL como en Rest.

2.6. Criterios Éticos

En este trabajo de investigación se tomaron en cuenta los siguientes criterios éticos:

- Criterio de la Confidencialidad

Los datos personales que se recopilaron para el cumplimiento de la propuesta, así como los que se recopilaron tras obtener los resultados de esta investigación, fueron obtenidos de manera lícita y con total profesionalidad para no causar ningún daño a las personas involucradas; tal como lo expone **La ley N° 29733: “Ley de protección de datos personales”, en su Título IV: “Obligaciones del titular y del encargado del banco de datos personales”**, la cual señala que, recopilar datos personales por medios ilícitos, fraudulentos o

desleales está penado por el Estado Peruano. Además, prohíbe utilizar los datos personales para finalidades distintas de aquellas que motivaron su recopilación, salvo que se efectúe previamente un procedimiento de disociación, con la finalidad de que no se pueda identificar al titular a través de los datos. Otro punto que se aclara es que cuando los datos hayan dejado de ser necesarios o pertinentes a la finalidad para la cual hubiesen sido recopilados, estos deberían ser suprimidos del banco de datos.

- **Criterio de Confirmabilidad**

Todos los resultados y afirmaciones respecto a los servidores web Linux que se hagan en esta investigación serán fiables y deberán ser confirmados por los investigadores, tal como lo expresa el *Código Deontológico del Colegio de Ingenieros del Perú en su Capítulo III “Faltas Contra la Ética Profesional y Sanciones”, en el Sub Capítulo II “De la Relación con El Público” en su Artículo 105: “Los ingenieros serán objetivos y veraces en sus informes y declaraciones, y expresarán opiniones en temas de ingeniería solamente cuando ellas se basen en un adecuado análisis y conocimiento de los hechos, detalles técnicos y científicos suficientes y veraces”.*

2.7. Criterios de Rigor Científico

- **Consistencia**

Los datos que se recolectaron para esta investigación son de carácter científico y formal. El análisis realizado a los datos está hecho con total profesionalidad aplicando habilidades, técnicas y conocimientos de la ingeniería y de la investigación para mantener la consistencia de los datos y para obtener información consistente y útil.

- Validez

Los datos obtenidos de las tecnologías GraphQL y Rest luego de su implementación, fueron correctamente evaluados y analizados para lograr un resultado válido que verifique la hipótesis.

- Fiabilidad

Las técnicas e instrumentos de medición utilizados para obtener los valores de los parámetros e indicadores medibles de las tecnologías objeto de estudio, siempre dieron como resultado valores aproximadamente iguales en las mismas circunstancias y situaciones.

- Objetividad

Los datos que se obtuvieron de la experimentación con las tecnologías GraphQL y Rest, así como los resultados del análisis de dichos datos están exentos de la influencia de la perspectiva de los investigadores.

CAPÍTULO III: RESULTADOS

3.1. Resultados en Tablas y Figuras

Tabla 2.
Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST -Listado de datos

TECNOLOGÍA EVALUADA	PROCESO	Consumo de RAM (Mb)
REST- LISTADO DE DATOS	Clientes	6
	Productos	5
	Ventas	9

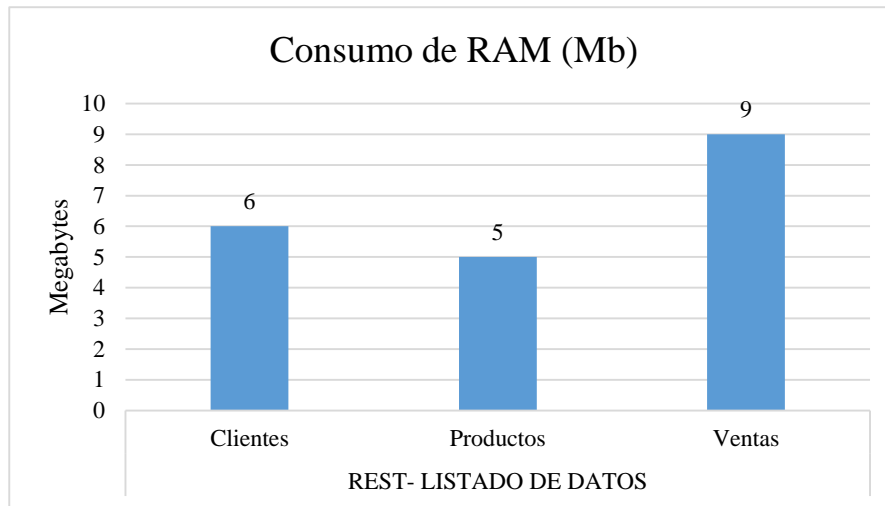


Figura 9: Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST - Listado de datos
Fuente: Elaboración propia.

Para la obtención de la figura anterior se empleó el siguiente código

$$A = (\text{sumResultadosPeticonesCliente} / \text{numPeticonesCliente}), \quad B$$

$$= (\text{sumResultadosPeticonesProducto} / \text{numPeticonesProducto}) \quad \text{y} \quad C$$

$$= (\text{sumResultadosPeticonesVentas} / \text{numPeticonesVentas}),$$

evidenciando la Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST, observándose, durante el listado de datos, que el proceso que más consume megabytes es el de ventas, seguido de clientes y de productos.



Tabla 3.
Llamadas desde la API REST – Listado de datos

CLIENTES		PRODUCTOS		VENTAS	
LLAMADAS	MEGABYTES	LLAMADAS	MEGABYTES	LLAMADAS	MEGABYTES
Llamada 1	6	Llamada 1	5	Llamada 1	11
Llamada 2	8	Llamada 2	4	Llamada 2	4
Llamada 3	5	Llamada 3	5	Llamada 3	5
Llamada 4	7	Llamada 4	3	Llamada 4	8
Llamada 5	8	Llamada 5	4	Llamada 5	13
Llamada 6	9	Llamada 6	3	Llamada 6	11
Llamada 7	2	Llamada 7	4	Llamada 7	15
Llamada 8	4	Llamada 8	6	Llamada 8	4
Llamada 9	5	Llamada 9	7	Llamada 9	11
Llamada 10	6	Llamada 10	9	Llamada 10	8
\bar{x}	6	\bar{x}	5	\bar{x}	9

En la tabla 3, podemos observar el número de llamadas a la API REST durante el listado de datos por clientes, productos y ventas; hallándose su respectiva media.

Tabla 4.
Visualización del promedio de Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Listado de datos

TECNOLOGÍA EVALUADA	Promedio Consumo de RAM (Mb)
REST- LISTADO DE DATOS	6

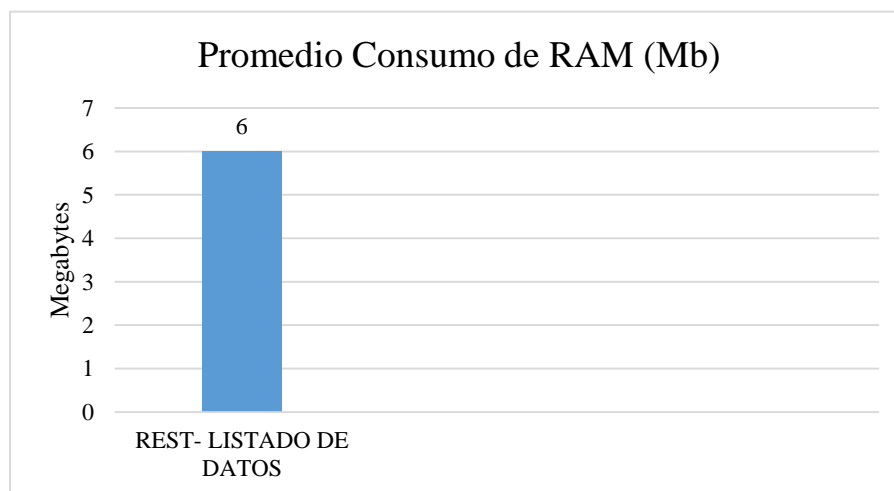


Figura 10: Visualización del promedio de Consumo de RAM (Mb) del proceso de llamadas desde la API REST - Listado de datos

Fuente: Elaboración propia.



En la figura anterior se evidencia el Consumo de RAM (Mb) del proceso de llamadas desde la API REST, observándose que, durante el listado de datos se consume en promedio 6 megabytes. Para calcularlo, se utilizó la fórmula de la media para datos no agrupados, tomando en cuenta los promedios hallados en la tabla 3:

$$\bar{x} = \frac{\sum xi}{n} = \frac{6 + 5 + 9}{3} = 6$$

Tabla 5.
Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Listado de datos

TECNOLOGÍA EVALUADA	PROCESO	Consumo de RAM (Mb)
GRAPHQL- LISTADO DE DATOS	Cientes	3
	Productos	4
	Ventas	4

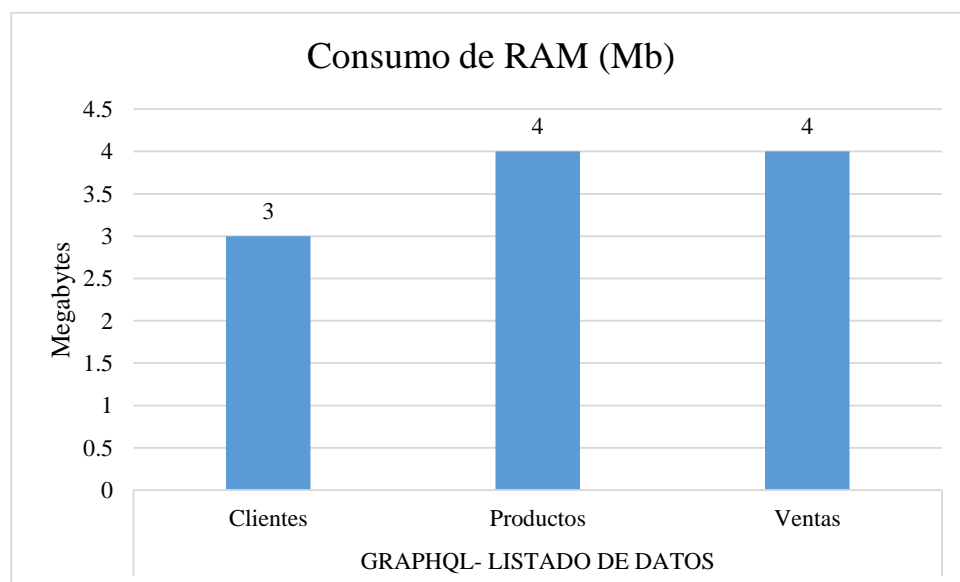


Figura 11: Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL - Listado de datos

Fuente: Elaboración propia.



Para la obtención de la figura anterior se empleó el siguiente código
 $A = (\text{sumResultadosPeticonesCliente} / \text{numPeticonesCliente})$, B
 $= (\text{sumResultadosPeticonesProducto} / \text{numPeticonesProducto})$ y C
 $= (\text{sumResultadosPeticonesVentas} / \text{numPeticonesVentas})$, evidenciando la Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST, observándose, durante el listado de datos, que el proceso que más consume megabytes es el de ventas y productos seguido de clientes.

Tabla 6.
 Llamadas desde la API GRAPHQL por consumo de RAM – Listado de datos

CLIENTES		PRODUCTOS		VENTAS	
LLAMADAS	MEGABYTE S	LLAMADAS	MEGABYTE S	LLAMADAS	MEGABYTE S
Llamada 1	2	Llamada 1	5	Llamada 1	3
Llamada 2	3	Llamada 2	7	Llamada 2	4
Llamada 3	2	Llamada 3	2	Llamada 3	4
Llamada 4	3	Llamada 4	3	Llamada 4	3
Llamada 5	4	Llamada 5	4	Llamada 5	4
Llamada 6	4	Llamada 6	4	Llamada 6	5
Llamada 7	4	Llamada 7	5	Llamada 7	4
Llamada 8	2	Llamada 8	5	Llamada 8	5
Llamada 9	3	Llamada 9	3	Llamada 9	6
Llamada 10	3	Llamada 10	2	Llamada 10	2
\bar{x}	3	\bar{x}	4	\bar{x}	4

En la tabla 6, podemos observar el número de llamadas a la API GRAPHQL durante el listado de datos, por clientes, productos y ventas; hallándose su respectiva media.



Tabla 7.
Visualización del promedio del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Listado de datos

TECNOLOGÍA EVALUADA	Promedio Consumo de RAM (Mb)
GRAPHQL- LISTADO DE DATOS	3

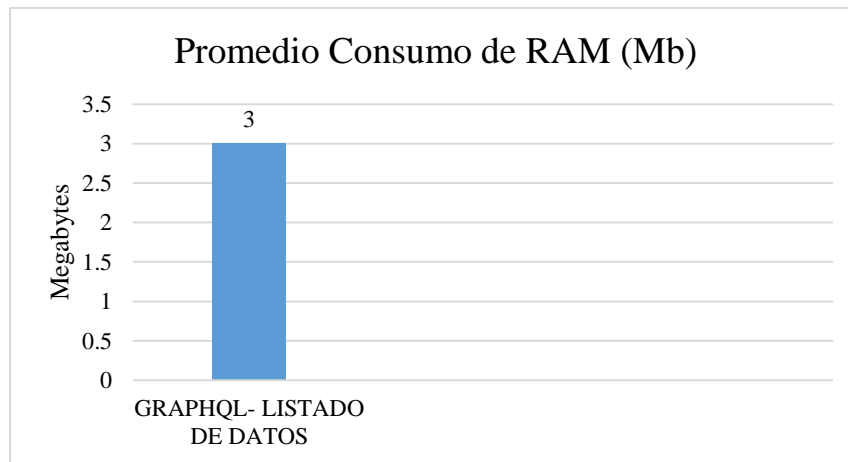


Figura 12: Visualización del promedio del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Listado de datos

Fuente: Elaboración propia.

En la figura anterior se evidencia el Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL, observándose que, durante el listado de datos se consume en promedio 3 megabytes. Para calcularlo, se utilizó la fórmula de la media para datos no agrupados, tomando en cuenta los promedios hallados en la tabla 6:

$$\bar{x} = \frac{\sum xi}{n} = \frac{3 + 4 + 4}{3} = 3$$



Tabla 8.
Comparación de promedios de Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL –Listado de datos

TECNOLOGÍA EVALUADA	Promedio Consumo de RAM (Mb)
REST- LISTADO DE DATOS	6
GRAPHQL- LISTADO DE DATOS	3

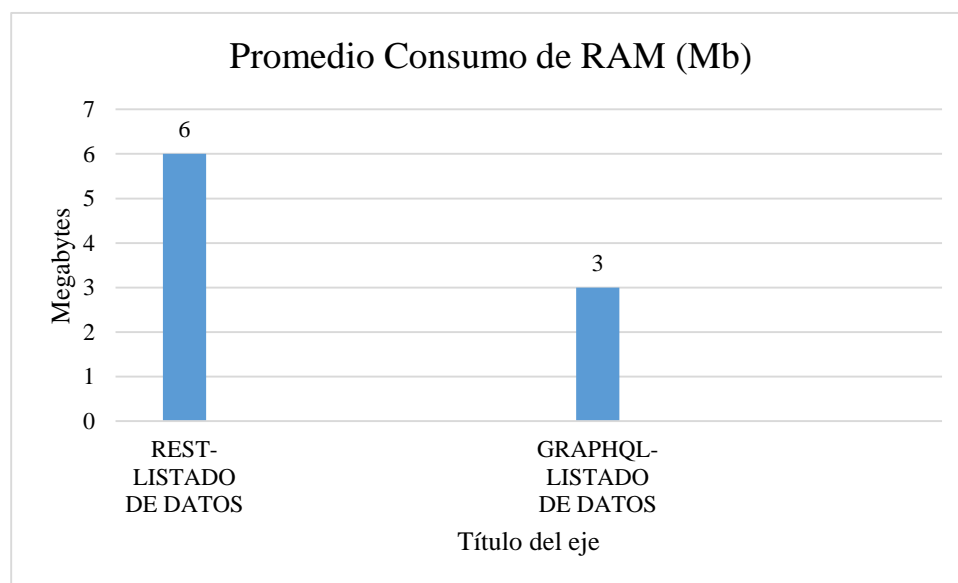


Figura 13: *Comparación de promedios de Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL –Listado de datos*

Fuente: *Elaboración propia.*

En la figura anterior se evidencia la comparación del Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL, concluyéndose que, durante el listado de datos, la tecnología que más consume Megabytes es la tecnología REST y la de menor consumo es la tecnología GRAPHQL. (Anexo N° 3, rendimiento: comparativo)



Tabla 9.
Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Registro de datos

TECNOLOGÍA EVALUADA	PROCESO	Consumo de RAM (Mb)
REST- REGISTRO DE DATOS	Clientes	9
	Productos	5
	Ventas	8

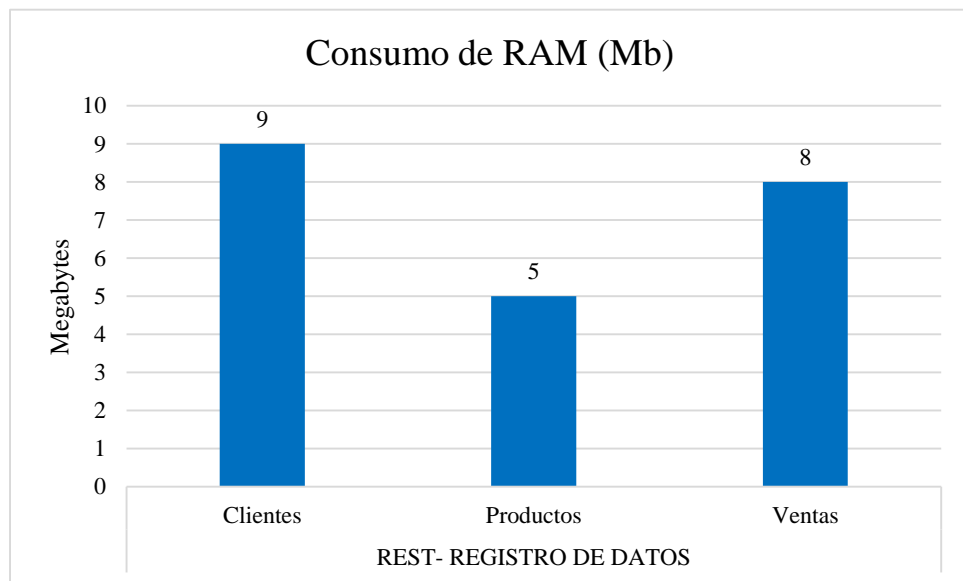


Figura 14: *Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST – Registro de datos*

Fuente: *Elaboración propia.*

Para la obtención de la figura anterior se empleó el siguiente código
 $A = (\text{sumResultadosPeticionesCliente} / \text{numPeticionesCliente})$, B
 $= (\text{sumResultadosPeticionesProducto} / \text{numPeticionesProducto})$ y C
 $= (\text{sumResultadosPeticionesVentas} / \text{numPeticionesVentas})$, se evidencia la Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST, observándose que, durante el registro de datos, el proceso que más consume megabytes es el de clientes, seguido de ventas y de productos.



Tabla 10.
Llamadas desde la API REST por consumo de RAM – Registro de datos

CLIENTES		PRODUCTOS		VENTAS	
LLAMADAS	MEGABYTES	LLAMADAS	MEGABYTES	LLAMADAS	MEGABYTES
Llamada 1	10	Llamada 1	2	Llamada 1	7
Llamada 2	8	Llamada 2	5	Llamada 2	9
Llamada 3	12	Llamada 3	2	Llamada 3	9
Llamada 4	16	Llamada 4	4	Llamada 4	3
Llamada 5	7	Llamada 5	2	Llamada 5	5
Llamada 6	8	Llamada 6	4	Llamada 6	9
Llamada 7	10	Llamada 7	5	Llamada 7	10
Llamada 8	11	Llamada 8	7	Llamada 8	13
Llamada 9	3	Llamada 9	9	Llamada 9	6
Llamada 10	5	Llamada 10	10	Llamada 10	9
\bar{x}	9	\bar{x}	5	\bar{x}	8

En la tabla 10, podemos observar el número de llamadas a la API REST durante el registro de datos, por clientes, productos y ventas; hallándose su respectiva media.

Tabla 11.
Visualización del promedio de Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Registro de datos

TECNOLOGÍA EVALUADA	Promedio Consumo de RAM (Mb)
REST- REGISTRO DE DATOS	7

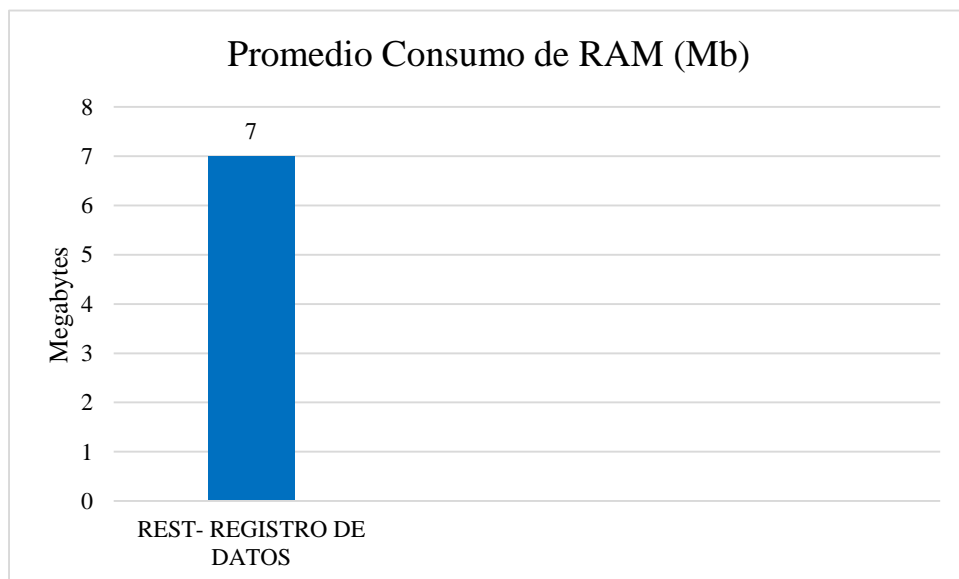


Figura 15: Visualización del promedio de Consumo de RAM (Mb) del proceso de llamadas desde la API REST –Registro de datos

Fuente: Elaboración propia.



En la figura anterior se evidencia la Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST, observándose que, durante el registro de datos se consume en promedio 7 megabytes. Para calcularlo, se utilizó la fórmula de la media para datos no agrupados, tomando en cuenta los promedios hallados en la tabla 10:

$$\bar{x} = \frac{\sum xi}{n} = \frac{9 + 5 + 8}{3} = 7$$

Tabla 12.
Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Registro de datos

TECNOLOGÍA EVALUADA	PROCESO	Consumo de RAM (Mb)
GRAPHQL-REGISTRO DE DATOS	Clientes	3
	Productos	4
	Ventas	8

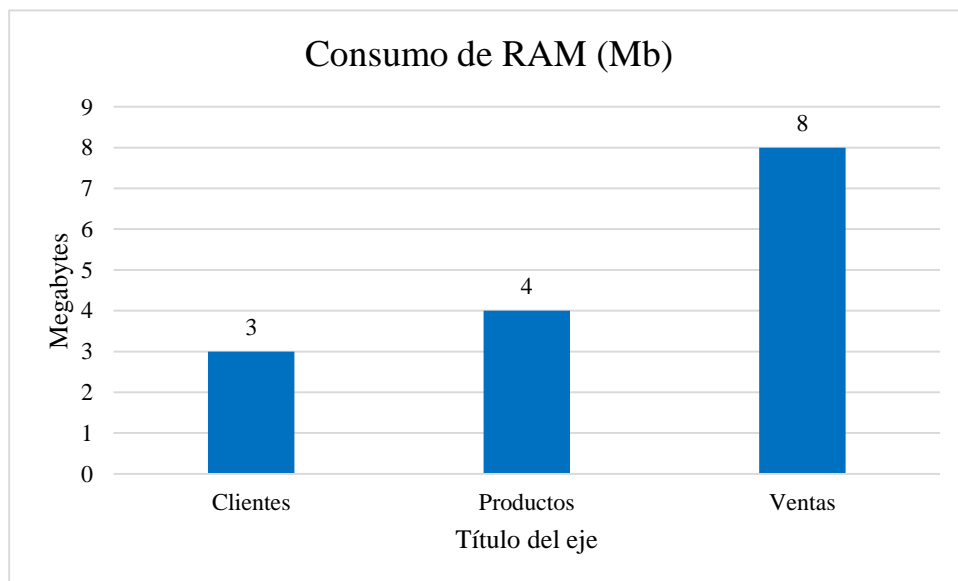


Figura 16: Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Registro de datos

Fuente: Elaboración propia.



Para la obtención de la figura anterior se empleó el siguiente código

$$A = (\text{sumResultadosPeticonesCliente} / \text{numPeticonesCliente}), \quad B$$

$$= (\text{sumResultadosPeticonesProducto} / \text{numPeticonesProducto}) \quad \text{y} \quad C$$

$$= (\text{sumResultadosPeticonesVentas} / \text{numPeticonesVentas}),$$

se evidencia la Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL, observándose que, durante el registro de datos, el proceso que más consume megabytes es el de ventas y productos seguido de clientes.

Tabla 13.
Llamadas desde la API GRAPHQL por consumo de RAM – Registro de datos

CLIENTES		PRODUCTOS		VENTAS	
LLAMADAS	MEGABYTES	LLAMADAS	MEGABYTES	LLAMADAS	MEGABYTES
Llamada 1	4	Llamada 1	5	Llamada 1	5
Llamada 2	3	Llamada 2	6	Llamada 2	11
Llamada 3	1	Llamada 3	2	Llamada 3	6
Llamada 4	2	Llamada 4	3	Llamada 4	9
Llamada 5	4	Llamada 5	2	Llamada 5	13
Llamada 6	3	Llamada 6	5	Llamada 6	7
Llamada 7	5	Llamada 7	3	Llamada 7	10
Llamada 8	1	Llamada 8	5	Llamada 8	4
Llamada 9	3	Llamada 9	5	Llamada 9	10
Llamada 10	4	Llamada 10	4	Llamada 10	5
\bar{x}	3	\bar{x}	4	\bar{x}	8

En la tabla 13, podemos observar el número de llamadas a la API GRAPHQL durante el registro de datos por clientes, productos y ventas; hallándose su respectiva media.



Tabla 14.

Visualización del promedio del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Registro de datos

TECNOLOGÍA EVALUADA	Promedio Consumo de RAM (Mb)
GRAPHQL- REGISTRO DE DATOS	5

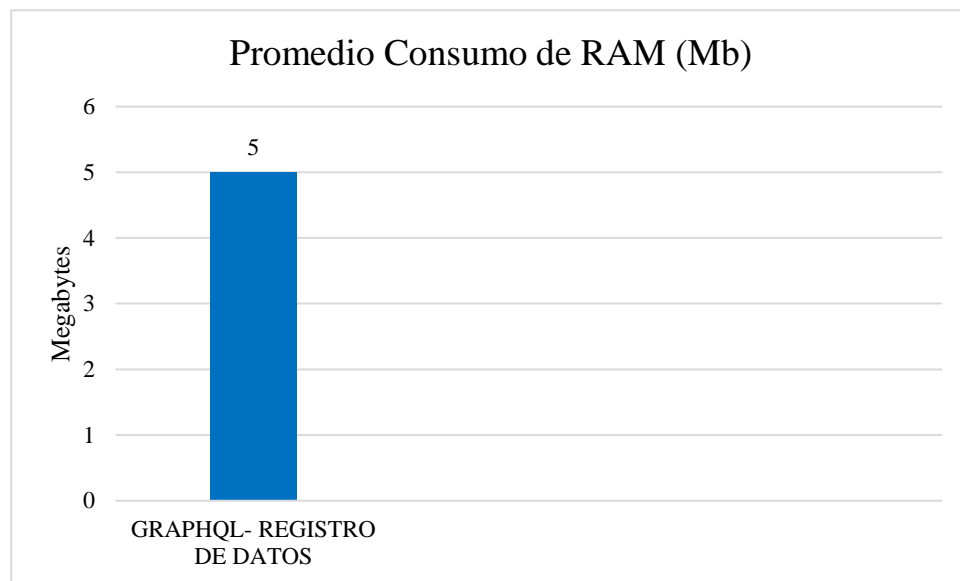


Figura 17: Visualización del promedio del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL –Registro de datos

Fuente: Elaboración propia.

En la figura anterior se evidencia la Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API GRAPHQL, observándose que, durante el registro de datos se consume en promedio 5 megabytes. Para calcularlo, se utilizó la fórmula de la media para datos no agrupados, tomando en cuenta los promedios hallados en la tabla 13:

$$\bar{x} = \frac{\sum xi}{n} = \frac{3 + 4 + 8}{3} = 5$$



Tabla 15.

Comparación de los promedios de Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL –Registro de datos

TECNOLOGÍA EVALUADA	Promedio Consumo de RAM (Mb)
REST- REGISTRO DE DATOS	7
GRAPHQL- REGISTRO DE DATOS	5

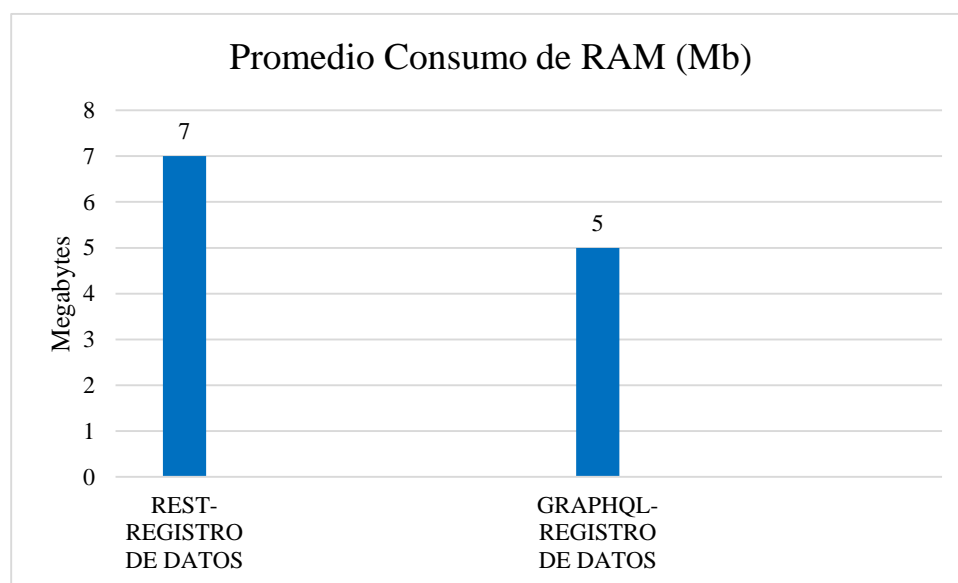


Figura 18: Comparación de los promedios de Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL –Registro de datos

Fuente: Elaboración propia.

En la figura anterior se evidencia la comparación del Consumo de RAM (Mb) del proceso de llamadas desde la API REST y GRAPHQL, concluyéndose que, durante el registro de datos, la tecnología que más consume megabytes es el REST; mientras que la tecnología GRAPHQL es la que consume menos megabytes. (Anexo N° 3, rendimiento: comparativo)



Tabla 16.

Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Listado de datos

TECNOLOGÍA EVALUADA	PROCESO	Tiempo de Respuesta (ms)
REST- LISTADO DE DATOS	Clientes	163
	Productos	186
	Ventas	187

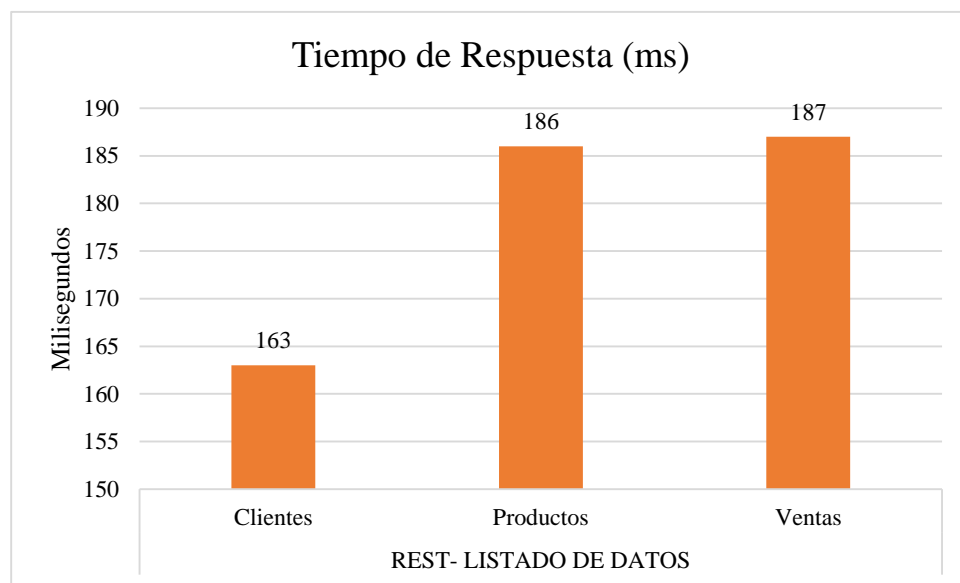


Figura 19: Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Listado de datos

Fuente: Elaboración propia.

Para la obtención de la figura anterior se empleó el siguiente código

$$A = (\text{sumResultadosPeticionesCliente} / \text{numPeticionesCliente}), \quad B$$

$$= (\text{sumResultadosPeticionesProducto} / \text{numPeticionesProducto}) \quad \text{y} \quad C$$

$$= (\text{sumResultadosPeticionesVentas} / \text{numPeticionesVentas}),$$

se evidencia la visualización del Tiempo de Respuesta (ms) del proceso de llamadas desde la API REST, observándose que, durante el listado de datos, el proceso que mayor tiempo de respuesta requiere es el de ventas, seguido de productos y de clientes.



Tabla 17.
Llamadas desde la API REST por consumo de RAM – Listado de datos

CLIENTES		PRODUCTOS		VENTAS	
LLAMADAS	MILISEGUNDOS	LLAMADAS	MILISEGUNDOS	LLAMADAS	MILISEGUNDOS
Llamada 1	195	Llamada 1	195	Llamada 1	243
Llamada 2	145	Llamada 2	189	Llamada 2	140
Llamada 3	159	Llamada 3	277	Llamada 3	149
Llamada 4	187	Llamada 4	156	Llamada 4	209
Llamada 5	137	Llamada 5	171	Llamada 5	191
Llamada 6	136	Llamada 6	140	Llamada 6	180
Llamada 7	180	Llamada 7	189	Llamada 7	168
Llamada 8	133	Llamada 8	167	Llamada 8	222
Llamada 9	178	Llamada 9	216	Llamada 9	200
Llamada 10	180	Llamada 10	160	Llamada 10	168
\bar{x}	163	\bar{x}	186	\bar{x}	187

En la tabla 17, podemos observar el número de llamadas a la API REST durante el listado de datos por clientes, productos y ventas; hallándose su respectiva media.

Tabla 18.
Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Listado de datos

TECNOLOGÍA EVALUADA	Promedio Tiempo de Respuesta (ms)
REST- LISTADO DE DATOS	178

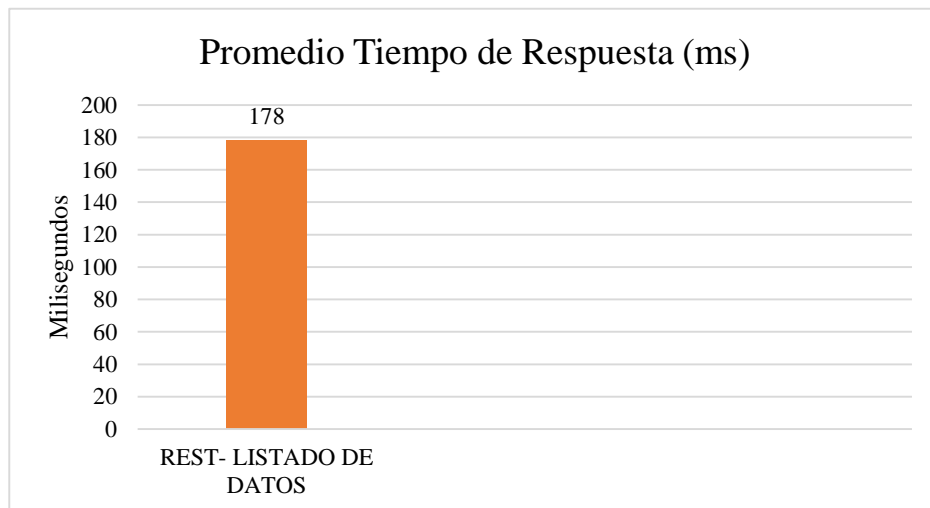


Figura 20: Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Listado de datos

Fuente: Elaboración propia.



En la figura anterior se evidencia la visualización del Tiempo de Respuesta (ms) del proceso de llamadas desde la API REST, observándose que, durante el listado de datos, el promedio de tiempo de respuesta (ms) es de 178 milisegundos. Para calcularlo, se utilizó la fórmula de la media para datos no agrupados, tomando en cuenta los promedios hallados en la tabla 17:

$$\bar{x} = \frac{\sum xi}{n} = \frac{163 + 186 + 187}{3} = 178$$

Tabla 19.

Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Listado de datos

TECNOLOGÍA EVALUADA	PROCESO	Tiempo de Respuesta (ms)
GRAPHQL- LISTADO DE DATOS	Cientes	133
	Productos	108
	Ventas	147

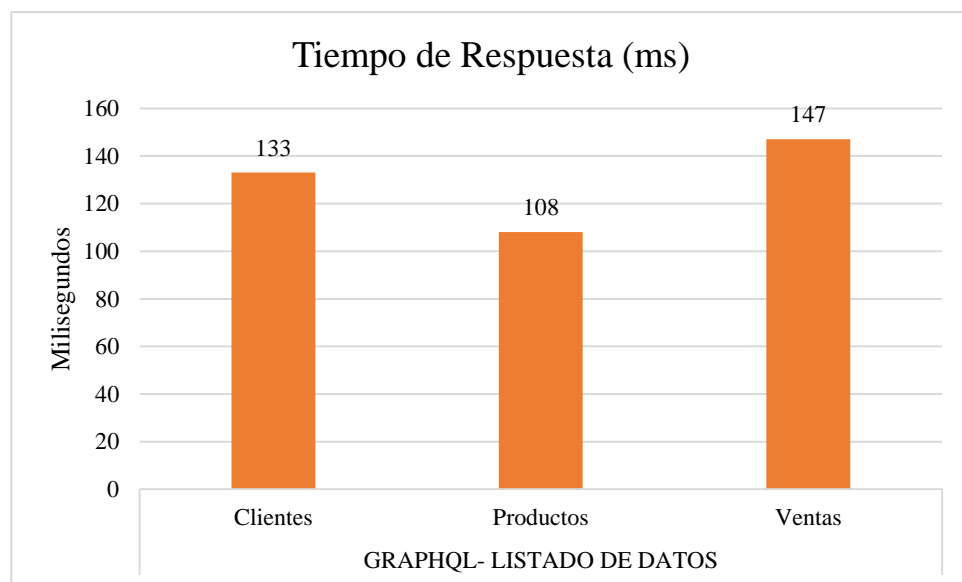


Figura 21: Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Listado de datos

Fuente: Elaboración propia.



Para la obtención de la figura anterior se empleó el siguiente código $A=(\text{sumResultadosPeticonesCliente}/\text{numPeticonesCliente})$, $B=(\text{sumResultadosPeticonesProducto}/\text{numPeticonesProducto})$ y $C=(\text{sumResultadosPeticonesVentas}/\text{numPeticonesVentas})$, se visualiza el Tiempo de Respuesta (ms) del proceso de llamadas desde la API GRAPHQL, observándose que, durante el listado de datos, el proceso que mayor tiempo de respuesta requiere es el de ventas, seguido de clientes y productos.

Tabla 20.
Llamadas desde la API GRAPHQL por tiempo de respuesta– Listado de datos

CLIENTES		PRODUCTOS		VENTAS	
LLAMADAS	MILISEGUNDOS	LLAMADAS	MILISEGUNDOS	LLAMADAS	MILISEGUNDOS
Llamada 1	168	Llamada 1	108	Llamada 1	158
Llamada 2	134	Llamada 2	112	Llamada 2	148
Llamada 3	139	Llamada 3	107	Llamada 3	145
Llamada 4	140	Llamada 4	103	Llamada 4	147
Llamada 5	134	Llamada 5	106	Llamada 5	143
Llamada 6	133	Llamada 6	109	Llamada 6	148
Llamada 7	116	Llamada 7	107	Llamada 7	144
Llamada 8	122	Llamada 8	109	Llamada 8	147
Llamada 9	124	Llamada 9	110	Llamada 9	146
Llamada 10	120	Llamada 10	109	Llamada 10	144
\bar{x}	133	\bar{x}	108	\bar{x}	187

En la tabla 20, podemos observar el número de llamadas a la API GRAPHQL durante el listado de datos por clientes, productos y ventas; hallándose su respectiva media.



Tabla 21.

Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Listado de datos

TECNOLOGÍA EVALUADA	Promedio Tiempo de Respuesta (ms)
GRAPHQL- LISTADO DE DATOS	129

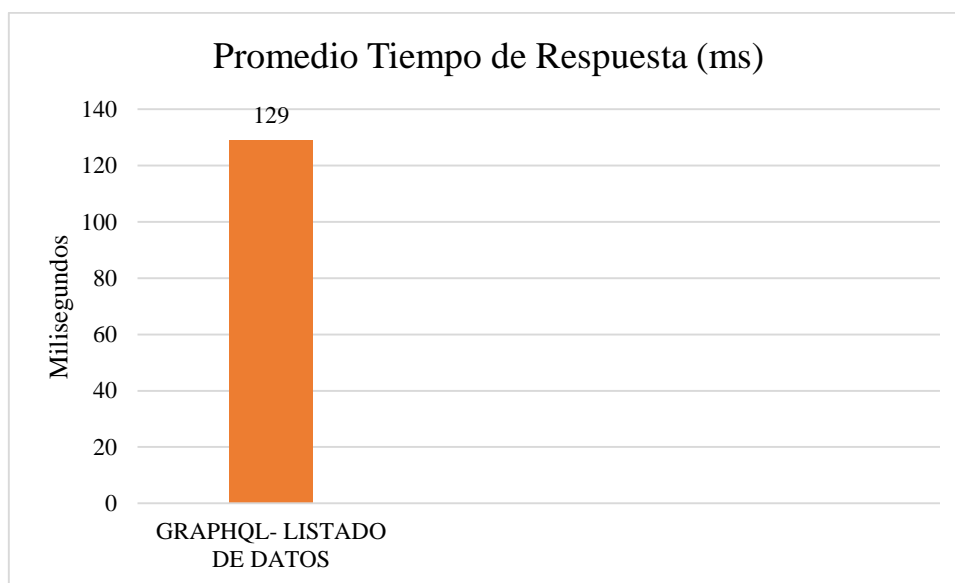


Figura 22: Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Listado de datos

Fuente: Elaboración propia.

En la figura anterior se evidencia la visualización del Tiempo de Respuesta (ms) del proceso de llamadas desde la API GRAPHQL, observándose que, durante el listado de datos, el promedio de tiempo de respuesta (ms) es de 129 milisegundos. Para calcularlo, se utilizó la fórmula de la media para datos no agrupados, tomando en cuenta los promedios hallados en la tabla 20:

$$\bar{x} = \frac{\sum xi}{n} = \frac{133 + 108 + 187}{3} = 129$$



Tabla 22.
 Comparación de promedios del tiempo de respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL –Listado de datos

-TECNOLOGÍA EVALUADA	Promedio Tiempo de Respuesta (ms)
REST- LISTADO DE DATOS	178
GRAPHQL- LISTADO DE DATOS	129

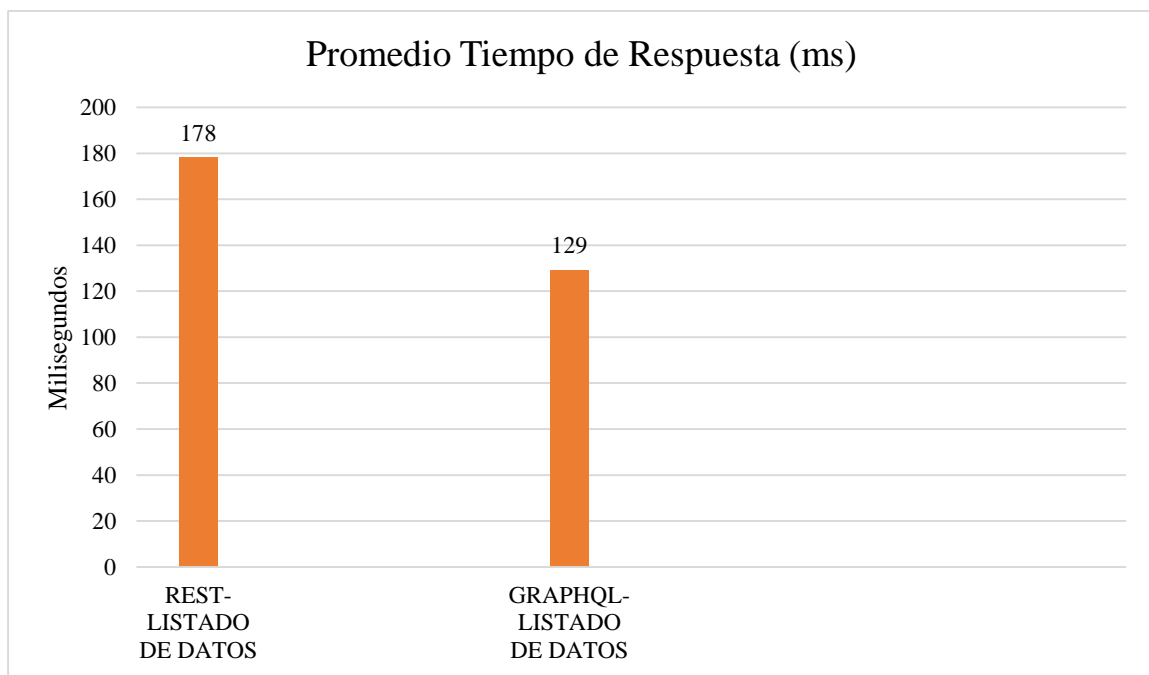


Figura 23: Comparación de promedios del tiempo de respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL –Listado de datos

Fuente: Elaboración propia.

En la figura anterior se evidencia la comparación del Tiempo de Respuesta (ms) del proceso de llamadas desde la API GRAPHQL, concluyéndose que, durante el listado de datos, el promedio de tiempo de respuesta (ms) mayor es el de la tecnología REST,



con 178 ms; mientras que la tecnología GRAPHQL, con 129 ms es la que tiene un promedio de tiempo de respuesta menor. (Anexo N° 3, rendimiento: comparativo)

Tabla 23.

Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API REST – Registro de datos

TECNOLOGÍA EVALUADA	PROCESO	Tiempo de Respuesta (ms)
REST- REGISTRO DE DATOS	Clientes	229
	Productos	179
	Ventas	203

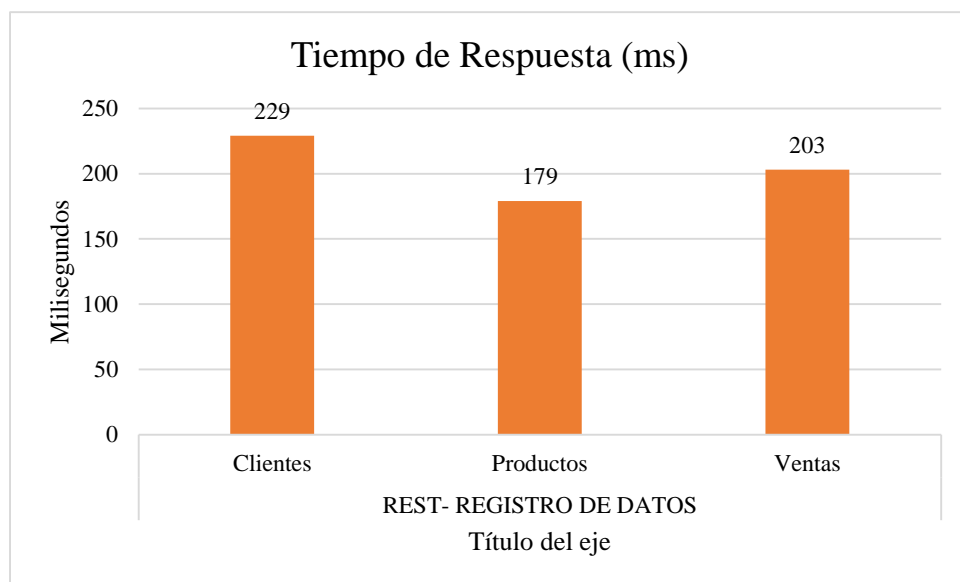


Figura 24: Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Registro de datos

Fuente: Elaboración propia.

Para la obtención de la figura anterior se empleó el siguiente código

$$A = (\text{sumResultadosPeticionesCliente} / \text{numPeticionesCliente}), \quad B$$

$$= (\text{sumResultadosPeticionesProducto} / \text{numPeticionesProducto}) \quad \text{y} \quad C$$

$$= (\text{sumResultadosPeticionesVentas} / \text{numPeticionesVentas}),$$

se evidencia la Visualización del Consumo de RAM (Mb) del proceso de llamadas desde la API REST, observándose



que, durante el registro de datos, el proceso que mayor tiempo de respuesta requiere es el de clientes seguido de ventas y productos.

Tabla 24.
Llamadas desde la API REST por tiempo de respuesta – Registro de datos

CLIENTES		PRODUCTOS		VENTAS	
LLAMADAS	MILISEGUNDOS	LLAMADAS	MILISEGUNDOS	LLAMADAS	MILISEGUNDOS
Llamada 1	215	Llamada 1	187	Llamada 1	219
Llamada 2	198	Llamada 2	160	Llamada 2	156
Llamada 3	239	Llamada 3	318	Llamada 3	187
Llamada 4	198	Llamada 4	136	Llamada 4	201
Llamada 5	205	Llamada 5	157	Llamada 5	229
Llamada 6	310	Llamada 6	147	Llamada 6	189
Llamada 7	191	Llamada 7	135	Llamada 7	229
Llamada 8	258	Llamada 8	156	Llamada 8	157
Llamada 9	240	Llamada 9	235	Llamada 9	162
Llamada 10	236	Llamada 10	159	Llamada 10	301
\bar{x}	229	\bar{x}	179	\bar{x}	203

En la tabla 24, podemos observar el número de llamadas a la API REST durante el registro de datos por clientes, productos y ventas; hallándose su respectiva media.

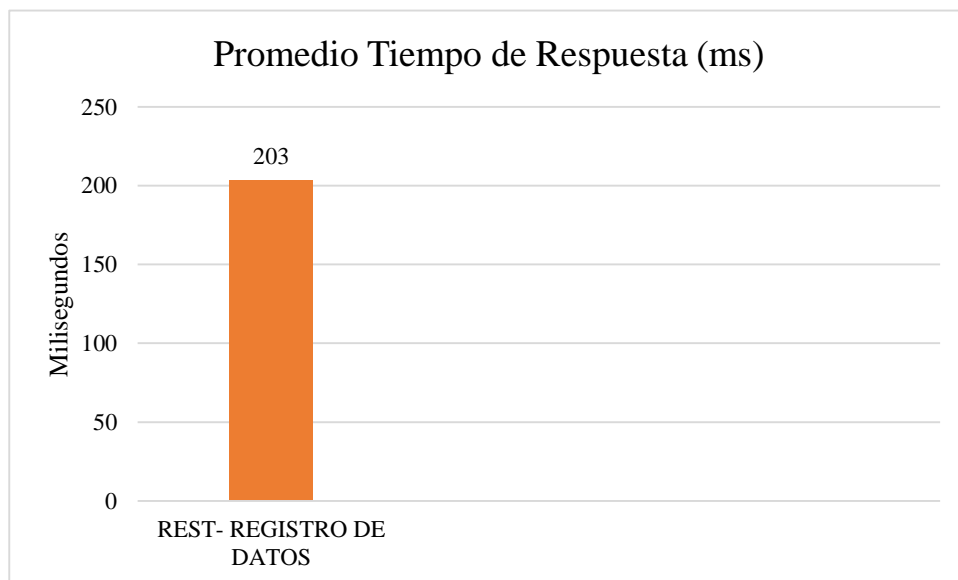


Figura 20: Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST –Registro de datos

Fuente: Elaboración propia.



En la figura anterior se evidencia la visualización del Tiempo de Respuesta (ms) del proceso de llamadas desde la API GRAPHQL, observándose que, durante el listado de datos, el promedio de tiempo de respuesta (ms) es de 203 milisegundos. Para calcularlo, se utilizó la fórmula de la media para datos no agrupados, tomando en cuenta los promedios hallados en la tabla 20:

$$\bar{x} = \frac{\sum xi}{n} = \frac{229 + 179 + 203}{3} = 203$$

Tabla 25.
Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Registro de datos

TECNOLOGÍA EVALUADA	PROCESO	Tiempo de Respuesta (ms)
GRAPHQL- REGISTRO DE DATOS	Clientes	133
	Productos	113
	Ventas	72

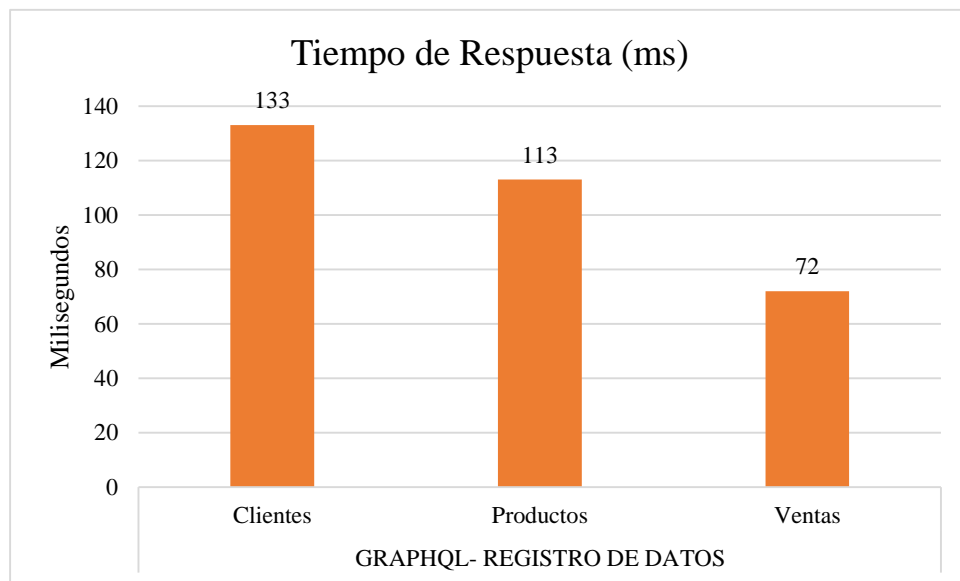


Figura 25: Visualización del tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –Registro de datos

Fuente: Elaboración propia.



Para la obtención de la figura anterior se empleó el siguiente código

$$A = (\text{sumResultadosPeticionesCliente} / \text{numPeticionesCliente}), \quad B$$

$$= (\text{sumResultadosPeticionesProducto} / \text{numPeticionesProducto}) \quad \text{y} \quad C$$

$$= (\text{sumResultadosPeticionesVentas} / \text{numPeticionesVentas}),$$

se evidencia la visualización del Tiempo de Respuesta (ms) del proceso de llamadas desde la API GRAPHQL, observándose que, durante el registro de datos, el proceso que mayor tiempo de respuesta requiere es el de clientes, seguido de productos y de ventas.

Tabla 26.
Llamadas desde la API GRAPHQL por tiempo de respuesta – Registro de datos

CLIENTES		PRODUCTOS		VENTAS	
LLAMADAS	MILISEGUNDOS	LLAMADAS	MILISEGUNDOS	LLAMADAS	MILISEGUNDOS
Llamada 1	133	Llamada 1	186	Llamada 1	54
Llamada 2	116	Llamada 2	117	Llamada 2	53
Llamada 3	174	Llamada 3	105	Llamada 3	51
Llamada 4	176	Llamada 4	109	Llamada 4	52
Llamada 5	128	Llamada 5	95	Llamada 5	108
Llamada 6	127	Llamada 6	97	Llamada 6	83
Llamada 7	125	Llamada 7	117	Llamada 7	54
Llamada 8	104	Llamada 8	94	Llamada 8	51
Llamada 9	123	Llamada 9	106	Llamada 9	56
Llamada 10	124	Llamada 10	104	Llamada 10	158
\bar{x}	133	\bar{x}	113	\bar{x}	72

En la tabla 26, podemos observar el número de llamadas a la API GRAPHQL durante el registro de datos por clientes, productos y ventas; hallándose su respectiva media.



Tabla 27.

Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –registro de datos

TECNOLOGÍA EVALUADA	Promedio Tiempo de Respuesta (ms)
GRAPHQL- REGISTRO DE DATOS	106

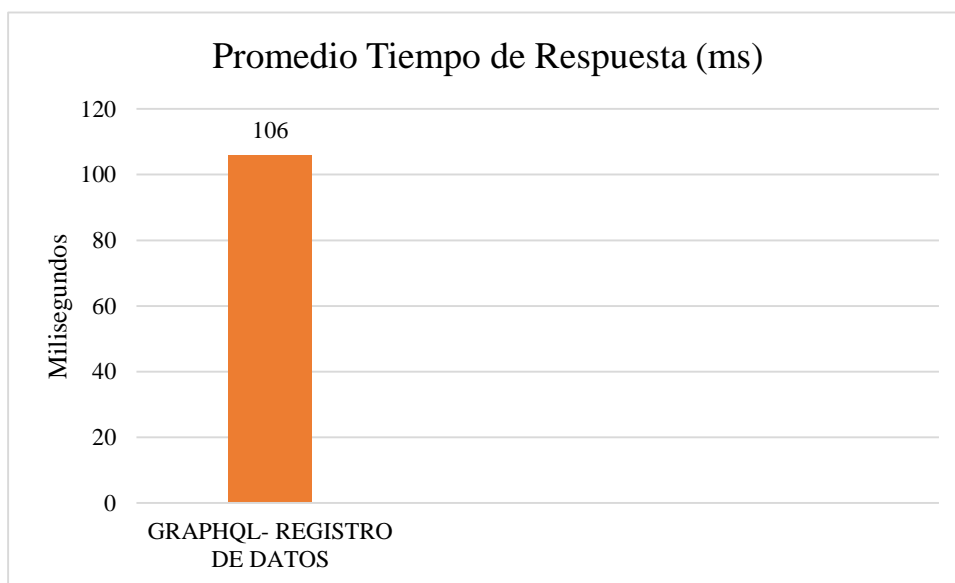


Figura 26: Visualización del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API GRAPHQL –registro de datos

Fuente: Elaboración propia.

En la figura anterior se evidencia la visualización del Tiempo de Respuesta (ms) del proceso de llamadas desde la API GRAPHQL, observándose que, durante el registro de datos, el promedio de tiempo de respuesta (ms) es de 106 milisegundos. Para calcularlo, se utilizó la fórmula de la media para datos no agrupados, tomando en cuenta los promedios hallados en la tabla 20:

$$\bar{x} = \frac{\sum xi}{n} = \frac{133 + 113 + 72}{3} = 106$$



Tabla 28.

Comparación del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL –Registro de datos

TECNOLOGÍA EVALUADA	Promedio Tiempo de Respuesta (ms)
REST- REGISTRO DE DATOS	203
GRAPHQL- REGISTRO DE DATOS	106

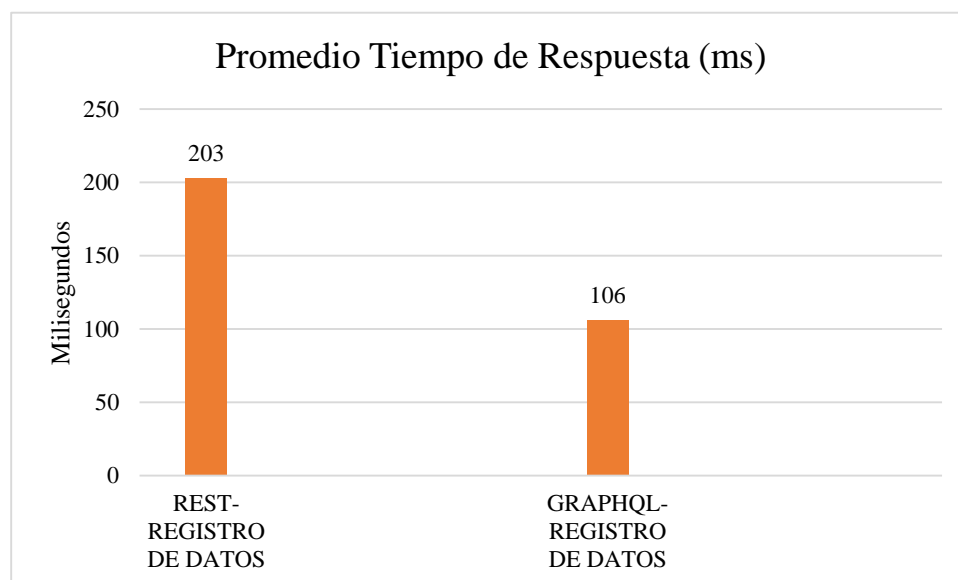


Figura 27: Comparación del promedio de tiempo de respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL –Registro de datos

Fuente: Elaboración propia.

En la figura anterior se evidencia la comparación del Tiempo de Respuesta (ms) del proceso de llamadas desde la API REST y GRAPHQL, concluyéndose que, durante el registro de datos, el promedio de tiempo de respuesta (ms) mayor es el de la tecnología REST, con 203 ms; mientras que la tecnología GRAPHQL, con 106 ms es la que tiene un promedio de tiempo de respuesta menor. (Anexo N° 3, rendimiento: comparativo)



3.2. Discusión de resultados

Se coincide con Brito, Bombach & Valente (2019), que en sus implementaciones basadas en GraphQL demostraron que el tamaño de los archivos devueltos por la API disminuyó en con respecto a Rest, con nuestra investigación comprobamos y afianzamos esta aseveración.

Nuestros resultados avalan la teoría de Helgason (2017) el cual expuso la comparación de rendimiento entre las tecnologías de servicios web GraphQL y Rest bajo la perspectiva del manejo de bases de datos relacionales. Aseverando que en las consultas que involucran combinación de campos y tablas, GraphQL es más eficiente respecto a Rest. Esto se pudo comprobar haciendo llamadas a procesos como Productos, clientes y ventas en nuestra API.

Gustavsson y Stenlund (2016); así como Cechak (2017) y Eeda (2018) en sus investigaciones implementaron una API, para determinar con precisión objetiva los escenarios en los cuales es más eficiente utilizar GraphQL o Rest. De las llamadas a la API se obtuvieron los tiempos de respuesta para cada uno de los prototipos implementados. Avalamos este enunciado ya que al desarrollar nuestra API pudimos comprobar que el esfuerzo y el tiempo de las llamadas a la API requieren de menos esfuerzo y tiempo que desarrollarla con Rest.

Se coincide con Pasma (2018), en su investigación usó una API para evaluar la capacidad de un sistema que ejecuta peticiones repetitivas. Para ello, se tomaron en cuenta indicadores como la tasa de uso del CPU, la tecnología GraphQL, es más eficiente en comparación a la tecnología Rest. Con nuestra investigación comprobamos que, en cuanto al consumo de RAM Pasma está en lo correcto.

En cuanto a las investigaciones de Eizinger (2017) y RitzilÄ (2017) se puede aseverar que sus teorías no pueden ser comprobadas del todo dado que, no demostraron cuantificablemente cuál es la tecnología adecuada o correcta y sus trabajos se basaron en la recopilación teórica de factores o puntos críticos de comparación para ambas tecnologías.

Nuestra investigación sugiere el uso de GraphQL así que, estamos en parte de acuerdo con Atencio y Mamani (2017) que realizaron un trabajo de investigación en el que los resultados obtenidos están la optimización del tiempo de los trámites al utilizar la API REST, mejorando significativamente el flujo de la información. Pero hemos podido demostrar que GraphQL es más eficiente en cuando al tiempo y consumo de RAM.

Se podría tomar en cuenta la investigación de Burgos (2017) en su investigación donde evaluaron las arquitecturas REST y SOAP, obteniendo que REST ofrece un rendimiento significativamente mayor al que ofrece SOAP en la implementación de servicios web. Por lo que podemos afirmar que GraphQL es más eficiente en cuando al tiempo y consumo de RAM que REST y SOAP.

3.3. Aporte práctico

3.3.1. Seleccionar la arquitectura de comunicación que establecen las tecnologías GraphQL y Rest

Los pilares fundamentales eran SOAP y REST para la programación de aplicaciones, sin embargo, en 2015, Facebook lanzó a GraphQL como alternativa para una mejor forma de ejecutar procesos en la construcción de aplicaciones. (PROMPTBYTES, 2019)

Como resultado final se muestra un cuadro comparativo entre estas dos tecnologías REST VS GraphQL que son importantes para el desarrollar una API.

Tabla 29.

Cuadro comparativo entre arquitecturas de REST y GRAPHQL.

REST	GRAPHQL
BENEFICIOS	BENEFICIOS
<ul style="list-style-type: none"> - Alto rendimiento, especialmente a través de HTTP2. - Probado por décadas. - Funciona con cualquier representación. - Centrado en la concordancia. - estado de la aplicación dirigida por el servidor. - Desacoplamiento completo de cliente y servidor que permite la evolución independiente. - Neutral. - Abarca múltiples API o recursos dedicados para las necesidades específicas del cliente. - No se necesita documentación de referencia. 	<ul style="list-style-type: none"> - Fácil de comenzar y producir y consumir. - Mucha mano. - Contrato impulsado por la naturaleza - Fácil de mantener consistente y gobernar. - introspección incorporada. - Está más cerca de SOAP (un protocolo) que es bueno para las empresas. - Neutral. - Una API para todos, incluidas las necesidades específicas del cliente - No se requiere documentación de referencia.



REST	GraphQL
DEFICIENCIAS	DEFICIENCIAS
<ul style="list-style-type: none"> - Viene con una gran barrera de entrada en el entrenamiento y el aprendizaje, que la mayoría de nosotros nunca superamos. - Requiere que los clientes jueguen juntos. - Sin marco ni guía de herramientas - Herramientas pobres o nulas para clientes. - Requiere disciplina en todos los lados - Tienes que ser un experto y aun así no lo harías bien. - Si lo haces mal, terminarías en un mundo de problemas. En ese caso, sería mejor usar GraphQL. - Requiere disciplina por todos lados. - Desafío para mantener la coherencia y cualquier gobierno. 	<ul style="list-style-type: none"> - Descuida los problemas del sistema distribuido. - Optimización de consultas. - Bikeshedding (negociación de contenido, errores de red, almacenamiento en caché, etc.) - representación JSON solamente. - Servidor y clientes acoplados en el momento de programación del cliente, el servidor no controla el estado de la aplicación. - Muy pocos vendedores en el ecosistema y el principal están fingiendo que es el dueño del programa. - Desecha todo lo que HTTP estuvo descubriendo durante los últimos 17 años.

Fuente: Adaptado de (PROMPTBYTES, 2019)

Proceso de análisis de tecnologías

Según (PROMPTBYTES, 2019) luego de comparar diversas tecnologías, sugieren el uso de las más recientes porque mejora notablemente el desarrollo de aplicaciones móviles. Tomando en cuenta lo siguiente:

- **SOAP:** Funciona con las dos funciones básicas: GET y POST. GET se usa para recuperar datos del servidor, mientras que POST se usa para agregar o modificar datos.
- **REST:** Cambia el estado de la fuente correspondiente haciendo una solicitud al URI (Identificador Uniforme de Recursos).



- **GraphQL:** aprovecha las solicitudes de dos tipos: consultas que recuperan datos del servidor y mutaciones que cambian los datos.

Esto sumado a nuestros antecedentes se resume en el siguiente cuadro:

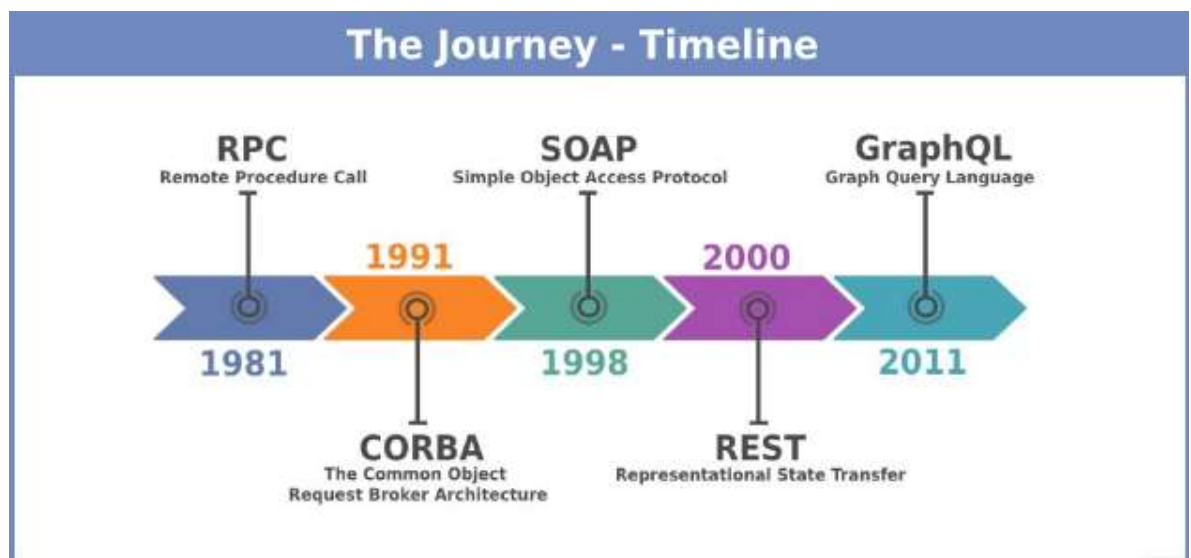


Figura 28: Línea de tiempo de tecnologías para el desarrollo de aplicaciones móviles.

Fuente: Adaptado de (PROMPTBYTES, 2019)

3.3.2. Seleccionar las dimensiones e indicadores de medición de calidad de software para las tecnologías GraphQL y REST, según la norma ISO/IEC 25010.

Para medir la calidad de nuestro aplicativo móvil se tomó en cuenta la siguiente evaluación acoplada al desarrollo y características propias de la API.

Basándose en nuestros antecedentes y según lo indicado en el cuadro de comparaciones de REST Y GRAPHQL por (PROMPTBYTES, 2019) tomaremos las siguientes dimensiones e indicadores de la norma ISO 25010. Para entender mejor la Valoración de Likert se detalla en la siguiente tabla.



Tabla 30.
Cuadro de valoración de tecnologías basado en la ISO 25010.

DIMENSIÓN	INDICADOR	ESCALA	GRAPHQL	REST
Rendimiento	Tiempos de respuesta y procesamiento	Likert de 5 niveles	5	5
	Recursos utilizados cuando el software funciona	Likert de 5 niveles	5	5
Usabilidad	Usuarios que entienden el software.	Likert de 5 niveles	2	2
	Operatividad y control el software	Likert de 5 niveles	2	2
	Interacción amigable y agradable.	Likert de 5 niveles	2	2
Portabilidad	Compatibilidad con varias versiones del sistema operativo y equipos.	Likert de 5 niveles	2	1
	Facilidad para instalar o desinstalar.	Likert de 5 niveles	2	3

Fuente: Elaboración propia.

*Nota. 1. No es importante, 2. Poco importante, 3. Ni importante ni poco importante, 4. Importante, 5. Muy importante

Asimismo, se tomó en cuenta la investigación de Burgos (2017) donde luego de evaluar las arquitecturas de REST Y GRAPHQL sugiere que el rendimiento es considerado parte fundamental al momento de evaluar una API.

Por lo mencionado anteriormente y luego de un análisis basado en la ISO 25010 sobre las dimensiones e indicadores, se ha determinado que el rendimiento y los



indicadores de tiempos y recursos, son los más importantes para ser medidos y evaluados en la comparación de las tecnologías REST Y GRAPHQL.

3.3.3. Seleccionar los procesos CRUD que se tomarán en cuenta para la medición de llamadas a la API para las tecnologías GraphQL y REST

Según (Codecademy, 2019), cuando estamos creando API, queremos que nuestros modelos proporcionen cuatro tipos básicos de funcionalidad. El modelo debe poder crear, leer, actualizar y eliminar recursos. El paradigma CRUD es común en la construcción de aplicaciones web, ya que proporciona un marco memorable para recordar a los desarrolladores cómo construir modelos completos y utilizables. Tomando en cuenta lo siguiente:

- Registrar: Para crear recursos, generalmente utilizamos el método HTTP POST. POST crea un nuevo recurso del tipo de recurso especificado.
- Listar: Para leer recursos, utilizamos el método GET. Leer un recurso nunca debería cambiar ninguna información, solo debería recuperarla.
- Actualizar: PUT es el método HTTP utilizado para la operación CRUD, Actualización.
- Eliminar: La operación CRUD Delete corresponde al método HTTP DELETE. Se utiliza para eliminar un recurso del sistema.

En este punto se escogerá el tipo de proceso que se va a evaluar tomando en cuenta la cantidad de campos que se van a seleccionar, para esto se ha dado una valoración de 0 a 100 y se podrá visualizar en la tabla 32.

Tabla 31.

Cuadro de valoración y puntaje para el grado de importancia de procesos CRUD involucrados en la medición del rendimiento.

VALORACIÓN	NOMENCLATURA	RANGO
1	No es importante	0-20
2	Poco importante	21-40
3	Ni importante ni poco importante	41-60
4	Importante	61-80
5	Muy importante	81-100

Fuente: Elaboración propia.

Tabla 32.

Cuadro de valoración y puntaje para el grado de importancia de procesos CRUD involucrados en la medición del rendimiento.

Nombre en la API (Para tema de selección de proceso)	Proceso CRUD	REST Y GRAPHQL	
		Puntaje	Valoración
REGISTRO DE DATOS	REGISTRAR	90	5
	ELIMINAR	70	4
	ACTUALIZAR	70	4
LISTADO DE DATOS	LISTAR	81	5

Fuente: Elaboración propia

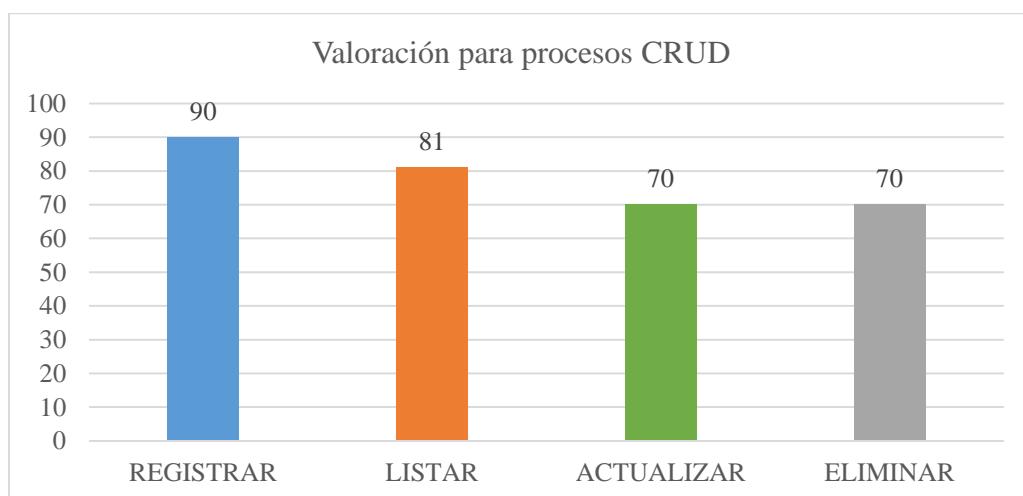


Figura 29: Valoración para procesos CRUD.

Fuente: Elaboración propia



Para el desarrollo de nuestra Api, en el caso de GRAPHQL, el registro, actualizar y eliminar son considerados un mismo tipo de petición, mientras que para REST se ha usado el método POST que básicamente cumple la misma función para registro, actualizar y eliminar.

Para temas de conveniencia y espacio se usará el término “Registro” como la medición total de estos tres procesos del CRUD, el registrar, actualizar y eliminar; es así que cuando hablemos de la cantidad de tiempo y consumo de RAM en cuanto a registro habremos medido estos tres procesos en conjunto.

Asimismo, podemos notar que los procesos CRUD más importantes en las APIS para la tienda de ropa, son registrar y listar, es por ello, que se les ha dado una mayor puntuación; mientras que, actualizar y eliminar tienen menos puntuación debido a su poco uso dentro de las API.

3.3.4. Implementar un API tanto en GraphQL y REST en base a la metodología SCRUM.

Según, (Castillo Asencio, 2016), SCRUM es un método de gestión de proyectos, el cual puede adaptarse a cualquier tipo de proyecto y no únicamente a los de desarrollo de software.

Aplicando Scrum al desarrollo de software, se basa en el modelo de las metodologías ágiles, incrementales, basadas en iteraciones y revisiones continuas.

El objetivo principal es elevar al máximo la productividad del equipo de desarrollo. Reduce al máximo las actividades no orientadas a producir software, tomando como parte fundamental la producción de resultados funcionales en períodos cortos de tiempo.

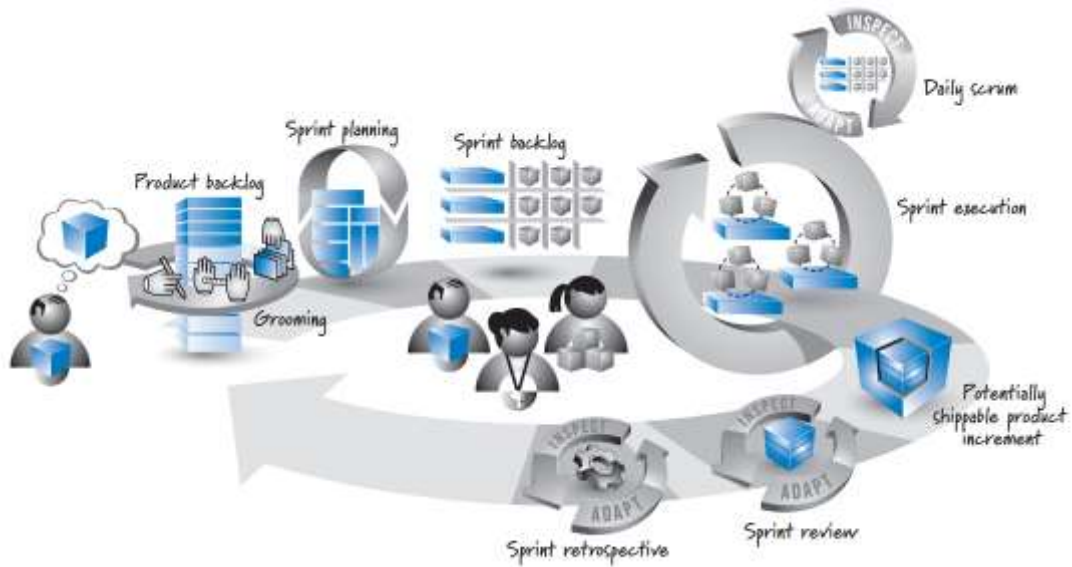
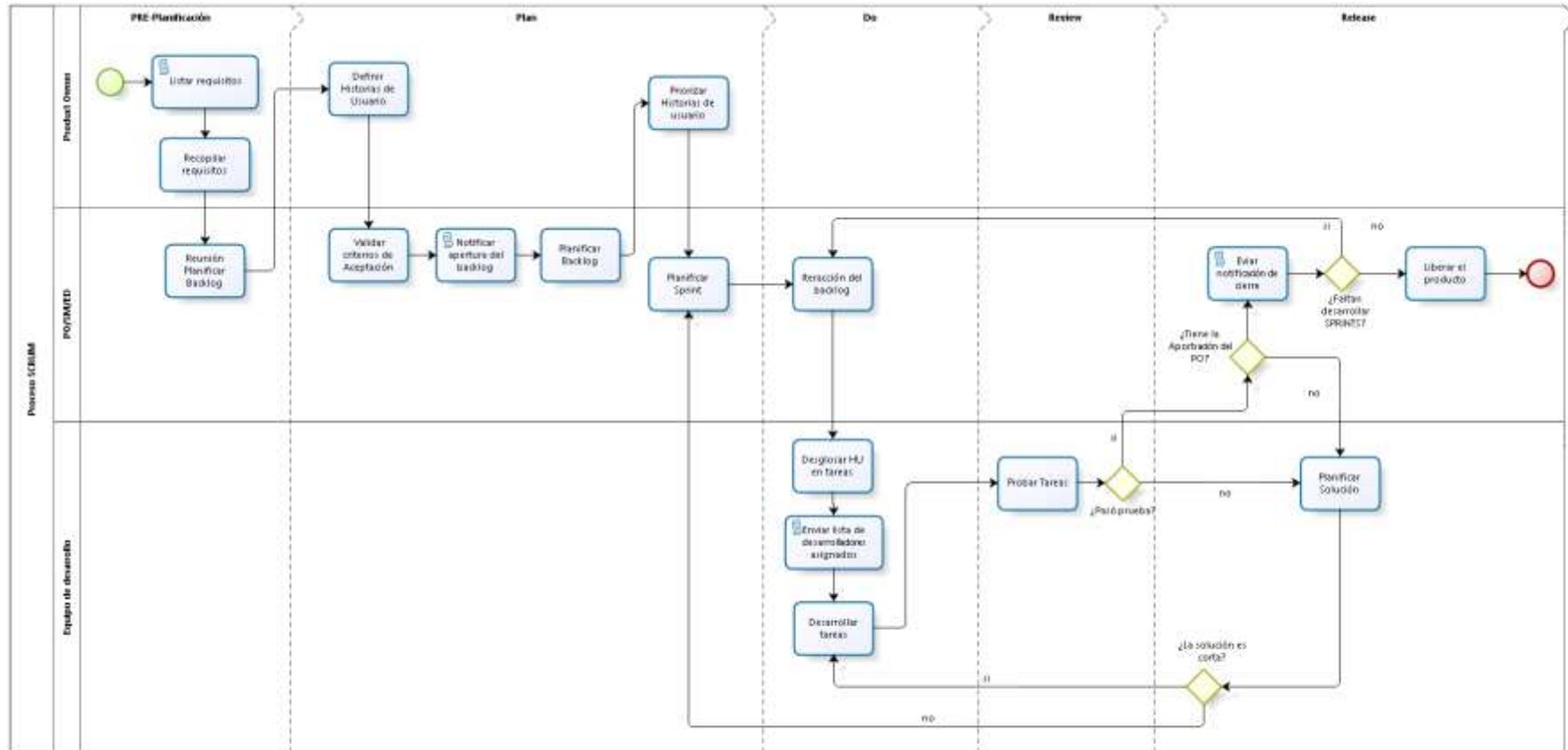


Figura 30. Metodología SCRUM.

Fuente: (Kennet, 2012)



Diagrama de procesos SCRUM



Fuente: (Bernilla & Livaque, 2019)

a. Lista de requerimientos

Se lista los siguientes requerimientos, pudiéndose visualizar el código fuente de cada uno de ellos en el ANEXO 03:

- Registrar clientes.
- Registrar productos.
- Registrar pedidos.
- Listar y seleccionar entre las Apis REST y GRAPHQL.
- Visualizar el rendimiento entre las dos Apis

b. Roles en la metodología Scrum

– *Dueño del producto*

El Dueño de Producto es el responsable de generar valor al producto y transmitir de forma clara, lo que el cliente realmente desea para que del Equipo de Desarrollo lleve a cabo el proyecto de la mejor manera posible, acercándose a las necesidades del cliente.



Figura 31. Dueño del producto

Fuente: (Alaimo, 2013)

– *Equipo de desarrollo:*

Son un grupo de profesionales, escogidos por la propia empresa para realizar el trabajo de entregar un producto funcional al final de cada Sprint.

El número de los miembros de este equipo, no deben ser muy robustos o muy escasos, según (Kennet, 2012)



Figura 32. Equipo de desarrollo

Fuente: (Alaimo, 2013)

– **Scrum Master:**

(Kennet, 2012), el Scrum Master ayuda a todos los grupos de la siguiente manera:

El **Scrum** Master da servicio al Dueño de Producto de varias formas, incluyendo:

- Encontrar técnicas para gestionar la Lista de Producto de manera efectiva.
- Entender la planificación del producto en un entorno empírico.
- Entender y practicar la agilidad.
- Facilitar los eventos de Scrum según se requiera o necesite.



Figura 33. Scrum Master

Fuente: (Alaimo, 2013)



c. Equipo de trabajo Scrum

Para este caso en específico los roles serán ejecutados por una sola persona y se detallan a continuación:

Tabla 33.
Roles Scrum

ROL	ENCARGADO
Dueño del producto	Jorge
Scrum Master	Jorge
Equipo de desarrollo	Jorge

Fuente: elaboración propia.

d. Backlog

La Lista de Producto es una lista ordenada de todos los requisitos necesarios para construir un producto funcional, y es la única fuente de requisitos para que el equipo de desarrollo pueda realizar cualquier avance o cambio en el producto. Cabe recalcar que el Dueño de Producto (Product Owner) es el único responsable de la Lista de Producto, incluyendo su contenido, disponibilidad y ordenación.

Esta actividad dura 8 horas para un sprint de un mes.

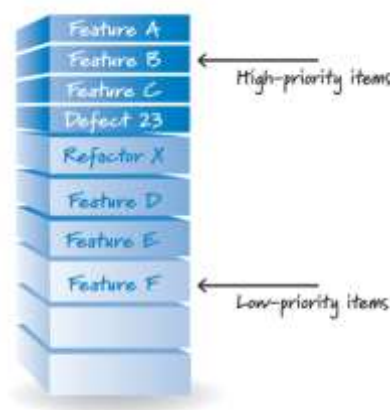


Figura 34. Product Backlog.

Fuente: (Kennet, 2012)



e. Historias de usuario

Una historia de usuario es valiosa para el comprador del Sistema o Software es por eso por lo que se planifica una descripción detallada de lo que va a contener, que se espera y que se puede probar.

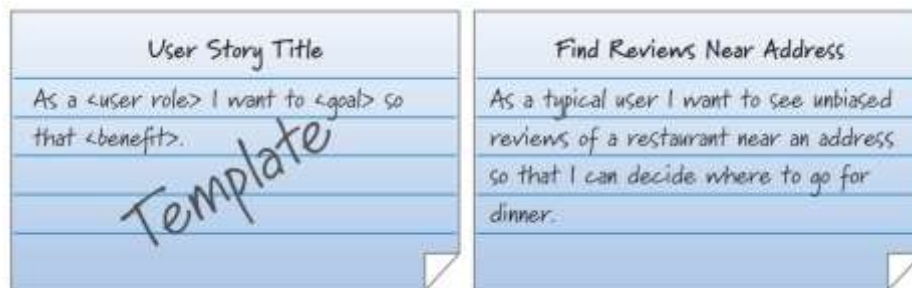


Figura 35. Historia de usuario.

Fuente: (Kennet, 2012)

f. Lista de Historias de usuario

En Scrum se muestra una lista de requerimientos refinados y detallados para ser entregados a todo el equipo Scrum para su desarrollo, esta lista viene descrita en un lenguaje técnico para ser entendido por todo el equipo, se lista a continuación las Historias de Usuario necesarias para el desarrollo de la aplicación móvil.

Tabla 34.
Lista de historias de usuario

Código	Título	Historia de Usuario	Criterio de Aceptación
001	Módulo de registro de clientes.	- Como administrador necesito registrar, modificar y eliminar mis clientes para tener una cartera de clientes actualizada.	- Debe de permitir registrar un cliente. - Datos de nombre son obligatorios. - El formulario debe de contemplar los datos de email, dirección y teléfono.



Código	Título	Historia de Usuario	Criterio de Aceptación
			<ul style="list-style-type: none"> - Mostrar un listado de mis clientes registrados. - Debe de permitir modificar los datos de los clientes. - Debe de permitir eliminar un cliente que no tiene ventas asociadas.
002	Módulo de registro de productos.	<ul style="list-style-type: none"> - Como administrador necesito modificar y eliminar mis productos para tener una cartera de productos actualizada. 	<ul style="list-style-type: none"> - Debe de permitir registrar un producto. - Todos los datos son obligatorios. - El formulario debe de contemplar los datos de nombre, precio de venta, precio de compra y stock físico. - Mostrar un listado de mis productos registrados. - Debe de permitir modificar los datos de los productos. - Debe de permitir eliminar un producto que no se encuentra en ninguna venta.
003	Módulo de registro de pedidos.	<ul style="list-style-type: none"> - Yo como administrador, necesito registrar los pedidos realizados por mis clientes, para poder controlar mis ingresos. 	<ul style="list-style-type: none"> - Seleccionar cliente. - Seleccionar productos y mostrar los precios respectivos. - Ingresar la cantidad solicitado por cada producto. - Visualizar el precio total a pagar. - Registrar el pedido.



Código	Título	Historia de Usuario	Criterio de Aceptación
004	Interfaz de una lista de APIS REST y GRAPHQL.	- Yo como administrador, necesito una opción para cambiar el tipo de API de las peticiones, para saber cuál es el API que presente mejor rendimiento.	- Mostrar una lista con ambas APIS. - Al seleccionar un ítem de la lista, dar a saber que API estoy usando.
005	Interfaz de rendimiento de APIS.	- Yo como administrador, necesito tener una interfaz donde visualizar la comparación de tiempos de respuesta, costo de CPU de las APIS REST y GRAPHQL.	- Mostrar interfaz donde visualice la cantidad de peticiones hechas por fechas. - Mostrar el indicador tiempo de respuesta. - Mostrar el indicador consumo de CPU. - Mostrar comparativas entre ambos indicadores.

Fuente: elaboración propia.

g. Priorización de Historias de usuario

Una vez que se lista las historias de usuario el dueño del producto, que es el encargado de transmitir la necesidad del cliente al equipo Scrum, enumera en orden de acuerdo a la prioridad que considera, estas historias van establecidas y diferenciadas desde la prioridad alta, media y culmina con prioridad baja. Para el desarrollo de nuestra aplicación se consideró la siguiente priorización de historias de usuario.



Tabla 35.
Priorización de historias

Código	Título	Prioridad
001	Módulo de registro de clientes.	ALTA
002	Módulo de registro de productos.	ALTA
003	Módulo de registro de pedidos.	ALTA
004	Interfaz de una lista de APIS REST y GRAPHQL.	MEDIA
005	Interfaz de rendimiento de APIS.	MEDIA

Fuente: elaboración propia.

h. Sprint

Esta reunión se lleva a cabo al inicio de cada sprint y los involucrados en son el dueño del producto, el Scrum Master y el grupo de desarrollo, el tiempo promedio no dura más de 8 horas.

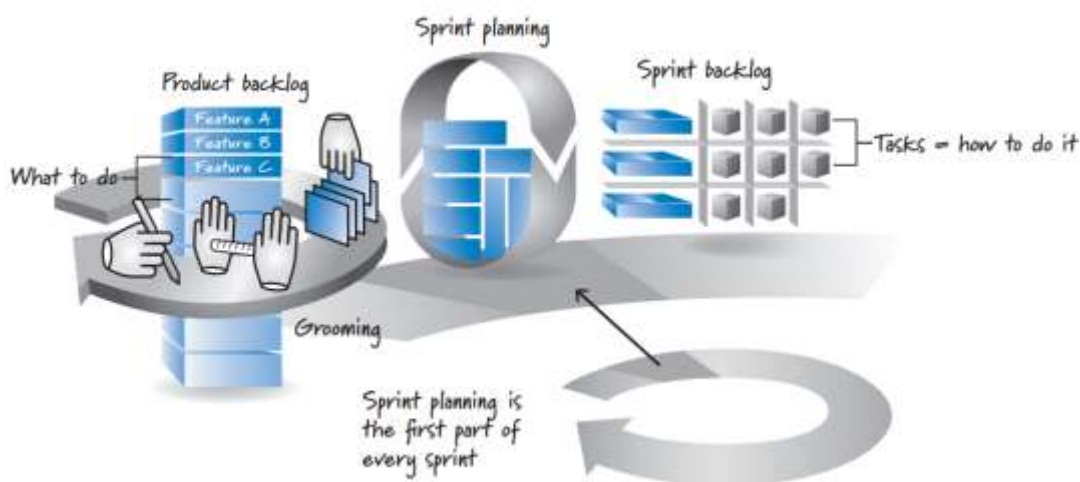


Figura 36. Planificación del Sprint.

Fuente: (Kennet, 2012)



i. Lista de Sprints

Para establecer el número y orden de los Sprints se toma en cuenta la prioridad de las historias de usuario, el tiempo y esfuerzo que demanda para su desarrollo, es así que las historias de usuario con la prioridad más alta se desarrollarán en el primer Sprint y así sucesivamente hasta culminar el desarrollo con las historias de usuario de menos prioridad.

Este desarrollo de Sprints está considerado para un promedio de cuatro meses.

– **Sprint 01**

Para el desarrollo del primer Sprint se tomaron en cuenta las Historias de usuario con mayor prioridad siendo las siguientes listadas con prioridad alta. El tiempo de desarrollo se ha considerado de 06 semanas.

Tabla 36.
Primer Sprint

Código	Título	Prioridad
001	Módulo de registro de clientes.	ALTA
002	Módulo de registro de productos.	ALTA
003	Módulo de registro de pedidos.	ALTA

Fuente: elaboración propia.



– **Sprint 02**

Para el desarrollo del segundo Sprint se tomaron en cuenta las Historias de usuario con menor prioridad siendo las siguientes listadas con prioridad media. El tiempo de desarrollo se ha considerado de 02 semanas.

Tabla 37.
Segundo Sprint

Código	Título	Prioridad
004	Interfaz de una lista de APIS REST y GraphQL.	MEDIA
005	Interfaz de rendimiento de APIS.	MEDIA

Fuente: elaboración propia.

j. Modelo Vista Presentador en Android

MVP es otro patrón de diseño que tiene como objetivo separar la interfaz de usuario de la lógica de las aplicaciones.

Básicamente este patrón consiste en 3 componentes:

- La vista. Compuesta de las ventanas y controles que forman la interfaz de usuario de la aplicación.
- El modelo. Que es donde se lleva a cabo toda la lógica de negocio.
- El presentador. Escucha los eventos que se producen en la vista y ejecuta las acciones necesarias a través del modelo. Además, puede tener acceso a las vistas a través de las interfaces que la vista debe implementar. (Arrivi, 2010)



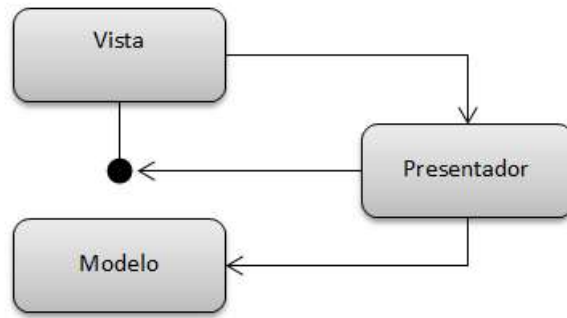


Figura 37. Gráfica de la distribución para arquitectura de solución.
Fuente: (Arrivi, 2010).

k. Modelo de datos

Se diseñó la base de datos de acuerdo a los requerimientos del aplicativo móvil, que será una app de ventas.

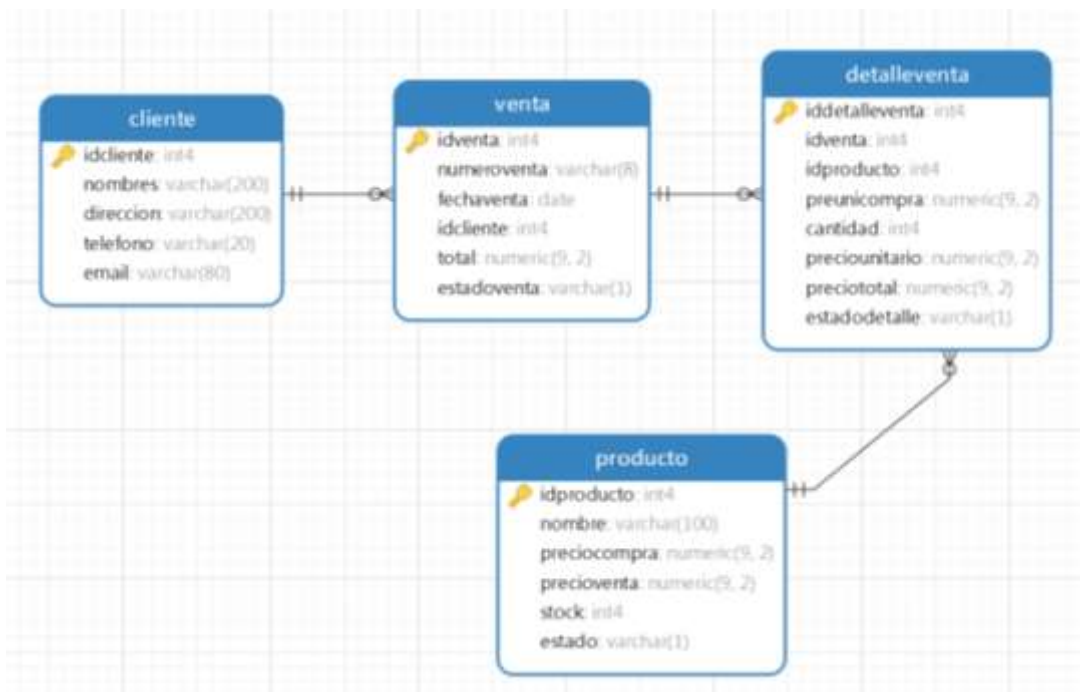


Figura 38. Modelo de datos
Fuente: Elaboración propia



1. Diagrama de clases

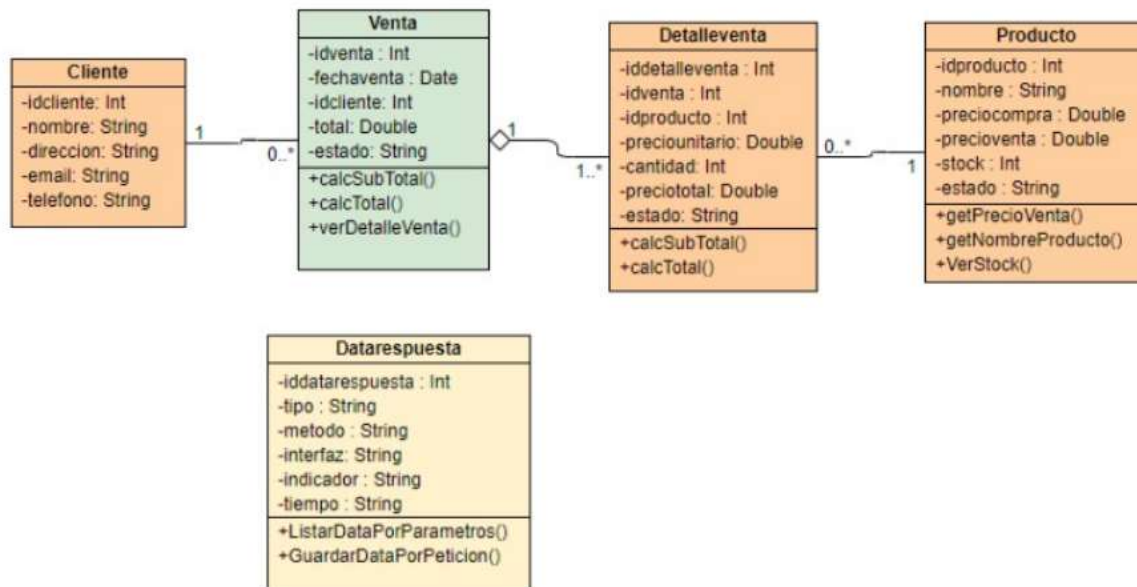


Figura 39. Diagrama de clases

Fuente: elaboración propia

m. Desarrollo de Sprints

SPRINT 01

Se muestran las pantallas de los módulos propios de la aplicación, entre ellos están los módulos de clientes, productos y ventas.

- **HU 001:** Módulo de registro de clientes.

Entradas: Datos del cliente



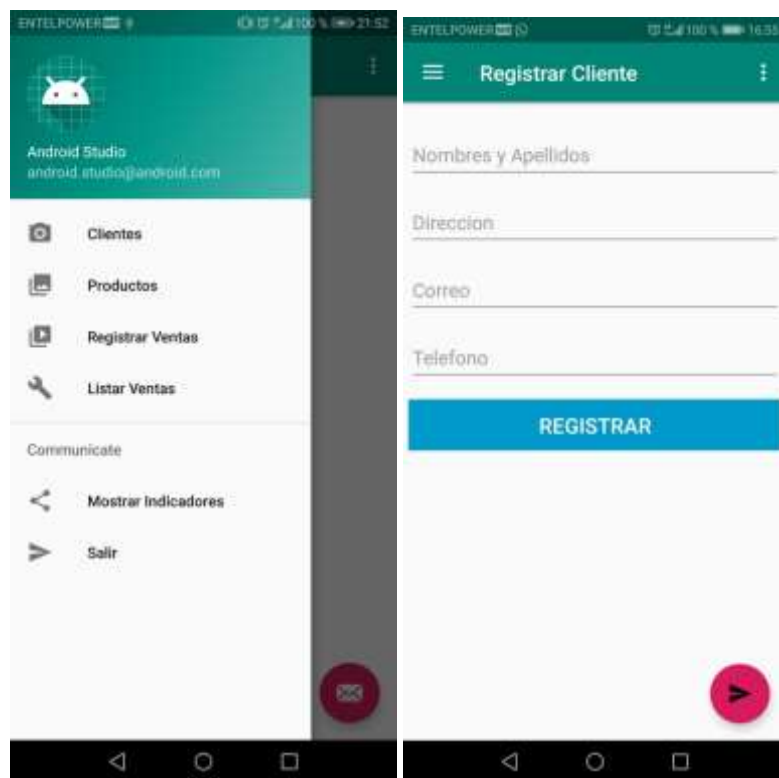


Figura 40. Interfaces de datos de clientes

Fuente: Elaboración propia.

Proceso: En este proceso se registrará uno a uno los clientes con los campos obligatorios solicitados, del mismo modo se puede modificar y eliminar algún cliente de la lista.

Resultados esperados:

- Debe de permitir registrar un cliente.
- Datos de nombre son obligatorios.
- El formulario debe de contemplar los datos de email, dirección y teléfono.



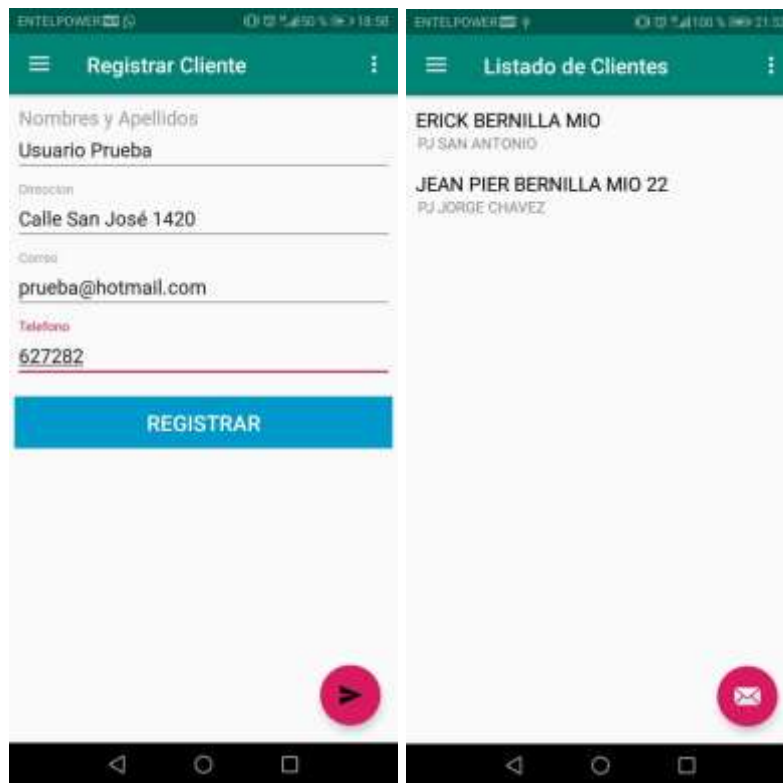


Figura 41. Interfaces de registro de clientes.

Fuente: Elaboración propia.

- Mostrar un listado de mis clientes registrados.
- Debe de permitir modificar los datos de los clientes.
- Debe de permitir eliminar un cliente que no tiene ventas asociadas.



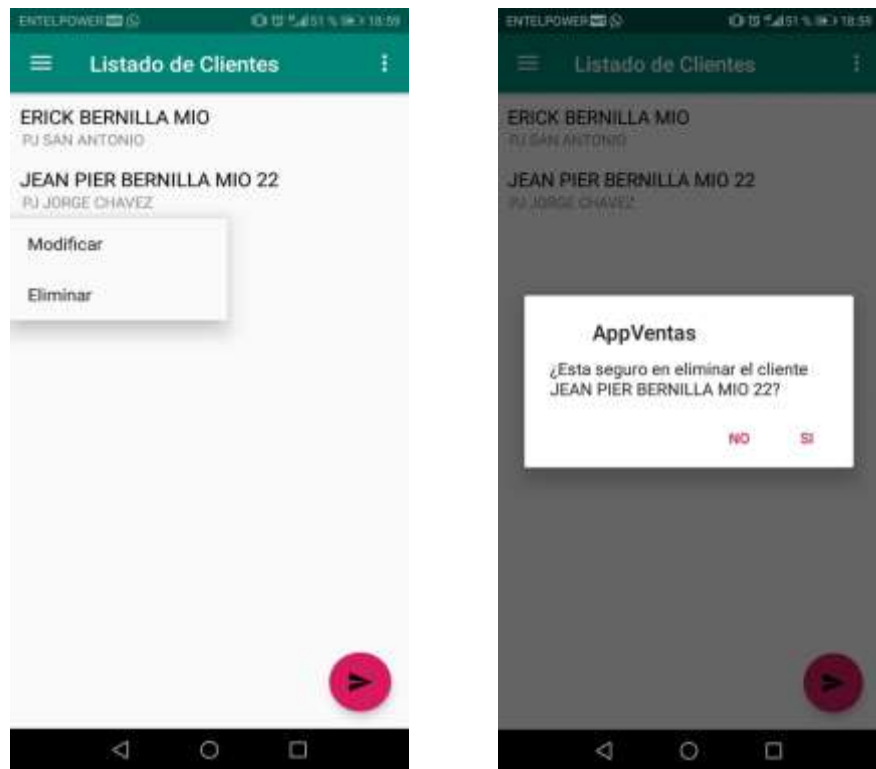


Figura 42. Interfaces de modificar y eliminar clientes.

Fuente: Elaboración propia.

- **HU 002:** Módulo de registro de productos.

Entradas: Nombre del producto, precio compra, precio venta y stock



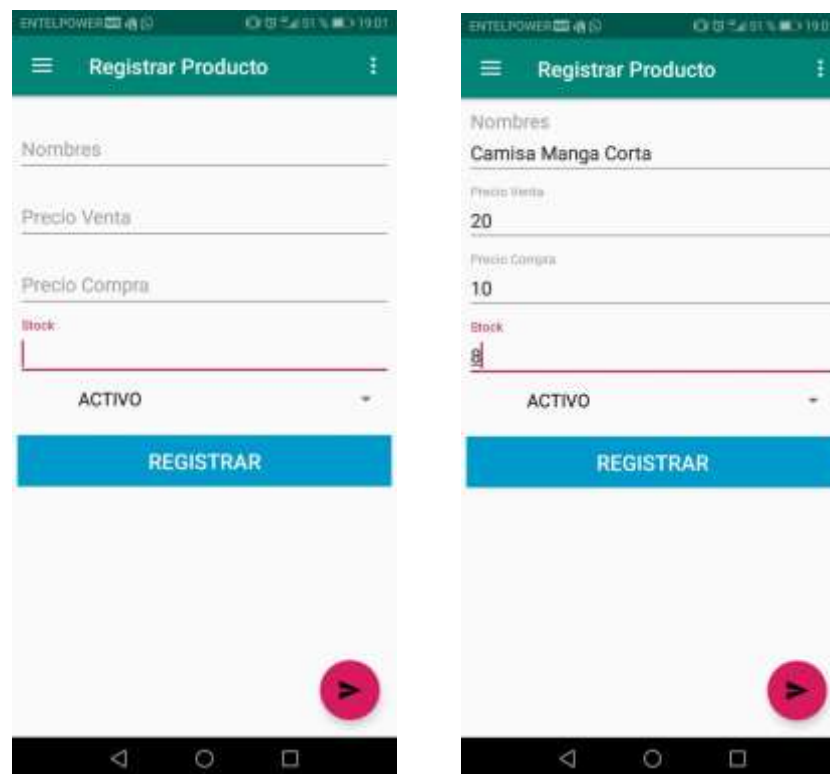


Figura 43. Interfaces de información de productos.

Fuente: Elaboración propia.

Proceso: En este proceso se registrará uno a uno los productos con los campos obligatorios solicitados, del mismo modo se puede modificar y eliminar algún producto de la lista.

Resultados esperados:

- Debe de permitir registrar un producto.
- Todos los datos son obligatorios.
- El formulario debe de contemplar los datos de nombre, precio de venta, precio de compra y stock físico.
- Mostrar un listado de mis productos registrados.
- Debe de permitir modificar los datos de los productos.



- Debe de permitir eliminar un producto que no se encuentra en ninguna venta.
- Seleccionar cliente.
- Seleccionar productos y mostrar los precios respectivos.
- Ingresar la cantidad solicitado por cada producto.
- Visualizar el precio total a pagar.
- Registrar el pedido.

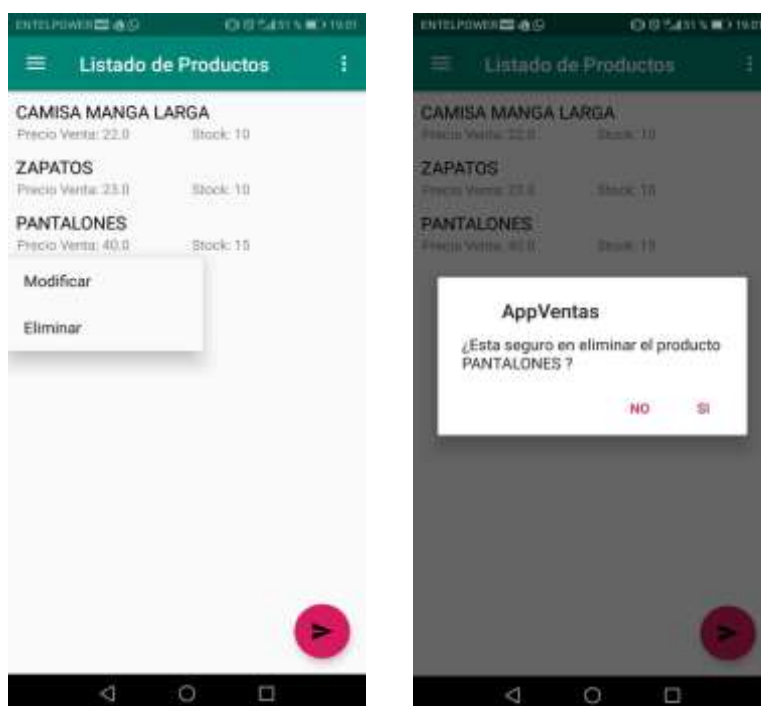


Figura 44. Interfaces de modificar y eliminar productos.

Fuente: Elaboración propia.

- **HU 003:** Módulo de registro de pedidos

Entradas: Cliente y selección de producto.



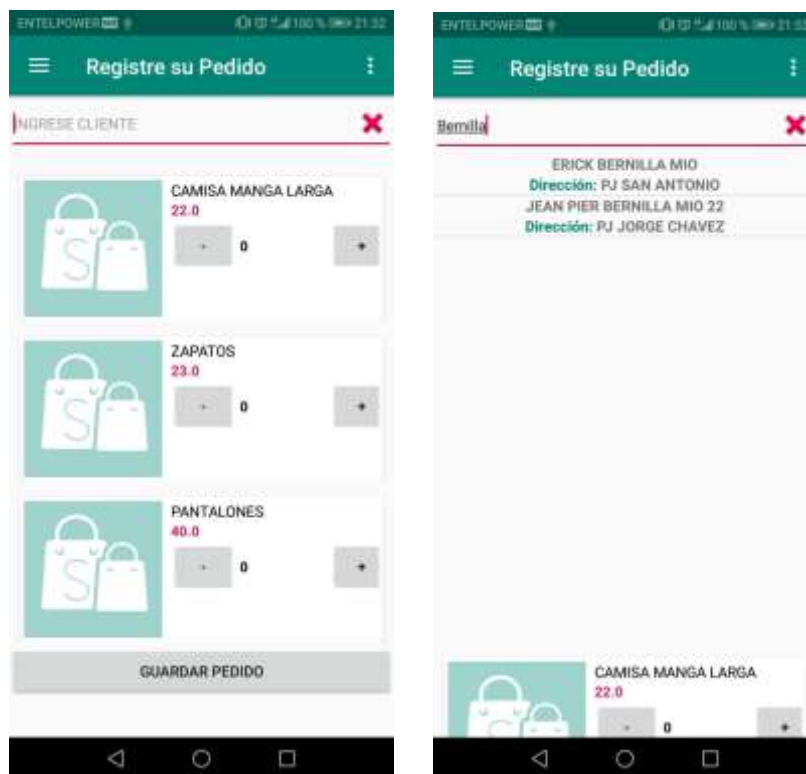


Figura 45. Interfaces de registro de pedidos

Fuente: Elaboración propia.

Proceso: En este proceso se seleccionará a un cliente y el producto que desea pedir.

Resultados esperados:

- Seleccionar cliente.
- Seleccionar productos y mostrar los precios respectivos.
- Ingresar la cantidad solicitado por cada producto.
- Visualizar el precio total a pagar.
- Registrar el pedido.



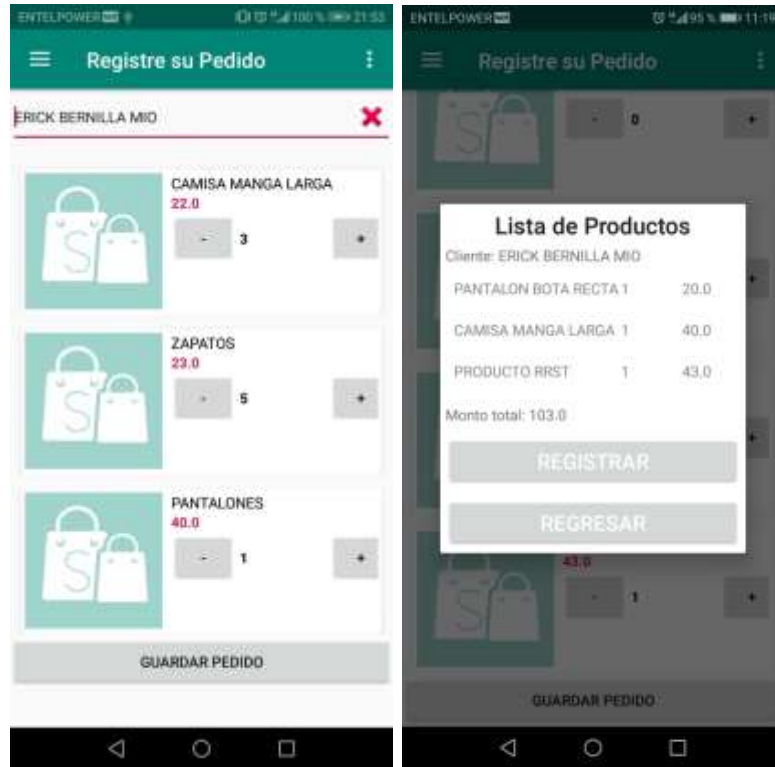


Figura 46. Interfaz de registro de ventas

Fuente: Elaboración propia.

SPRINT 02:

Se muestran las pantallas de los módulos propios de la aplicación, entre ellos están los resultados finales y la comparación entre las tecnologías Rest y GraphQL.



- **HU 004:** Interfaz de una lista de APIS REST y GRAPHQL

Entradas: Apis Rest y GraphQL.

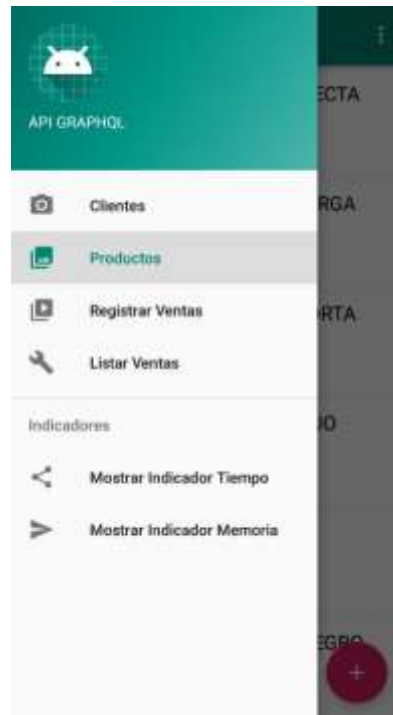


Figura 47. Interface de proceso productos

Fuente: Elaboración propia.

Proceso: En este proceso se visualizará las APIS REST y GRAPHQL.

Resultados esperados:

- Mostrar una lista con ambas APIS.
- Al seleccionar un ítem de la lista, dar a saber que API estoy usando.



Figura 48. Interfaz de selección de API

Fuente: Elaboración propia.

Código fuente Rest:

Para descargar

```
package com.api.rest.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.InputStreamResource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import javax.servlet.ServletContext;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
```

```

@Controller
public class DescargaController {

    @Autowired
    private ServletContext context;
    @Autowired
    private HttpHeaders httpHeaders;

    @RequestMapping(value = "/appventa", method = RequestMethod.GET, produces = "application/apk")
    public ResponseEntity<InputStreamResource> downloadAppImpresion() throws IOException {
        String path = "/resources/app/appventa.apk";
        String ruta = context.getRealPath(path);
        File file = new File(ruta);
        if (!file.exists()) {
            file = new File("D:/test.apk");
            if (!file.exists()) {
                throw new FileNotFoundException("Oops! File not found");
            }
        }
        InputStreamResource isResource = new InputStreamResource(new FileInputStream(file));
        FileSystemResource fileSystemResource = new FileSystemResource(file);
        String fileName = "appventa.apk";
        fileName = new String(fileName.getBytes("UTF-8"), "iso-8859-1");
        httpHeaders.setContentLength(fileSystemResource.contentLength());
        httpHeaders.setContentDispositionFormData("attachment", fileName);
        return new ResponseEntity<InputStreamResource>(isResource, httpHeaders, HttpStatus.OK);
    }
}

```

Para cargar indicadores

```

@Controller
public class RespuestaController {

    @Autowired
    private RespuestaService respuestaService;

    @GetMapping("listarRespuesta")
    public @ResponseBody List<DataRespuesta> ListarRespuestas(@ModelAttribute(value="tipo") String tipo,
        @ModelAttribute(value="interfaz") String interfaz,@ModelAttribute(value="metodo") String metodo,
        @ModelAttribute(value="indicador") String indicador) throws Exception {
        List<DataRespuesta> dato = new ArrayList<DataRespuesta>();
        try{
            dato = respuestaService.ListarRespuestasxTipo(tipo,interfaz,metodo,indicador);
        }catch (Exception e){
            return dato;
        }
        return dato;
    }

    @PostMapping("saveDataRespuesta")
    public Respuesta RegistrarTiempoRespuesta(@RequestBody DataRespuesta tiemporespuesta){
        Respuesta respuesta = new Respuesta();
        try{
            Object dato = respuestaService.RegistrarRespuesta(tiemporespuesta);
            respuesta.setDato("ERROR");

            if(dato.toString().equals("1")){
                respuesta.setDato("OK");
            }
        }catch (Exception e){
            respuesta.setDato("ERROR");
            respuesta.setMensaje(e.getMessage());
        }
        return respuesta;
    }
}

```

Para las ventas


```

@RestController
public class VentaController {

    @Autowired
    private VentaService ventaService;

    @GetMapping("listarVentas")
    public @ResponseBody List<Venta> ListarVentas(@ModelAttribute(value="fechaVenta") String fechaVenta) {
        List<Venta> dato = new ArrayList<Venta>();
        try{
            dato = ventaService.ListarVentas(new Date());
        }catch (Exception e){
            return dato;
        }
        return dato;
    }

    @PostMapping("saveVenta")
    public Respuesta RegistrarVenta(@RequestBody Venta venta){
        Respuesta respuesta = new Respuesta();
        try{
            Object dato = ventaService.RegistrarVenta(venta);
            respuesta.setDato("ERROR");

            if(dato.toString().equals("1")){
                respuesta.setDato("OK");
            }
        }catch (Exception e){
            respuesta.setDato("ERROR");
            respuesta.setMensaje(e.getMessage());
        }
        return respuesta;
    }
}
    
```

Para los clientes

```

@RestController
public class ClienteController {

    @Autowired
    private ClienteService clienteService;

    @GetMapping("listarClientes")
    public @ResponseBody
    List<Cliente> ListarClientes(@ModelAttribute(value="descripcion") String
descripcion) throws Exception {
        List<Cliente> dato = new ArrayList<Cliente>();
        try{
            dato = clienteService.ListarClientesxDescripcion(descripcion);
        }catch (Exception e){
            return dato;
        }
        return dato;
    }

    @PostMapping("saveCliente")
    public Respuesta RegistrarCliente(@RequestBody Cliente cliente){
        Respuesta respuesta = new Respuesta();
        try{
            Object dato = clienteService.RegistrarCliente(cliente);
            respuesta.setDato("ERROR");

            if(dato.toString().equals("1")){
                respuesta.setDato("OK");
            }
        }catch (Exception e){
            respuesta.setDato("ERROR");
            respuesta.setMensaje(e.getMessage());
        }
        return respuesta;
    }

    @PostMapping("updateCliente")
    public Respuesta ModificarCliente(@RequestBody Cliente cliente){
        Respuesta respuesta = new Respuesta();
        try{
            Object dato = clienteService.ModificarCliente(cliente);
            respuesta.setDato("ERROR");

            if(dato.toString().equals("1")){

```

```

        respuesta.setDato("OK");
    }else if(dato.toString().equals("2")){
        respuesta.setMensaje("Error en base de datos.");
    }
}catch (Exception e){
    respuesta.setDato("ERROR");
    respuesta.setMensaje(e.getMessage());
}
return respuesta;
}

@PostMapping("deleteCliente")
public Respuesta QuitarCliente(@RequestBody Integer idCliente){
    Respuesta respuesta = new Respuesta();
    try{
        Object dato = clienteService.EliminarCliente(idCliente);
        respuesta.setDato("ERROR");
        if(dato.toString().equals("1")){
            respuesta.setDato("OK");
        }else{
            respuesta.setMensaje(dato.toString());
        }
    }catch (Exception e){
        respuesta.setDato("ERROR");
        respuesta.setMensaje(e.getMessage());
    }
    return respuesta;
}
}
}

```

Para los productos

```

@RestController
public class ProductoController {

    @Autowired
    private ProductoService productoService;

    @GetMapping("listarProductos")
    public @ResponseBody List<Producto> ListarProductos(@ModelAttribute(value="descripcion") String descripcion) throws Exception {
        List<Producto> dato = new ArrayList<Producto>();
        try{
            dato = productoService.ListarProductosxDescripcion(descripcion);
        }catch (Exception e){
            return dato;
        }
        return dato;
    }

    @PostMapping("saveProducto")
    public Respuesta RegistrarProducto(@RequestBody Producto producto){
        Respuesta respuesta = new Respuesta();
        try{
            Object dato = productoService.RegistrarProducto(producto);
            respuesta.setDato("ERROR");

            if(dato.toString().equals("1")){
                respuesta.setDato("OK");
            }
        }catch (Exception e){
            respuesta.setDato("ERROR");
            respuesta.setMensaje(e.getMessage());
        }
        return respuesta;
    }

    @PostMapping("updateProducto")
    public Respuesta ModificarProducto(@RequestBody Producto producto){
        Respuesta respuesta = new Respuesta();
        try{
            Object dato = productoService.ModificarProducto(producto);
            respuesta.setDato("ERROR");

            if(dato.toString().equals("1")){
                respuesta.setDato("OK");
            }
        }
    }
}

```

```

        }else if(dato.toString().equals("2")){
            respuesta.setMensaje("Error en base de datos.");
        }
    }catch (Exception e){
        respuesta.setDato("ERROR");
        respuesta.setMensaje(e.getMessage());
    }
    return respuesta;
}

@PostMapping("deleteProducto")
public Respuesta QuitarProducto(@RequestBody Integer idProducto){
    Respuesta respuesta = new Respuesta();
    try{
        Object dato = productoService.EliminarProducto(idProducto);
        respuesta.setDato("ERROR");
        if(dato.toString().equals("1")){
            respuesta.setDato("OK");
        }else{
            respuesta.setMensaje(dato.toString());
        }
    }catch (Exception e){
        respuesta.setDato("ERROR");
        respuesta.setMensaje(e.getMessage());
    }
    return respuesta;
}
}
}

```

Código fuente GraphQL:

```

const { ApolloServer } = require('apollo-server');
const { makeExecutableSchema } = require('graphql-tools');
const Sequelize = require('sequelize');

const sequelize = new Sequelize('db_venta', 'master', '766767-422', {
  host: 'hotel.clti5xvn59je.us-east-1.rds.amazonaws.com',
  port: 5432,
  dialect: 'postgres'
});

sequelize.authenticate().then(() => {
  console.log('Conectado')
}).catch(err => {
  console.log('No se conecto')
})

const Producto = sequelize.define('producto', {
  idproducto: {
    type: Sequelize.SMALLINT,
    primaryKey: true,
    autoIncrement: true,
    allowNull: false
  },
  nombre: {
    type: Sequelize.STRING,
    allowNull: false,
    unique: true
  },
  preciocompra : Sequelize.DOUBLE,
  precioventa: Sequelize.DOUBLE,
  stock : {
    type: Sequelize.INTEGER,
    defaultValue: 0
  },
  estado: Sequelize.STRING,
}, {
  timestamps: false,
  paranoid: true,
  underscored: true,
  freezeTableName: true,
  tableName: 'producto'
});

```

```

const Cliente = sequelize.define('cliente', {
  idcliente: {
    type: Sequelize.SMALLINT,
    primaryKey: true,
    autoIncrement: true,
    allowNull: false
  },
  nombres:{
    type: Sequelize.STRING,
    allowNull : false
  },
  direccion : Sequelize.STRING,
  telefono: Sequelize.STRING,
  email: Sequelize.STRING
} , {
  timestamps: false,
  paranoid: true,
  underscored: true,
  freezeTableName: true,
  tableName: 'cliente'
});

const Venta = sequelize.define('venta', {
  idventa: {
    type: Sequelize.SMALLINT,
    primaryKey: true,
    autoIncrement: true,
    allowNull: false
  },
  numeroventa:{
    type: Sequelize.STRING,
    allowNull : false
  },
  fechaventa : Sequelize.DATE,
  idcliente: {
    type: Sequelize.INTEGER,
    //constraints: false
    references: {
      model: 'cliente',
      key: 'idcliente'
    }
  },
  total: Sequelize.DOUBLE,
  estadoventa: Sequelize.STRING
} , {
  timestamps: false,
  paranoid: true,

```

```

    underscored: true,
    freezeTableName: true,
    tableName: 'venta'
  });

const DetalleVenta = sequelize.define('detalleventa', {
  iddetalleventa: {
    type: Sequelize.SMALLINT,
    primaryKey: true,
    autoIncrement: true,
    allowNull: false
  },
  idventa: {
    type: Sequelize.INTEGER,
    references: {
      model: 'venta',
      key: 'idventa'
    }
  },
  idproducto: {
    type: Sequelize.INTEGER,
    references: {
      model: 'producto',
      key: 'idproducto'
    }
  },
  preunicompra: Sequelize.DOUBLE,
  cantidad: Sequelize.INTEGER,
  preciounitario: Sequelize.DOUBLE,
  preciototal: Sequelize.DOUBLE,
  estadodetalle: Sequelize.STRING
} , {
  timestamps: false,
  paranoid: true,
  underscored: true,
  freezeTableName: true,
  tableName: 'detalleventa'
});

Venta.belongsTo(Cliente , { as : 'cliente', foreignKey : 'idcliente'});
DetalleVenta.belongsTo(Venta, { as : 'venta', foreignKey: 'idventa'});
DetalleVenta.belongsTo(Producto, { as : 'producto', foreignKey: 'idproducto'
})

const typeDefs = `
  type Cliente {
    idcliente: ID!

```



```

    nombres: String!
    direccion : String!
    telefono: String!
    email: String!
}

type Producto {
  idproducto: ID!
  nombre: String!
  preciocompra: Float!
  precioventa: Float!
  stock: Int!
  estado: String!
}

type Venta {
  idventa: ID!
  numeroventa: String!
  fechaventa: String!
  total: Float!
  estadoventa: String!
  cliente: Cliente!
}

type DetalleVenta {
  iddetalleventa: ID!
  venta: Venta!
  producto: Producto!
  preunicompra: Float!
  cantidad: Int!
  preciounitario: Float!
  preciototal: Float!
  estadodetalle: String!
}

type Hello {
  message: String!, link: String!, image: String!,id: Int!
}

type Query {
  sayHello(name: String!): Hello
  productos: [Producto!]
  clientes: [Cliente!]
  producto(id: ID!): Producto!
  cliente(id: ID!): Cliente!
  ventas : [Venta!]
  detalles(idventa : ID!) : [DetalleVenta!]
}

```

```

    }

    type Mutation {
      createProducto(nombre: String!,preciocompra : Float!,precioventa: Fl
      oat!,stock: Int!, estado : String!): Producto!
      createCliente(nombres: String!,direccion: String!,telefono: String!,
      email: String!) : Cliente!
      updateCliente(idcliente: Int!,nombres: String!,direccion: String!,te
      lefono: String!,email: String!) : Cliente!
      updateProducto(idproducto: Int!,nombre: String!,preciocompra : Float
      !,precioventa: Float!,stock: Int!, estado : String!): Producto!
    } `;

const resolvers = {
  Query: {
    sayHello: (_,args) => {
      return {
        message: `Hello ${args.name || 'world'}`,
        link: "erick",
        image : "imagen 22222",
        id : 22
      }
    },
    productos: async (_,args) => {
      return await Producto.findAll();
    },
    clientes: async (_,args) => {
      return await Cliente.findAll();
    },
    producto: async (_,args) => {
      return await Producto.findByPk(args.id);
    },
    cliente: async (_,args) => {
      return await Cliente.findByPk(args.id);
    },
    ventas: async (_,args) => {
      return await Venta.findAll( { include : ['cliente'] } );
      //Venta.findAll({ offset: 5, limit: 5 })
    },
    detalles: async (_,args) => {
      return await DetalleVenta.findAll({
        include:['producto'],
        where: { idventa: args.idventa }
      });
    }
  },
  Mutation: {

```

```

    createProducto: async (_, {nombre,preciocompra,precioventa,estado})
=> {
        return await Producto.create({ nombre,preciocompra,precioventa,e
stado});
    },
    updateProducto: async (_,{idproducto,nombre,preciocompra,precioventa
,estado}) => {
        const producto = await Producto.findByPk(idproducto);
        producto.set('nombre', nombre);
        producto.set('preciocompra',preciocompra);
        producto.set('precioventa',precioventa);
        producto.set('estado',estado);
        return producto.save();
    },
    createCliente: async (_,{nombres,direccion,telefono,email}) => {
        return await Cliente.create({
            nombres,direccion,telefono,email
        });
    },
    updateCliente: async (_,{idcliente,nombres,direccion,telefono,email}
) => {
        const cliente = await Cliente.findByPk(idcliente);
        cliente.set('nombres', nombres);
        cliente.set('telefono',telefono);
        cliente.set('email',email);
        cliente.set('direccion',direccion);
        return cliente.save();
    }
}
}

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

const apolloServer = new ApolloServer({
  schema: schema,
  introspection: true, // enables introspection of the schema
  playground: true // enables the actual playground
});

apolloServer.listen(5000).then( ({ url }) => {
  console.log("ejecutandose en " + url);
});

```

- **HU 005:** Interfaz de rendimiento de APIS.

Entradas: Ejecución de procesos.

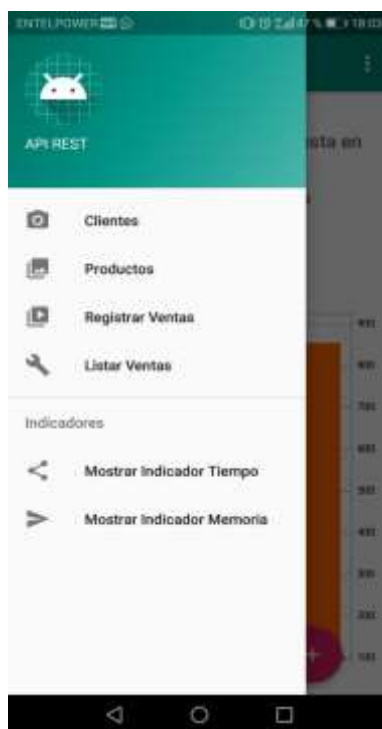


Figura 49. Interfaz de resultados REST - GraphQL.

Fuente: Elaboración propia.

Proceso: En este proceso se ejecutarán procesos de pedidos tanto en REST como GraphQL y se mostrará en un cuadro estadístico los resultados de dichos procesos para analizarlos y compararlos.

Resultados esperados:

- Mostrar interfaz donde visualice la cantidad de peticiones hechas por fechas.
- Mostrar el indicador tiempo de respuesta.
- Mostrar el indicador consumo de CPU.
- Mostrar comparativas entre ambos indicadores.



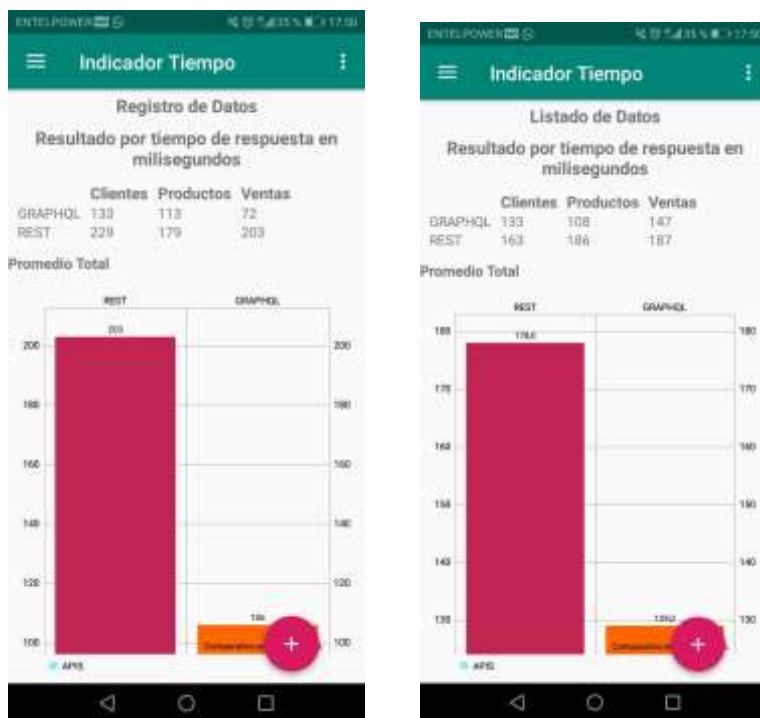


Figura 50. Gráfico de indicador de tiempo REST - GraphQL.

Fuente: Elaboración propia.



Figura 51. Gráfico de indicador de consumo RAM REST-GraphQL.

Fuente: Elaboración propia.



3.3.5. Evaluar los resultados para las tecnologías GraphQL y Rest

Para evaluar los resultados tenemos el siguiente cuadro resumen, donde se detalla el tipo de tecnología, el método usado para las llamadas, la cantidad (Milisegundos – megabytes) según sea el caso de acuerdo al registro y listado de datos.

Tabla 38.
Rendimiento de las tecnologías REST y GRAPHQL

	TECNOLOGÍA EVALUADA	MÉTODO	PROCESO	Consumo de RAM (Mb)	Promedio Consumo de RAM (Mb)	Tiempo de Respuesta (ms)	Promedio Tiempo de Respuesta (ms)
Listado de datos	REST	QUERY	Clientes	6	6	163	178
			Productos	5		186	
			Ventas	9		187	
	GRAPHQL	GET	Clientes	3	3	133	129
			Productos	4		108	
			Ventas	4		147	
Registro de datos	REST	MUTATION	Clientes	9	7	229	203
			Productos	5		179	
			Ventas	8		203	
	GRAPHQL	POST	Clientes	3	5	133	106
			Productos	4		113	
			Ventas	8		72	

Fuente: Elaboración propia

Los resultados obtenidos para los procesos de clientes, productos y ventas, en cuanto al tiempo de respuesta REST tiene la mayor cantidad con 178 milisegundos en cuanto al listado de datos, mientras que se obtuvo 203 milisegundos en el registro de datos; por otro lado, en GraphQL el tiempo total fue de 129 milisegundos en cuanto al listado de datos y 109 milisegundos en el registro de datos. Respecto al consumo de RAM, por el dispositivo se obtuvo que REST consume mayor memoria



con 6 Mb en cuanto al listado de datos y 7 Mb en el registro de datos, mientras que GraphQL consume 3 Mb en cuanto al listado de datos y 5 Mb en el registro de datos. Con estos datos podemos ver que GraphQL es más eficiente en cuanto al rendimiento de tipo y consumo de memoria RAM.

CAPÍTULO IV: CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- En esta investigación se evaluó la eficiencia de las tecnologías GraphQL y Rest en la implementación de servicios web consumidos desde aplicaciones móvil Android, bajo la norma ISO/IEC 25010 en cuanto a rendimiento de tiempo de demora de llamadas y consumo RAM al ejecutar procesos como clientes, productos y ventas.
- Se realizó la selección de la arquitectura de comunicación que establecen las tecnologías GraphQL y Rest en base a antecedentes de investigaciones anteriores, beneficios, deficiencias y el factor cronológico en cuanto a uso de tecnologías para el desarrollo de aplicaciones móviles.
- Se seleccionaron las dimensiones e indicadores de medición de calidad de software para las tecnologías GraphQL y REST, tomando en cuenta investigaciones anteriores de evaluación de arquitecturas donde determinan que el interés de medir estas tecnologías se centra en el rendimiento. La norma ISO/IEC 25010 consta de dimensiones e indicadores a los cuales se han dado valoraciones entre 0-5 medidos mediante una escala de Likert, donde el rendimiento tuvo una mejor puntuación, siendo 5 la escala más alta de importancia para las tecnologías GraphQL y Rest.
- Se seleccionaron procesos CRUD que se tomarán en cuenta para la medición de llamadas a la API para las tecnologías GraphQL y REST, siendo valoradas en un grado de importancia de 0 – 100; siendo las más altas registrar (90 puntos) y listar (81 puntos), mientras que eliminar y actualizar tuvieron menos puntaje, ambas con 70 puntos. Se han tomado los cuatro procesos CRUD dado que todos se ven involucrados en las actividades dentro de la API de venta de ropa; siendo los procesos más utilizados registrar y listar.

- Se implementó una API tanto en GraphQL y REST en base a la metodología SCRUM, se detalló cada requerimiento en una historia de usuario, se listó en el Backlog, se hizo la priorización y se programaron el desarrollo de los Sprints siendo 02 en total, el primero con una duración de 06 semanas donde se desarrolló los módulos de clientes, productos y pedidos; el otro Sprint consta de la Interfaz de la API Rest - GraphQL y la interfaz de rendimiento de las API's programadas para ser desarrolladas en 02 semanas, se desarrolló con éxito siguiendo todos los procesos de Scrum como método ágil para proyectos pequeños como el desarrollado en esta investigación.
- Al evaluar los resultados obtenidos para las tecnologías GraphQL y Rest, haciendo las pruebas respectivas de consultas, se consideró la cantidad de 30 llamadas para los procesos de clientes, productos y ventas; lo que trajo como resultado que en cuanto al tiempo de respuesta REST tiene la mayor cantidad con 178 milisegundos en cuanto al listado de datos y 203 milisegundos en el registro de datos, mientras que en GraphQL el tiempo total fue de 129 milisegundos en cuanto al listado de datos y 109 milisegundos en el registro de datos, con respecto al consumo de RAM por el dispositivo se obtuvo que REST consume mayor memoria con 6 Mb en cuanto al listado de datos y 7 Mb en el registro de datos, mientras que GraphQL consume 3 Mb en cuanto al listado de datos y 5 Mb en el registro de datos.
- Con esto se concluye que luego de evaluar el tiempo de respuesta y consumo de RAM la tecnología GraphQL resulta ser mejor en cuanto a medición de consumo de tiempo y recursos según la dimensión de rendimiento de la norma ISO/IEC 25010 para APIS compatibles con dispositivos Android.

RECOMENDACIONES

- Se recomienda evaluar APIS con otras tecnologías para poder determinar si el tiempo de respuesta y consumo de RAM supera a la tecnología REST.
- Se puede evaluar más aspectos de la norma ISO 25001 para determinar si la API está considerada como un software de calidad.
- Se recomienda hacer pruebas con un número mayor y menor de peticiones y comparar si REST y GraphQL se comportan de la misma manera.
- Se puede crear y ejecutar procesos más robustos para evaluar el comportamiento y eficacia de las tecnologías de REST y GraphQL para corroborar que los indicadores de que REST es más rápida se mantiene con respecto a GraphQL.

REFERENCIAS

- Alaimo, D. M. (2013). *Proyectos ágiles con Scrum: flexibilidad, aprendizaje, innovación y colaboración en contextos complejos*. Buenos Aires: Kleer.
- Albertos, E. (2018). *Arquitecturas Software para Microservicios: Una revisión sistemática de la Literatura* (tesis de maestría). Universidad Politécnica de Madrid, Madrid, España.
- Atencio, D., & Mamani, D. (2017). *Interconectividad basada en API REST en aplicaciones de la Municipalidad Provincial de Lampa* (tesis de pre grado). Universidad Nacional del Altiplano, Puno, Perú.
- Arrivi, O. (3 de Octubre de 2010). *El blog de Oscar Arrivi* . Obtenido de <http://theartoftheleftfoot.blogspot.pe/2010/10/el-patron-modelo-vista-presentador-mvp.html>
- Barry, D. (2013). *Web Services, Service-Oriented Architectures, and Cloud Computing*. Recuperado de <https://www.sciencedirect.com/book/9780123983572/web-services-service-oriented-architectures-and-cloud-computing>
- Braubach, L., Murillo, J., Kaviani, N., Lama, M., Burgueño, L., Moha, N., & Oriol, M. (2017). *Service Oriented Computing – ICSOC 2017 Workshops*. Madrid, España: Springer.
- Brito, G., Mombach, T., & Valente, M. (Enero de 2019). *Migrating to GraphQL: A Practical Assessment*. Conferencia: SANER en Hangzhou, China.
- Burk, N. (20 de marzo de 2018). *Top 5 Reasons to use GraphQL*. [Mensaje en un blog]. Recuperado de <https://www.prisma.io/blog/top-5-reasons-to-use-graphql-b60cfa683511>



- Burgos, L. (2017). *Análisis y evaluación de las Arquitecturas REST Y SOAP, para el desarrollo de servicios web aplicados al ERP “ADRISERP” y su versión móvil en Android* (tesis de pre grado). Universidad Señor de Sipán, Chiclayo, Perú.
- Bush, T. (16 de julio de 2019). ¿ What is the difference between web services and APIs? [Mensaje en un blog]. Recuperado de <https://nordicapis.com/what-is-the-difference-between-web-services-and-apis/>
- Castillo Asencio, P. L. (2016). *Desarrollo e implementación de un sistema web para generar valor en una Pyme aplicando metodología ágil*. Lima.
- Castro, M., Sánchez, D., Farfán, J., Castro, D., Cándido, A., & Vargas, A. Aplicación de Servicios Web SOAP/ REST para funcionalidades existentes en sistemas informáticos provinciales. *7mo Simposio Argentino De Informática*. Simposio llevado a cabo en el Estado - SIE 2013.
- Cechák, D. (2017). *Using GraphQL for Content Delivery in Kentico Cloud* (Bachelor's Thesis). Masaryk University, Faculty of Informatics, República de Checa.
- Chandra, S., & Kumar, S. (2009). *An Introduction to Client /Server Computing: New Age International (P) Ltd., Publishers*.
- Codecademy. (2019). www.codecademy.com/. Obtenido de What is CRUD?: <https://www.codecademy.com/articles/what-is-crud>
- Eeda, N. (2018). *Rendering real-time dashboards using a GraphQL-based UI Architecture* (Master's Thesis). The University of Western Ontario, Canada.
- Eizinger, T. (2017). *API Design in Distributed Systems: A Comparison between GraphQL and REST* (Master's Thesis). University of Applied Sciences Technikum Wien - Degree Program Software Engineering, Austria.

- Fielding et al. (1999). Hypertext Transfer Protocol -- HTTP/1.1. Recuperado de <https://www.w3.org/Protocols/rfc2616/rfc2616.html>
- Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (doctoral thesis), University of California Irvine.
- Fredrich, T. (2012). RESTful Service Best Practices. Recommendations for Creating Web Services. Recuperado de <https://s3.amazonaws.com/tfpearsoncollege/bestpractices/RESTful+Best+Practices.pdf>
- Gustavsson, K., & Stenlund, E. (2016). *Efficient data communication between a webclient and a cloud environment* (Master's Thesis). Department of Electrical and Information Technology, Faculty of Engineering, LTH, Lund University.
- Haupt, F., Leymann, F., Scherer, A., & Vukojevic-Haupt, K. (2017). A Framework for the Structural Analysis of REST APIs. Recuperado de <https://www.iaas.uni-stuttgart.de/publications/INPROC-2017-08-A-Framework-for-the-Structural-Analysis-of-REST-APIs.pdf>
- Helgason, A. (2017). *Performance analysis of Web Services Comparison between RESTful & GraphQL web services* (Bachelor's Thesis). University of Skövde, School of Informatics, Suecia.
- International Organization for Standardization (2019). ISO/ IEC 25010. Recuperado de <http://www.iso25000.com/index.php/normas-iso-25000/iso-25010>
- Kennet, R. (2012). *Essential Scrum*. Michigan: Pearson Education, Inc.
- Kumar, D. (2018). Best practices for building restful web services. Recuperado de <https://www.infosys.com/digital/insights/Documents/restful-web-services.pdf>

López, F. (2015). Sistemas distribuidos. Recuperado de http://dccd.cua.uam.mx/libros/archivos/03IXStream_sistemas_distribuidos.pdf

NETCRAFT. (2018). *Encuesta de servidores Web*. Londres.

Papazoglou, M. (2008). Web services. Principles and Technology. Recuperado de <http://www.nortonaudio.com/Ficheiros/Web.services...principles.and.technology.pdf>

Pasma, T. (2018). *Report on web API Capacity Study A comparison of REST and GraphQL*. Recuperado de http://cs.au.dk/fileadmin/site_files/cs/EVU/Tjip_Pasma_-_WACS-Final-Tjip.pdf

Poniatowicz, T. (25 de setiembre de 2019). GraphQL vs REST: Performance. GraphQL Editor. Recuperado de <https://dev.to/graphqleditor/graphql-vs-rest-performance-38jb>

PROMPTBYTES. (05 de junio de 2019). <https://web.whatsapp.com/>. Obtenido de REST VS GraphQL: ¿qué se debe utilizar para desarrollar una API web y por qué?: <https://www.promptbytes.com/blog/rest-vs-graphql-web-api-development>

Ritsila, A. (2017). *GraphQL: The API Design Revolution* (Bachelor's Thesis). Haaga-Helia University of Applied Sciences, Finlandia.

Semere, E. (2017). *Transformation of REST API to GraphQL for OpenTOSCA* (Master's Thesis). Institute of Architecture of Application Systems, University of Stuttgart, Alemania.

Singh, R., & Kumar, S. (2016). International Journal of Advanced Research in Computer Science and Software Engineering. *International Journal of Advanced Research in Computer Science and Software Engineering*, 6(5), pp.1-4.

Statista (2019). Annual number of global mobile app downloads 2017-2021. *Number of mobile app downloads worldwide in 2017, 2018 and 2022 (in billions)*. statista.com. Recuperado de <https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/>.

Writtern, E., Ying, A., Zheng, Y., Laredo, J., Dolby, J., Young, C., & Slominski, A. (18 de mayo de 2017). Opportunities in Software Engineering Research for Web API Consumption. *ArXiv*. Recuperado de <https://arxiv.org/pdf/1705.06586.pdf>

ANEXOS

ANEXO N° 1: FORMATO DE RECOPIACIÓN DE DATOS

Objetivo: Recopilar los datos, identificar las Características y especificaciones de las tecnologías REST y GRAPHQL.

Características y Especificaciones técnicas de REST y GRAPHQL

Especificación- Característica	REST	GRAPHQL
Arquitectura Cliente- Servidor	SI	SI
Multiplataforma	SI	SI
Arquitectura de código libre	SI	SI
Lenguajes de programación que soporta	POO	POO
Protocolo sobre el cual Operan	HTTP	HTTP
Consultas múltiples en un End Point	NO	SI
Sobrecarga de datos	SI	NO
Seguridad en solicitudes arbitrarias	SI	SI

Fuente: Elaboración propia



ANEXO N° 2: GUÍA DE OBSERVACIÓN

Objetivo: Identificar y documentar los valores o lecturas obtenidos de la experimentación en cada una de las tecnologías evaluadas.

Guía de Observación de los End Point realizados para REST y GRAPHQL

TECNOLOGIA	END POINT	MÉTODO	DATOS A RECUPERAR	N° PETICIONES
API REST	http://imos.us-east-1.elasticbeanstalk.com/	QUERY	CLIENTE	10
			PRODUCTO	10
			VENTA	10
		MUTATION	CLIENTE	10
			PRODUCTO	10
			VENTA	10
API GRAPHQL	https://tranquil-chamber-83237.herokuapp.com/graphql/	GET	CLIENTE	10
			PRODUCTO	10
			VENTA	10
		POST	CLIENTE	10
			PRODUCTO	10
			VENTA	10

Fuente: Elaboración Propia



ANEXO N° 3: Código fuente de requerimientos

- Registrar clientes

```

- public class RegClienteFragment extends Fragment {
-
-     @BindView(R.id.btnRegistrar)
-     Button btnRegistrar;
-     @BindView(R.id.txtNombres)
-     EditText txtNombres;
-     @BindView(R.id.txtDireccion)
-     EditText txtDireccion;
-     @BindView(R.id.txtCorreo)
-     EditText txtCorreo;
-     @BindView(R.id.txtUsername)
-     EditText txtUsername;
-     ProgressDialog progressDialog;
-     Cliente clienteM;
-     String accion;
-     private Session session;
-     private String urlBaseRest = "";
-     private String urlBaseGraphql = "";
-
-     public static RegClienteFragment newInstance(Bundle arguments, Cliente
cliente){
-         RegClienteFragment f = new RegClienteFragment();
-         if(arguments != null){
-             f.setClienteM(cliente);
-             f.setArguments(arguments);
-         }
-         return f;
-     }
-
-     @Override
-     public View onCreateView(LayoutInflater inflater, ViewGroup container
, Bundle savedInstanceState) {
-         View rootView = inflater.inflate(R.layout.fragment_reg_cliente, c
ontainer, false);
-         ButterKnife.bind(this, rootView);
-         Bundle parametros = getArguments();
-         accion = (String)parametros.get("accion");
-         getActivity().setTitle(accion.equals("M") ? "Modificar Cliente":
"Registrar Cliente");
-     }
- }

```

```

-     FloatingActionButton fabButtonAdd = (FloatingActionButton) getActivity().findViewById(R.id.fab);
-     session = new Session(getContext());
-     urlBaseRest = session.getApiRest();
-     urlBaseGraphQL = session.getApiGraphQL();
-     if(accion.equals("M")){
-         btnRegistrar.setText("MODIFICAR");
-         txtNombres.setText(clienteM.getNombres());
-         txtDireccion.setText(clienteM.getDireccion());
-         txtCorreo.setText(clienteM.getEmail());
-         txtUsername.setText(clienteM.getTelefono());
-     }
-     fabButtonAdd.setImageResource(R.drawable.ic_menu_send);
-     fabButtonAdd.setOnClickListener(new View.OnClickListener() {
-         @Override
-         public void onClick(View view) {
-             viewAtras();
-         }
-     });
-     try {
-         btnRegistrar.setOnClickListener(new View.OnClickListener(){
-             @Override
-             public void onClick(View v){
-                 RegistrarRecord(v);
-             }
-         });
-     }catch(Exception e){
-         Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHORT).show();
-     }
-     return rootView;
- }

-     public static long getUsedMemorySize() {
-         long freeSize = 0L;
-         long totalSize = 0L;
-         long usedSize = -1L;
-         try {
-             Runtime info = Runtime.getRuntime();
-             freeSize = info.freeMemory();
-             totalSize = info.totalMemory();
-             usedSize = totalSize - freeSize;
-         } catch (Exception e) {

```

```

-         e.printStackTrace();
-     }
-     return usedSize;
- }

-     public void RegistrarRecord(View v){
-         try{
-             Cliente cliente = new Cliente(clienteM != null ? clienteM.getIdcliente() : 0);
-             cliente.setNombres(Metodos.parsingEmptyUppercase(txtNombres.getText().toString()));
-             cliente.setDireccion(Metodos.parsingEmptyUppercase(txtDireccion.getText().toString()));
-             cliente.setEmail(Metodos.parsingEmpty(txtCorreo.getText().toString()));
-             cliente.setTelefono(Metodos.parsingEmpty(txtUsername.getText().toString()));

-             List<String> errores = validarData(cliente);
-             validate(v,errores);
-             if(errores.isEmpty()){
-                 progressDialog = new ProgressDialog(getActivity(), R.style.MyAlertDialogStyle);
-                 progressDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);

-                 progressDialog.setIndeterminate(true);
-                 progressDialog.setCancelable(false);
-                 progressDialog.setCanceledOnTouchOutside(false);
-                 progressDialog.show();

-                 String tip = session.getTipo();

-                 if(tip.equals("REST")) {
-                     ApiRetrofit retrofit = new ApiRetrofit();
-                     ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-                     Call<Respuesta> call = clienteM != null ? api.ModificarCliente(cliente) : api.RegistrarCliente(cliente);
-                     call.enqueue(new retrofit2.Callback<Respuesta>() {
-                         @Override
-                         public void onResponse(Call<Respuesta> call, Response<Respuesta> response) {
-                             Respuesta respuesta = response.body();
-                             if (respuesta.getDato().equals("OK")){

```

```

-         long tx = response.raw().sentRequestAtMillis();
-         long rx = response.raw().receivedResponse
AtMillis());
-         Integer dif = Integer.parseInt(String.val
ueOf(rx - tx));
-
-         DataRespuesta dataRespuesta = new DataRespuesta();
-         dataRespuesta.setTiempo(dif);
-         dataRespuesta.setIndicador("TIEMPO");
-         dataRespuesta.setInterfaz("CLIENTE");
-         dataRespuesta.setMetodo("POST");
-         dataRespuesta.setEstado("1");
-         dataRespuesta.setTipo("REST");
-
-         Long memoria = getUsedMemorySize()/(1024*
1024); // media en KB
-
-         DataRespuesta dataRespuestaRam = new Data
Respuesta();
-         dataRespuestaRam.setEstado("1");
-         dataRespuestaRam.setMetodo("POST");
-         dataRespuestaRam.setIndicador("RAM");
-         dataRespuestaRam.setTipo("REST");
-         dataRespuestaRam.setInterfaz("CLIENTE");
-         dataRespuestaRam.setTiempo(memoria.intValue());
-
-         GuardarTiempoRespuesta(dataRespuesta,data
RespuestaRam);
-         } else {
-             progressDialog.dismiss();
-             Toast.makeText(getContext(), respuesta.ge
tMensaje(), Toast.LENGTH_SHORT).show();
-         }
-     }
-     @Override
-     public void onFailure(Call<Respuesta> call, Throw
able t) {
-         progressDialog.dismiss();
-         Toast.makeText(getContext(), "PROBLEMAS CON E
L SERVIDOR", Toast.LENGTH_SHORT).show();
-     }
- });
-     } else if(tip.equals("GRAPHQL")){

```

```

-         Date inicio = new Date();
-
-         if(clienteM == null) {
-             ApiApollo.getApolloClientWithURL(this.urlBaseGrap
- hql).mutate(MantClienteMutation.builder()
-             .nombres(cliente.getNombres())
-             .direccion(cliente.getDireccion())
-             .email(cliente.getEmail())
-             .telefono(cliente.getTelefono()).build())
-             .enqueue(new ApolloCall.Callback<MantClienteMutat
- ion.Data>() {
-                 @Override
-                 public void onResponse(@NotNull com.apollogra
- phql.apollo.api.Response<MantClienteMutation.Data> response) {
-                     getActivity().runOnUiThread(new Runnable() {
-                         @Override
-                         public void run() {
-                             Date fin = new Date();
-                             long tie = fin.getTime() - inicio.getTime();
-
-                             Integer dif = Integer.parseInt(St
- ring.valueOf(tie));
-
-                             DataRespuesta dataRespuesta = new
- DataRespuesta();
-
-                             dataRespuesta.setEstado("1");
-                             dataRespuesta.setIndicador("TIEMPO");
-                             dataRespuesta.setMetodo("MUTATION");
-                             dataRespuesta.setTipo("GRAPHQL");
-                             dataRespuesta.setInterfaz("CLIENTE");
-                             dataRespuesta.setTiempo(dif);
-
-                             Long memoria = getUsedMemorySize(
- )/(1024*1024); // media en KB
-
-                             DataRespuesta dataRespuestaRam =
- new DataRespuesta();
-
-                             dataRespuestaRam.setEstado("1");
-                             dataRespuestaRam.setIndicador("RAM");
-                             dataRespuestaRam.setMetodo("MUTATION");
-                             dataRespuestaRam.setTipo("GRAPHQL");
-                             dataRespuestaRam.setInterfaz("CLIENTE");

```

```

-                                     dataRespuestaRam.setTiempo(memori
a.intValue());
-
-                                     GuardarTiempoRespuesta(dataRespu
sta,dataRespuestaRam);
-                                     }
-                                     });
-                                     }
-                                     @Override
-                                     public void onFailure(@NotNull ApolloException e) {
-                                     progressDialog.dismiss();
-                                     Toast.makeText(getContext(), "PROBLEMAS C
ON EL SERVIDOR", Toast.LENGTH_SHORT).show();
-                                     }
-                                     });
-                                     } else {
-                                     ApiApollo.getApolloClientWithURL(this.urlBaseGrap
hql).mutate(MantUpdClienteMutation.builder()
-                                     .idcliente(cliente.getIdcliente())
-                                     .nombres(cliente.getNombres())
-                                     .direccion(cliente.getDireccion())
-                                     .email(cliente.getEmail())
-                                     .telefono(cliente.getTelefono()).build())
-                                     .enqueue(new ApolloCall.Callback<MantUpdClienteMu
tation.Data>() {
-                                     @Override
-                                     public void onResponse(@NotNull com.apollograp
hql.apollo.api.Response<MantUpdClienteMutation.Data> response) {
-                                     getActivity().runOnUiThread(new Runnable() {
-                                     @Override
-                                     public void run() {
-                                     Date fin = new Date();
-                                     long tie = fin.getTime() - inicio.getTime();
-                                     Integer dif = Integer.parseInt(St
ring.valueOf(tie));
-                                     DataRespuesta dataRespuesta = new
DataRespuesta();
-                                     dataRespuesta.setEstado("1");
-                                     dataRespuesta.setIndicador("TIEMPO");
-                                     dataRespuesta.setMetodo("MUTATION");
-                                     dataRespuesta.setTipo("GRAPHQL");
-                                     dataRespuesta.setInterfaz("CLIENTE");
-                                     dataRespuesta.setTiempo(dif);

```

```

-
-                                     Long memoria = getUsedMemorySize(
-                                     )/(1024*1024); // media en KB
-
-                                     DataRespuesta dataRespuestaRam =
new DataRespuesta();
-                                     dataRespuestaRam.setEstado("1");
-                                     dataRespuestaRam.setIndicador("RAM");
-                                     dataRespuestaRam.setMetodo("MUTATION");
-                                     dataRespuestaRam.setTipo("GRAPHQL");
-                                     dataRespuestaRam.setInterfaz("CLIENTE");
-                                     dataRespuestaRam.setTiempo(memori
a.intValue());
-
-                                     GuardarTiempoRespuesta(dataRespue
sta,dataRespuestaRam);
-                                     }
-                                     });
-                                     }
-                                     @Override
-                                     public void onFailure(@NotNull ApolloException e) {
-                                     progressDialog.dismiss();
-                                     Toast.makeText(getContext(), "PROBLEMAS C
ON EL SERVIDOR", Toast.LENGTH_SHORT).show();
-                                     }
-                                     });
-                                     }
-                                     }
-                                     }
-                                     }catch(Exception e){
-                                     Toast.makeText(getActivity(),e.getMessage(),Toast.LENGTH_SHOR
T).show();
-                                     }
-                                     }
-
-                                     public void GuardarTiempoRespuesta(DataRespuesta tiempoDataRespuesta,
DataRespuesta ramDataRespuesta){
-                                     try{
-                                     ApiRetrofit retrofit = new ApiRetrofit();
-                                     ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-                                     Call<Respuesta> call = api.RegistrarTiempoRespuesta(tiempoData
Respuesta);
-                                     call.enqueue(new retrofit2.Callback<Respuesta>(){

```



```

-         @Override
-         public void onResponse(Call<Respuesta> call, final Respon
se<Respuesta> response) {
-             GuardarConsumoMemoria(ramDataRespuesta);
-         }
-         @Override
-         public void onFailure(Call<Respuesta> call, Throwable t) {
-             progressDialog.dismiss();
-             Toast.makeText(getContext(), "PROBLEMAS CON EL SERVID
OR", Toast.LENGTH_SHORT).show();
-         }
-     });
- }catch (Exception e){
-     progressDialog.dismiss();
-     Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHOR
T).show();
- }
- }

-     public void GuardarConsumoMemoria(DataRespuesta tiempoDataRespuesta){
-         try{
-             ApiRetrofit retrofit = new ApiRetrofit();
-             ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-             Call<Respuesta> call = api.RegistrarTiempoRespuesta(tiempoData
Respuesta);
-             call.enqueue(new retrofit2.Callback<Respuesta>(){
-                 @Override
-                 public void onResponse(Call<Respuesta> call, final Respon
se<Respuesta> response) {
-                     Toast.makeText(getContext(),"Cliente Registrado Corre
ctamente.", Toast.LENGTH_SHORT).show();
-                     progressDialog.dismiss();
-                     viewAtras();
-                 }
-                 @Override
-                 public void onFailure(Call<Respuesta> call, Throwable t) {
-                     progressDialog.dismiss();
-                     Toast.makeText(getContext(), "PROBLEMAS CON EL SERVID
OR", Toast.LENGTH_SHORT).show();
-                 }
-             });
-         }catch (Exception e){
-             progressDialog.dismiss();

```

```

-         Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHOR
T).show();
-     }
- }
-
-     public void viewAtras(){
-         Fragment fragment = new ClienteFragment();
-         FragmentManager fragmentManager = getActivity().getSupportFragmen
tManager();
-         fragmentManager.beginTransaction().replace(R.id.frag_inicio, frag
ment).commit();
-     }
-
-     public void validate(View view,List<String> errores) {
-         for (int i=0;i<errores.size();i++){
-             if(errores.get(i).equals("E1")){txtNombres.setError("INGRESE
SUS NOMBRES");}
-             if(errores.get(i).equals("E2")){txtDireccion.setError("INGRES
E DIRECCIÓN");}
-             if(errores.get(i).equals("E3")){txtCorreo.setError("INGRESE S
US CORREO");}
-             if(errores.get(i).equals("E4")){txtCorreo.setError("CORREO IN
VALIDO");}
-             if(errores.get(i).equals("E5")){txtUsername.setError("TELEFON
O INCORRECTO");}
-             if(errores.get(i).equals("E6")){txtUsername.setError("TELEFON
O MAYOR A 30 DIGITOS");}
-         }
-         if(!errores.isEmpty()){
-             if(errores.get(0).equals("E1")){txtNombres.requestFocus();}
-             if(errores.get(0).equals("E2")){txtDireccion.requestFocus();}
-             if(errores.get(0).equals("E3")){txtCorreo.requestFocus();}
-             if(errores.get(0).equals("E4")){txtCorreo.requestFocus();}
-             if(errores.get(0).equals("E5")){txtUsername.requestFocus();}
-             if(errores.get(0).equals("E6")){txtUsername.requestFocus();}
-         }
-     }
-
-     private List<String> validarData(Cliente objeto){
-         List<String> errores = new ArrayList<String>();
-         if(objeto.getNombres() == null){
-             errores.add("E1");
-         }else if(objeto.getNombres().equals("")){

```

```

-         errores.add("E1");
-     }
-     if(objeto.getDireccion() == null){
-         errores.add("E2");
-     }else if(objeto.getDireccion().equals("")){
-         errores.add("E2");
-     }
-     if(objeto.getEmail() == null){
-         errores.add("E3");
-     }else if(objeto.getEmail().equals("")){
-         errores.add("E3");
-     }else if(!Metodos.isValidEmailAddress(objeto.getEmail())){
-         errores.add("E4");
-     }
-     if(objeto.getTelefono() == null){
-         errores.add("E5");
-     }else if(objeto.getTelefono().equals("")){
-         errores.add("E5");
-     }else if(objeto.getTelefono().length() > 30){
-         errores.add("E6");
-     }
-     return errores;
- }
-
- public void setClienteM(Cliente clienteM) {
-     this.clienteM = clienteM;
- }
-
- }

```

- **Listado de clientes**

```

- public class ClienteFragment extends Fragment {
-
-     private FloatingActionButton fabButtonAdd = null;
-     @BindView(R.id.listusuarios)
-     ListView listView;
-     @BindView(R.id.barraUsuarios)
-     ProgressBar progressBar;
-     private ClienteAdapterList adapter;
-     private Session session;
-     private FloatingActionsMenu fabMenu;
-     private String urlBaseRest = "";

```

```

-     private String urlBaseGraphQL = "";
-
-     public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup
- container, Bundle savedInstanceState) {
-         View root = inflater.inflate(R.layout.fragment_home, container, f
-         else);
-         ButterKnife.bind(this,root);
-         getActivity().setTitle("Listado de Clientes");
-         session = new Session(getContext());
-         urlBaseRest = session.getApiRest();
-         urlBaseGraphQL = session.getApiGraphQL();
-         fabButtonAdd = (FloatingActionButton) getActivity().findViewById(
-         R.id.fab);
-         fabButtonAdd.setImageResource(R.drawable.ic_add_white_48dp);
-         fabMenu = (FloatingActionsMenu) getActivity().findViewById(R.id.f
-         abMenu);
-         fabMenu.setVisibility(View.GONE);
-         ListarViewUsuarios();
-         fabButtonAdd.setOnClickListener(new View.OnClickListener(){
-             @Override
-             public void onClick(View view){
-                 CrearDialogo();
-             }
-         });
-         fabButtonAdd.show();
-         listView.setOnItemClickListener(new AdapterView.OnItemClickListener()
-         er() {
-             @Override
-             public void onItemClick(AdapterView<?> parent, final View vie
-             w, final int position, long id) {
-                 PopupMenu menu = new PopupMenu(getContext(), view);
-                 menu.getMenu().add(Menu.NONE, 1, 1, "Modificar");
-                 menu.getMenu().add(Menu.NONE, 2, 2, "Eliminar");
-                 menu.show();
-                 menu.setOnMenuItemClickListener(new PopupMenu.OnMenuItemC
-                 lickListener(){
-                     @Override
-                     public boolean onOptionsItemSelected(MenuItem item) {
-                         int i = item.getItemId();
-                         if (i == 1) {
-                             CargarCliente(view,position);
-                             return false;
-                         } else if (i == 2) {

```

```

-         EliminarCliente(view,position);
-         return false;
-     } else {
-         return false;
-     }
-     }
-     });
-     });
-     return root;
- }

- private float readUsage() {
-     try {
-         RandomAccessFile reader = new RandomAccessFile("/proc/stat",
- "r");
-         String load = reader.readLine();
-         String[] toks = load.split(" "); // Split on one or more spaces
-
-         long idle1 = Long.parseLong(toks[4]);
-         long cpu1 = Long.parseLong(toks[2]) + Long.parseLong(toks[3])
+ Long.parseLong(toks[5])
-             + Long.parseLong(toks[6]) + Long.parseLong(toks[7]) +
- Long.parseLong(toks[8]);
-
-         try {
-             Thread.sleep(360);
-         } catch (Exception e) {}
-
-         reader.seek(0);
-         load = reader.readLine();
-         reader.close();
-
-         toks = load.split(" ");
-
-         long idle2 = Long.parseLong(toks[4]);
-         long cpu2 = Long.parseLong(toks[2]) + Long.parseLong(toks[3])
+ Long.parseLong(toks[5])
-             + Long.parseLong(toks[6]) + Long.parseLong(toks[7]) +
- Long.parseLong(toks[8]);
-
-         return (float)(cpu2 - cpu1) / ((cpu2 + idle2) - (cpu1 + idle1));
-     }
- }

```

```

-     } catch (IOException ex) {
-         ex.printStackTrace();
-     }
-
-     return 0;
- }
-
- public static long getUsedMemorySize() {
-     long freeSize = 0L;
-     long totalSize = 0L;
-     long usedSize = -1L;
-     try {
-         Runtime info = Runtime.getRuntime();
-         freeSize = info.freeMemory();
-         totalSize = info.totalMemory();
-         usedSize = totalSize - freeSize;
-     } catch (Exception e) {
-         e.printStackTrace();
-     }
-     return usedSize;
- }
-
- public void EliminarCliente(View v,final int posicion){
-     final Cliente clienteEli = adapter.getClientes().get(posicion);
-     AlertDialog.Builder builder = new AlertDialog.Builder(getActivity
- ());
-     builder.setTitle(R.string.app_name);
-     builder.setMessage("¿Esta seguro en eliminar el cliente "+cliente
- Eli.getNombres()+"?");
-     builder.setIcon(R.drawable.ic_delete_forever_black_24dp);
-     builder.setPositiveButton("Si", new DialogInterface.OnClickListener
- er() {
-         public void onClick(final DialogInterface dialog, int id) {
-             progressBar.setVisibility(View.VISIBLE);
-             ApiRetrofit retrofit = new ApiRetrofit();
-             ApiRest api = retrofit.getApiWithURL(urlBaseRest);
-             Call<Respuesta> call = api.BorrarCliente(clienteEli.getId
- cliente());
-             call.enqueue(new retrofit2.Callback<Respuesta>() {
-                 @Override
-                 public void onResponse(Call<Respuesta> call, final Re
- sponse<Respuesta> response) {
-                     Respuesta datos = response.body();

```

```

-         if(datos.getDato().equals("OK")) {
-             Toast.makeText(getContext(),"Cliente eliminad
o correctamente.",Toast.LENGTH_SHORT).show();
-             List<Cliente> clientes = adapter.getClientes();
-             clientes.remove(posicion);
-             adapter.updateRecords(clientes);
-             dialog.dismiss();
-             progressBar.setVisibility(View.GONE);
-         }else{
-             progressBar.setVisibility(View.GONE);
-             dialog.dismiss();
-             Toast.makeText(getContext(),datos.getMensaje(
),Toast.LENGTH_SHORT).show();
-         }
-     }
-     @Override
-     public void onFailure(Call<Respuesta> call, Throwable t) {
-         progressBar.setVisibility(View.GONE);
-         dialog.dismiss();
-         Toast.makeText(getContext(), "PROBLEMAS CON EL SE
RVIDOR" + t.getMessage(), Toast.LENGTH_SHORT).show();
-     }
-     });
-     //fabButtonAdd.setVisibility(View.VISIBLE);
- }
- });
- builder.setNegativeButton("No", new DialogInterface.OnClickListen
er() {
-     public void onClick(DialogInterface dialog, int id) {
-         dialog.dismiss();
-     }
- });
- AlertDialog alert = builder.create();
- alert.show();
- }

-     public void CargarCliente(View v, final int posicion){
-         Bundle bundle = new Bundle();
-         bundle.putString("accion","M");
-         RegClienteFragment fragment = RegClienteFragment.newInstance(bundle,adapter.getClientes().get(posicion));
-         FragmentManager fragmentManager = getActivity().getSupportFragmen
tManager();

```

```
-     fragmentManager.beginTransaction().replace(R.id.frag_inicio, frag  
ment).commit();  
-     }  
  
-     public void CrearDialogo() {  
-         Bundle bundle = new Bundle();  
-         bundle.putString("accion", "R");  
-         RegClienteFragment fragment = RegClienteFragment.newInstance(bundle,  
null);  
-         fragmentManager = getActivity().getSupportFragmentManager();  
-         fragmentManager.beginTransaction().replace(R.id.frag_inicio, frag  
ment).commit();  
-     }  
  
-     public void ListarViewCLIENTES(){  
-         try{  
-             String tip = session.getTipo();  
-             progressBar.setVisibility(View.VISIBLE);  
  
-             if(tip.equals("REST")) {  
-                 ApiRetrofit retrofit = new ApiRetrofit();  
-                 ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);  
-                 Map<String, String> data = new HashMap<String, String>();  
-                 Call<List<Cliente>> call = api.ListarClientes(data);  
-                 call.enqueue(new retrofit2.Callback<List<Cliente>>(){  
-                     @Override  
-                     public void onResponse(Call<List<Cliente>> call, fina  
l Response<List<Cliente>> response) {  
-                         List<Cliente> lista = response.body();  
-                         long tx = response.raw().sentRequestAtMillis();  
-                         long rx = response.raw().receivedResponseAtMillis();  
-                         Integer dif = Integer.parseInt(String.valueOf(rx  
- tx));  
  
-                         DataRespuesta dataRespuesta = new DataRespuesta();  
-                         dataRespuesta.setEstado("1");  
-                         dataRespuesta.setMetodo("GET");  
-                         dataRespuesta.setTipo("REST");  
-                         dataRespuesta.setInterfaz("CLIENTE");  
-                         dataRespuesta.setIndicador("TIEMPO");  
-                         dataRespuesta.setTiempo(dif);  
-                     }  
-                 });  
-             }  
-         }  
-     }  
- }
```



```

-                                     RecargarUsuarios(lista);
-
-                                     Long memoria = getUsedMemorySize()/(1024*1024); /
- / media en KB
-
-                                     DataRespuesta dataRespuestaRam = new DataRespuesta();
-                                     dataRespuestaRam.setEstado("1");
-                                     dataRespuestaRam.setMetodo("GET");
-                                     dataRespuestaRam.setIndicador("RAM");
-                                     dataRespuestaRam.setTipo("REST");
-                                     dataRespuestaRam.setInterfaz("CLIENTE");
-                                     dataRespuestaRam.setTiempo(memoria.intValue());
-
-                                     GuardarTiempoRespuesta(dataRespuesta,dataRespuestaRam);
-                                     }
-                                     @Override
-                                     public void onFailure(Call<List<Cliente>> call, Thrown
- able t) {
-                                     progressBar.setVisibility(View.GONE);
-                                     Toast.makeText(getContext(), "PROBLEMAS CON EL SE
- RVIDOR", Toast.LENGTH_SHORT).show();
-                                     }
-                                     });
-                                     } else {
-                                     Date inicio = new Date();
-                                     ApiApollo.getApolloClientWithURL(urlBaseGraphQL).query(CL
- ientesQuery.builder().build()).
-                                     enqueue(new ApolloCall.Callback<ClientesQuery.Data>() {
-                                     @Override
-                                     public void onStatusEvent(@NotNull ApolloCall.StatusE
- vent event) {
-                                     if (event == ApolloCall.StatusEvent.COMPLETED) {
-
-                                     }
-                                     }
-                                     @Override
-                                     public void onResponse(com.apollographql.apollo.api.R
- esponse<ClientesQuery.Data> response ) {
-                                     getActivity().runOnUiThread(new Runnable() {
-                                     @Override public void run() {
-                                     Date fin = new Date();
-                                     long tie = fin.getTime() - inicio.getTime();

```

```

-                                     List<Cliente> lista = ParsearResultApollo
Cliente(response.data().clientes());
-                                     RecargarUsuarios(lista);
-
-                                     Integer dif = Integer.parseInt( String.va
lueOf(tie) ) ;
-
-                                     DataRespuesta dataRespuesta = new DataRespuesta();
-                                     dataRespuesta.setEstado("1");
-                                     dataRespuesta.setIndicador("TIEMPO");
-                                     dataRespuesta.setMetodo("QUERY");
-                                     dataRespuesta.setTipo("GRAPHQL");
-                                     dataRespuesta.setInterfaz("CLIENTE");
-                                     dataRespuesta.setTiempo(dif);
-
-                                     Long memoria = getUsedMemorySize()/(1024*
1024); // media en KB
-
-                                     DataRespuesta dataRespuestaRam = new Data
Respuesta();
-                                     dataRespuestaRam.setEstado("1");
-                                     dataRespuestaRam.setIndicador("RAM");
-                                     dataRespuestaRam.setMetodo("QUERY");
-                                     dataRespuestaRam.setTipo("GRAPHQL");
-                                     dataRespuestaRam.setInterfaz("CLIENTE");
-                                     dataRespuestaRam.setTiempo(memoria.intValue());
-
-                                     GuardarTiempoRespuesta(dataRespuesta,data
RespuestaRam);
-                                     }
-                                     });
-                                     }
-                                     @Override
-                                     public void onFailure(ApolloException e) {
-                                     progressBar.setVisibility(View.GONE);
-                                     Toast.makeText(getContext(),e.getMessage(), Toast
.LENGTH_SHORT).show();
-                                     }
-                                     });
-                                     }
-                                     } catch (Exception e){
-                                     progressBar.setVisibility(View.GONE);

```

```

-         Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHOR
- T).show();
-     }
- }
-
-     public void RecargarUsuarios(List<Cliente> clientes){
-         adapter = new ClienteAdapterList(getContext(),clientes,getActivit
- y());
-         listView.setAdapter(adapter);
-         progressBar.setVisibility(View.GONE);
-     }
-
-     public void GuardarConsumoMemoria(DataRespuesta tiempoDataRespuesta){
-         try{
-             progressBar.setVisibility(View.VISIBLE);
-             ApiRetrofit retrofit = new ApiRetrofit();
-             ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-             Call<Respuesta> call = api.RegistrarTiempoRespuesta(tiempoData
- Respuesta);
-             call.enqueue(new retrofit2.Callback<Respuesta>(){
-                 @Override
-                 public void onResponse(Call<Respuesta> call, final Respon
- se<Respuesta> response) {
-                     progressBar.setVisibility(View.GONE);
-                 }
-                 @Override
-                 public void onFailure(Call<Respuesta> call, Throwable t) {
-                     progressBar.setVisibility(View.GONE);
-                     Toast.makeText(getContext(), "PROBLEMAS CON EL SERVID
- OR", Toast.LENGTH_SHORT).show();
-                 }
-             });
-         }catch (Exception e){
-             progressBar.setVisibility(View.GONE);
-             Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHOR
- T).show();
-         }
-     }
-
-     public void GuardarTiempoRespuesta(DataRespuesta tiempoDataRespuesta,
- DataRespuesta tiempoDataRespuestaRam){
-         try{
-             progressBar.setVisibility(View.VISIBLE);

```

```

-         ApiRetrofit retrofit = new ApiRetrofit();
-         ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-         Call<Respuesta> call = api.RegistrarTiempoRespuesta(tiempoData
Respuesta);
-         call.enqueue(new retrofit2.Callback<Respuesta>(){
-             @Override
-                 public void onResponse(Call<Respuesta> call, final Respon
se<Respuesta> response) {
-                     GuardarConsumoMemoria(tiempoDataRespuestaRam);
-                 }
-             @Override
-                 public void onFailure(Call<Respuesta> call, Throwable t) {
-                     progressBar.setVisibility(View.GONE);
-                     Toast.makeText(getContext(), "PROBLEMAS CON EL SERVID
OR", Toast.LENGTH_SHORT).show();
-                 }
-             });
-         } catch (Exception e){
-             progressBar.setVisibility(View.GONE);
-             Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHOR
T).show();
-         }
-     }
-
-     public List<Cliente> ParsearResultApolloCliente(List<ClientesQuery.Cl
iente> datos){
-         List<Cliente> listado = new ArrayList<Cliente>();
-
-         for(int i=0;i<datos.size();i++){
-             Cliente cliente = new Cliente();
-             cliente.setIdcliente(Integer.parseInt(datos.get(i).idcliente(
)))
-             cliente.setNombres(datos.get(i).nombres());
-             cliente.setTelefono(datos.get(i).telefono());
-             cliente.setEmail(datos.get(i).email());
-             cliente.setDireccion(datos.get(i).direccion());
-             listado.add(cliente);
-         }
-         return listado;
-     }
- }

```


- Registrar productos

```

- public void RegistrarRecord(View v){
-     try{
-         Producto producto = new Producto(productoM != null ? producto
M.getIdproducto() : 0);
-         producto.setNombre(Metodos.parsingEmptyUppercase(txtNombres.g
etText().toString()));
-         producto.setPrecioventa(Metodos.VerificarNumeroDecimal(txtDir
eccion.getText().toString()) ? Double.parseDouble(txtDireccion.getText().
toString()) : null);
-         producto.setPreciocompra(Metodos.VerificarNumeroDecimal(txtCo
rreo.getText().toString()) ? Double.parseDouble(txtCorreo.getText().ToStr
ing()) : null);
-         producto.setStock(Metodos.VerificarNumero(txtUsername.getText
().toString()) ? Integer.parseInt(txtUsername.getText().toString()) : nul
l);
-         producto.setEstado(Metodos.parsingEmpty(seleccionado.equals("
ACTIVO") ? "1" : "0"));
-
-         List<String> errores = validarData(producto);
-         validate(v,errores);
-         if(errores.isEmpty()){
-             progressDialog = new ProgressDialog(getActivity(), R.styl
e.MyAlertDialogStyle);
-             progressDialog.setProgressStyle(ProgressDialog.STYLE_SPIN
NER);
-             progressDialog.setIndeterminate(true);
-             progressDialog.setCancelable(false);
-             progressDialog.setCanceledOnTouchOutside(false);
-             progressDialog.show();
-
-             String tip = session.getTipo();
-
-             if(tip.equals("REST")) {
-                 ApiRetrofit retrofit = new ApiRetrofit();
-                 ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-
-                 Call<Respuesta> call = productoM != null ? api.Modifi
carProducto(producto) : api.RegistrarProducto(producto);
-                 call.enqueue(new retrofit2.Callback<Respuesta>() {
-                     @Override
-                     public void onResponse(Call<Respuesta> call, Resp
onse<Respuesta> response) {

```

```

-         Respuesta respuesta = response.body();
-         if (respuesta.getDato().equals("OK")) {
-             long tx = response.raw().sentRequestAtMillis();
-                 long rx = response.raw().receivedResponse
AtMillis();
-                 Integer dif = Integer.parseInt(String.val
ueOf(rx - tx));
-
-                 DataRespuesta dataRespuesta = new DataRespuesta();
-                 dataRespuesta.setIdrespuesta(0);
-                 dataRespuesta.setTiempo(dif);
-                 dataRespuesta.setIndicador("TIEMPO");
-                 dataRespuesta.setInterfaz("PRODUCTO");
-                 dataRespuesta.setMetodo("POST");
-                 dataRespuesta.setEstado("1");
-                 dataRespuesta.setTipo("REST");
-
-                 Long memoria = getUsedMemorySize()/(1024*
1024); // media en KB
-
-                 DataRespuesta dataRespuestaRam = new Data
Respuesta();
-                 dataRespuestaRam.setEstado("1");
-                 dataRespuestaRam.setMetodo("POST");
-                 dataRespuestaRam.setIndicador("RAM");
-                 dataRespuestaRam.setTipo("REST");
-                 dataRespuestaRam.setInterfaz("PRODUCTO");
-                 dataRespuestaRam.setTiempo(memoria.intValue());
-
-                 GuardarTiempoRespuesta(dataRespuesta,data
RespuestaRam);
-
-                 } else {
-                     progressDialog.dismiss();
-                     Toast.makeText(getContext(), respuesta.ge
tMensaje(), Toast.LENGTH_SHORT).show();
-                 }
-             }
-             @Override
-             public void onFailure(Call<Respuesta> call, Throw
able t) {
-                 progressDialog.dismiss();
-                 Toast.makeText(getContext(), "PROBLEMAS CON E
L SERVIDOR", Toast.LENGTH_SHORT).show();

```

```

-     }
-     });
-     } else if(tip.equals("GRAPHQL")) {
-         Date inicio = new Date();
-         if(productoM == null)
-         {
-             ApiApollo.getApolloClientWithURL(this.urlBaseGrap
- hql).mutate(MantProductoMutation.builder()
-             .nombre(producto.getNombre())
-             .preciocompra(producto.getPreciocompra())
-             .precioventa(producto.getPrecioventa())
-             .estado(producto.getEstado())
-             .stock(producto.getStock())
-             .build()
-             .enqueue(new ApolloCall.Callback<MantProductoMuta
- tion.Data>() {
-                 @Override
-                 public void onResponse(@NotNull com.apollogra
- phql.apollo.api.Response<MantProductoMutation.Data> response) {
-                     getActivity().runOnUiThread(new Runnable() {
-                         @Override
-                         public void run() {
-                             Date fin = new Date();
-                             long tie = fin.getTime() - inicio.getTime();
-
-                             Integer dif = Integer.parseInt(St
- ring.valueOf(tie));
-
-                             DataRespuesta dataRespuesta = new
- DataRespuesta();
-
-                             dataRespuesta.setEstado("1");
-                             dataRespuesta.setIndicador("TIEMPO");
-                             dataRespuesta.setMetodo("MUTATION");
-                             dataRespuesta.setTipo("GRAPHQL");
-                             dataRespuesta.setInterfaz("PRODUCTO");
-                             dataRespuesta.setTiempo(dif);
-
-                             Long memoria = getUsedMemorySize(
- )/(1024*1024); // media en KB
-
-                             DataRespuesta dataRespuestaRam =
- new DataRespuesta();
-                             dataRespuestaRam.setEstado("1");

```



```

-         dataRespuestaRam.setIndicador("RAM");
-         dataRespuestaRam.setMetodo("MUTATION");
-         dataRespuestaRam.setTipo("GRAPHQL");
-         dataRespuestaRam.setInterfaz("PRODUCTO");
-         dataRespuestaRam.setTiempo(memori
a.intValue());
-
-         GuardarTiempoRespuesta(dataRespu
sta,dataRespuestaRam);
-     }
- });
- }
- @Override
- public void onFailure(@NotNull ApolloException e) {
-     progressDialog.dismiss();
-     Toast.makeText(getContext(), "PROBLEMAS C
ON EL SERVIDOR", Toast.LENGTH_SHORT).show();
- }
- });
- } else {
-     ApiApollo.getApolloClientWithURL(this.urlBaseGrap
hql).mutate(MantUpdProductoMutation.builder()
-         .idproducto(producto.getIdproducto())
-         .nombre(producto.getNombre())
-         .preciocompra(producto.getPreciocompra())
-         .precioventa(producto.getPrecioventa())
-         .stock(producto.getStock())
-         .estado(producto.getEstado()).build())
-         .enqueue(new ApolloCall.Callback<MantUpdProductoM
utation.Data>() {
-             @Override
-             public void onResponse(@NotNull com.apollogra
phql.apollo.api.Response<MantUpdProductoMutation.Data> response) {
-                 getActivity().runOnUiThread(new Runnable() {
-                     @Override
-                     public void run() {
-                         Date fin = new Date();
-                         long tie = fin.getTime() - inicio.getTime();
-
-                         Integer dif = Integer.parseInt(St
ring.valueOf(tie));
-

```

```

-                                     DataRespuesta dataRespuesta = new
DataRespuesta();
-                                     dataRespuesta.setEstado("1");
-                                     dataRespuesta.setIndicador("TIEMPO");
-                                     dataRespuesta.setMetodo("MUTATION");
-                                     dataRespuesta.setTipo("GRAPHQL");
-                                     dataRespuesta.setInterfaz("PRODUCTO");
-                                     dataRespuesta.setTiempo(dif);
-
-                                     Long memoria = getUsedMemorySize(
-                                     )/(1024*1024); // media en KB
-
-                                     DataRespuesta dataRespuestaRam =
new DataRespuesta();
-                                     dataRespuestaRam.setEstado("1");
-                                     dataRespuestaRam.setIndicador("RAM");
-                                     dataRespuestaRam.setMetodo("MUTATION");
-                                     dataRespuestaRam.setTipo("GRAPHQL");
-                                     dataRespuestaRam.setInterfaz("PRODUCTO");
-                                     dataRespuestaRam.setTiempo(memori
a.intValue());
-
-                                     GuardarTiempoRespuesta(dataRespu
sta,dataRespuestaRam);
-                                     }
-                                     });
-                                     }
-                                     @Override
-                                     public void onFailure(@NotNull ApolloException e) {
-                                     progressDialog.dismiss();
-                                     Toast.makeText(getContext(), "PROBLEMAS C
ON EL SERVIDOR", Toast.LENGTH_SHORT).show();
-                                     }
-                                     });
-                                     }
-                                     }
-                                     }
-                                     }catch(Exception e){
-                                     Toast.makeText(getActivity(),e.getMessage(),Toast.LENGTH_SHOR
T).show();
-                                     }
-                                     }

```

- **Listado de productos**

```

- public void ListarViewPRODUCTOS(){
-     try{
-         String tip = session.getTipo();
-         progressBar.setVisibility(View.VISIBLE);
-
-         if(tip.equals("REST")){
-             progressBar.setVisibility(View.VISIBLE);
-             ApiRetrofit retrofit = new ApiRetrofit();
-             ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-             Map<String, String> data = new HashMap<String,String>();
-             Call<List<Producto>> call = api.ListarProductos(data);
-             call.enqueue(new retrofit2.Callback<List<Producto>>(){
-                 @Override
-                 public void onResponse(Call<List<Producto>> call, final Response<List<Producto>> response) {
-                     List<Producto> lista = response.body();
-
-                     long tx = response.raw().sentRequestAtMillis();
-                     long rx = response.raw().receivedResponseAtMillis();
-                     Integer dif = Integer.parseInt( String.valueOf(rx
- - tx) ) ;
-
-                     DataRespuesta dataRespuesta = new DataRespuesta();
-                     dataRespuesta.setEstado("1");
-                     dataRespuesta.setIndicador("TIEMPO");
-                     dataRespuesta.setMetodo("GET");
-                     dataRespuesta.setTipo("REST");
-                     dataRespuesta.setInterfaz("PRODUCTO");
-                     dataRespuesta.setTiempo(dif);
-
-                     RecargarUsuarios(lista);
-
-                     Long memoria = getUsedMemorySize()/(1024*1024); /
- / media en KB
-
-                     DataRespuesta dataRespuestaRam = new DataRespuesta();
-                     dataRespuestaRam.setEstado("1");
-                     dataRespuestaRam.setMetodo("GET");
-                     dataRespuestaRam.setIndicador("RAM");
-                     dataRespuestaRam.setTipo("REST");
-                     dataRespuestaRam.setInterfaz("PRODUCTO");
-                     dataRespuestaRam.setTiempo(memoria.intValue());

```

```

-         GuardarTiempoRespuesta(dataRespuesta,dataRespuestaRam);
-     }
-     @Override
-     public void onFailure(Call<List<Producto>> call, Thro
wable t) {
-         progressBar.setVisibility(View.GONE);
-         Toast.makeText(getContext(), "PROBLEMAS CON EL SE
RVIDOR", Toast.LENGTH_SHORT).show();
-     }
- });
- } else {
-     Date inicio = new Date();
-     ApiApollo.getApolloClientWithURL(this.urlBaseGraphQL).que
ry(ClientesQuery.builder().build()).
-     enqueue(new ApolloCall.Callback<ClientesQuery.Data>() {
-         @Override
-         public void onResponse(com.apollographql.apollo.api.R
esponse<ClientesQuery.Data> response ) {
-             getActivity().runOnUiThread(new Runnable() {
-                 @Override public void run() {
-                     Date fin = new Date();
-
-                     long tie = fin.getTime() - inicio.getTime();
-
-                     List<Producto> lista = ParsearResultApoll
oProducto(response.data().productos());
-                     RecargarUsuarios(lista);
-
-                     Integer dif = Integer.parseInt( String.va
lueOf(tie) ) ;
-
-                     DataRespuesta dataRespuesta = new DataRespuesta();
-                     dataRespuesta.setEstado("1");
-                     dataRespuesta.setIndicador("TIEMPO");
-                     dataRespuesta.setMetodo("QUERY");
-                     dataRespuesta.setTipo("GRAPHQL");
-                     dataRespuesta.setInterfaz("PRODUCTO");
-                     dataRespuesta.setTiempo(dif);
-
-                     Long memoria = getUsedMemorySize()/(1024*
1024); // media en KB

```

```

-                                     DataRespuesta dataRespuestaRam = new Data
Respuesta();
-                                     dataRespuestaRam.setEstado("1");
-                                     dataRespuestaRam.setIndicador("RAM");
-                                     dataRespuestaRam.setMetodo("QUERY");
-                                     dataRespuestaRam.setTipo("GRAPHQL");
-                                     dataRespuestaRam.setInterfaz("PRODUCTO");
-                                     dataRespuestaRam.setTiempo(memoria.intValue());
-
-                                     GuardarTiempoRespuesta(dataRespuesta,data
RespuestaRam);
-                                     }
-                                     });
-                                     }
-                                     @Override
-                                     public void onFailure(ApolloException e) {
-                                     System.out.println("error2");
-                                     System.out.println(e.getMessage());
-                                     }
-                                     });
-                                     }
-                                     }catch (Exception e){
-                                     progressBar.setVisibility(View.GONE);
-                                     Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHOR
T).show();
-                                     }
-                                     }

```

- **Registrar pedidos**

```

-     public void RegistrarVenta(View v){
-         try{
-             progressDialog = new ProgressDialog(getActivity(), R.style.My
AlertDialogStyle);
-             progressDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
-             progressDialog.setIndeterminate(true);
-             progressDialog.setCancelable(false);
-             progressDialog.setCanceledOnTouchOutside(false);
-             progressDialog.show();
-
-             String tip = session.getTipo();
-             ApiRetrofit retrofit = new ApiRetrofit();
-             ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);

```

```

-         Call<Respuesta> call = api.RegistrarVenta(venta);
-         call.enqueue(new retrofit2.Callback<Respuesta>() {
-             @Override
-             public void onResponse(Call<Respuesta> call, Response<Res
- respuesta> response) {
-                 Respuesta respuesta = response.body();
-                 if(respuesta.getDatos().equals("OK")){
-                     long tx = response.raw().sentRequestAtMillis();
-                     long rx = response.raw().receivedResponseAtMillis();
-                     Integer dif = Integer.parseInt(String.valueOf(rx - tx));
-
-                     DataRespuesta dataRespuesta = new DataRespuesta();
-                     dataRespuesta.setInterfaz("VENTA");
-                     dataRespuesta.setIndicador("TIEMPO");
-                     if(tip.equals("REST")){
- CONECTARAPIREST();
-                         dataRespuesta.setMetodo("POST");
-                     } else {
- CONECTARAPIGRAPHQL();
-                         dataRespuesta.setMetodo("MUTATION");
-                     }
-                     dataRespuesta.setEstado("1");
-                     dataRespuesta.setTipo(tip);
-                     dataRespuesta.setTiempo(dif);
-
-                     Long memoria = getUsedMemorySize()/(1024*1024); /
- / media en KB
-
-                     DataRespuesta dataRespuestaRam = new DataRespuesta();
-                     dataRespuestaRam.setEstado("1");
-                     dataRespuestaRam.setIndicador("RAM");
-                     if(tip.equals("REST")){
-                         dataRespuestaRam.setMetodo("POST");
-                     } else {
-                         dataRespuestaRam.setMetodo("MUTATION");
-                     }
-                     dataRespuestaRam.setTipo(tip);
-                     dataRespuestaRam.setInterfaz("VENTA");
-                     dataRespuestaRam.setTiempo(memoria.intValue());
-
-                     GuardarTiempoRespuesta(dataRespuesta,dataRespuestaRam);
-                 }else{
-                     progressDialog.dismiss();

```

```

-         Toast.makeText(getApplicationContext(), respuesta.getMensaje(
- ), Toast.LENGTH_SHORT).show();
-     }
- }
- @Override
- public void onFailure(Call<Respuesta> call, Throwable t) {
-     progressDialog.dismiss();
-     Toast.makeText(getApplicationContext(), "PROBLEMAS CON EL SERVIDO
- R", Toast.LENGTH_SHORT).show();
- }
- });
- }catch(Exception e){
-     Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHOR
- T).show();
- }
- }

-     public void GuardarTiempoRespuesta(DataRespuesta tiempoDataRespuesta,
- DataRespuesta ramdataRespuesta){
-         try{
-             ApiRetrofit retrofit = new ApiRetrofit();
-             ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-             Call<Respuesta> call = api.RegistrarTiempoRespuesta(tiempoData
- Respuesta);
-             call.enqueue(new retrofit2.Callback<Respuesta>(){
-                 @Override
-                 public void onResponse(Call<Respuesta> call, final Respon
- se<Respuesta> response) {
-                     GuardarConsumoMemoria(ramdataRespuesta);
-                 }
-                 @Override
-                 public void onFailure(Call<Respuesta> call, Throwable t) {
-                     progressDialog.dismiss();
-                     Toast.makeText(getApplicationContext(), "PROBLEMAS CON EL SERVID
- OR", Toast.LENGTH_SHORT).show();
-                 }
-             });
-         }catch (Exception e){
-             progressDialog.dismiss();
-             Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHOR
- T).show();
-         }
-     }
- }

```

```

- public void GuardarConsumoMemoria(DataRespuesta tiempoDataRespuesta){
-     try{
-         ApiRetrofit retrofit = new ApiRetrofit();
-         ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-         Call<Respuesta> call = api.RegistrarTiempoRespuesta(tiempoData
Respuesta);
-         call.enqueue(new retrofit2.Callback<Respuesta>(){
-             @Override
-             public void onResponse(Call<Respuesta> call, final Respon
se<Respuesta> response) {
-                 Toast.makeText(getContext(),"Venta registrada correct
amente.", Toast.LENGTH_SHORT).show();
-                 progressDialog.dismiss();
-                 ReiniciarCampos();
-             }
-             @Override
-             public void onFailure(Call<Respuesta> call, Throwable t) {
-                 progressDialog.dismiss();
-                 Toast.makeText(getContext(), "PROBLEMAS CON EL SERVID
OR", Toast.LENGTH_SHORT).show();
-             }
-         });
-     }catch (Exception e){
-         progressDialog.dismiss();
-         Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHOR
T).show();
-     }
- }

```

- **Listado de pedidos**

```

- public void ListarVentas(){
-     try{
-         String tip = session.getTipo();
-         progressBar.setVisibility(View.VISIBLE);
-
-         if(tip.equals("REST")){
-             ApiRetrofit retrofit = new ApiRetrofit();
-             ApiRest api = retrofit.getApiWithURL(this.urlBaseRest);
-             Map<String, String> data = new HashMap<String,String>();
-             Call<List<Venta>> call = api.ListarVentas(data);
-             call.enqueue(new retrofit2.Callback<List<Venta>>(){
-                 @Override

```



```

-         public void onResponse(Call<List<Venta>> call, final
Response<List<Venta>> response) {
-             List<Venta> lista = response.body();
-
-             long tx = response.raw().sentRequestAtMillis();
-             long rx = response.raw().receivedResponseAtMillis();
-             Integer dif = Integer.parseInt( String.valueOf(rx
- tx) );
-
-             DataRespuesta dataRespuesta = new DataRespuesta();
-             dataRespuesta.setEstado("1");
-             dataRespuesta.setIndicador("TIEMPO");
-             dataRespuesta.setMetodo("GET");
-             dataRespuesta.setTipo("REST");
-             dataRespuesta.setInterfaz("VENTA");
-             dataRespuesta.setTiempo(dif);
-
-             RecargarDatosLista(lista);
-
-             Long memoria = getUsedMemorySize()/(1024*1024); /
/ media en KB
-
-             DataRespuesta dataRespuestaRam = new DataRespuesta();
-             dataRespuestaRam.setEstado("1");
-             dataRespuestaRam.setMetodo("GET");
-             dataRespuestaRam.setIndicador("RAM");
-             dataRespuestaRam.setTipo("REST");
-             dataRespuestaRam.setInterfaz("VENTA");
-             dataRespuestaRam.setTiempo(memoria.intValue());
-
-             GuardarTiempoRespuesta(dataRespuesta,dataRespuestaRam);
-         }
-         @Override
-         public void onFailure(Call<List<Venta>> call, Thrownab
le t) {
-             progressBar.setVisibility(View.GONE);
-             Toast.makeText(getContext(), "PROBLEMAS CON EL SE
RVIDOR", Toast.LENGTH_SHORT).show();
-         }
-     });
-     } else {
-         Date inicio = new Date();

```

```

-         ApiApollo.getApolloClientWithURL(this.urlBaseGraphQL).que
ry(ClientesQuery.builder().build()).
-         enqueue(new ApolloCall.Callback<ClientesQuery.Data>() {
-             @Override
-             public void onResponse(com.apollographql.apollo.api.R
response<ClientesQuery.Data> response ) {
-                 getActivity().runOnUiThread(new Runnable() {
-                     @Override public void run() {
-                         Date fin = new Date();
-                         long tie = fin.getTime() - inicio.getTime();
-
-                         List<Venta> lista = ParsearResultApolloVe
nta(response.data().ventas());
-                         RecargarDatosLista(lista);
-
-                         Integer dif = Integer.parseInt( String.va
lueOf(tie) ) ;
-
-                         DataRespuesta dataRespuesta = new DataRespuesta();
-                         dataRespuesta.setEstado("1");
-                         dataRespuesta.setIndicador("TIEMPO");
-                         dataRespuesta.setMetodo("QUERY");
-                         dataRespuesta.setTipo("GRAPHQL");
-                         dataRespuesta.setInterfaz("VENTA");
-                         dataRespuesta.setTiempo(dif);
-
-                         Long memoria = getUsedMemorySize()/(1024*
1024); // media en KB
-
-                         DataRespuesta dataRespuestaRam = new Data
Respuesta();
-                         dataRespuestaRam.setEstado("1");
-                         dataRespuestaRam.setIndicador("RAM");
-                         dataRespuestaRam.setMetodo("QUERY");
-                         dataRespuestaRam.setTipo("GRAPHQL");
-                         dataRespuestaRam.setInterfaz("VENTA");
-                         dataRespuestaRam.setTiempo(memoria.intValue());
-
-                         GuardarTiempoRespuesta(dataRespuesta,data
RespuestaRam);
-                     }
-                 });
-             }
-         });
-     }

```

```

-         @Override
-         public void onFailure(ApolloException e) {
-             progressBar.setVisibility(View.GONE);
-             Toast.makeText(getContext(),e.getMessage(), Toast
.LENGTH_SHORT).show();
-         }
-     });
- }
- }catch (Exception e){
-     progressBar.setVisibility(View.GONE);
-     Toast.makeText(getContext(),e.getMessage(), Toast.LENGTH_SHOR
T).show();
- }
- }
    
```

- **Listar y seleccionar entre las Apis REST y GRAPHQL.**

```

- public class ParametroFragment extends DialogFragment {
-
-     @BindView(R.id.btnRegistrar)
-     Button btnRegistrar;
-     @BindView(R.id.btnAtras)
-     Button btnAtras;
-     @BindView(R.id.txtRest)
-     EditText txtRest;
-     @BindView(R.id.txtGraphql)
-     EditText txtGraphql;
-     private Session session;
-
-     public static ParametroFragment newInstance(Bundle arguments) {
-         ParametroFragment f = new ParametroFragment();
-         if (arguments != null) {
-             f.setArguments(arguments);
-         }
-         return f;
-     }
-
-     public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup
- container, Bundle savedInstanceState) {
-         try {
-             View rootView = inflater.inflate(R.layout.fragment_parametros
- , container, false);
-             ButterKnife.bind(this, rootView);
-             setStyle(DialogFragment.STYLE_NORMAL,0);
-             setCancelable(true);
-             getDialog().setCanceledOnTouchOutside(false);
-             session = new Session(getContext());
-             String rest = Metodos.parsingEmpty(session.getApiRest());
-             String graphql = Metodos.parsingEmpty(session.getApiGraphQL());
-             txtRest.setText(rest != null ? rest : "");
-             txtGraphql.setText(graphql != null ? graphql : "");
-             btnRegistrar.setOnClickListener(new View.OnClickListener() {
-                 @Override
-                 public void onClick(View v) {
-                     session.GuardarApiRest(txtRest.getText().toString());
-                     session.GuardarApiGraphQL(txtGraphql.getText().toStri
- ng());
-                     getDialog().dismiss();
-                 }
-             }
-         }
-     }
- }

```

```

-         });
-         btnAtras.setOnClickListener(new View.OnClickListener() {
-             @Override
-             public void onClick(View v) {
-                 getDialog().dismiss();
-             }
-         });
-         return rootView;
-     } catch (Exception e) {
-         System.out.println("ERICK");
-         System.out.println(e.getMessage());
-         return null;
-     }
- }
- }

```

Rendimiento (Promedio y Comparativo)

```
private void CargarDatosGraficaReg(){
    Integer proRest = (Integer.parseInt(cantCliente.getText().toString()) +
        Integer.parseInt(cantProducto.getText().toString()+
        Integer.parseInt(cantVenta.getText().toString()))/3 ;

    Integer proGraphql = ( Integer.parseInt(cantClienteQL.getText().toSt
ring()) +
        Integer.parseInt(cantProductoQL.getText().toString()+
        Integer.parseInt(cantVentaQL.getText().toString()) )/3;

    ArrayList<BarEntry> entries = new ArrayList<>();

    entries.add(new BarEntry(proRest,0));
    entries.add(new BarEntry(proGraphql,1));

    BarDataSet barDataSet = new BarDataSet(entries,"APIS");

    ArrayList<String> labels = new ArrayList<String>();
    labels.add("REST");
    labels.add("GRAPHQL");

    BarData data = new BarData(labels,barDataSet);
    barChart.setData(data);
    barChart.setDescription("Comparativo en milisegundos.");

    barDataSet.setColors(ColorTemplate.COLORFUL_COLORS);

    barChart.animateY(2000);
}
```

Tiempo

```
private void CargarDatosGraficaReg(){
    Integer proRest = (Integer.parseInt(cantCliente.getText().toString()) +
        Integer.parseInt(cantProducto.getText().toString()+
        Integer.parseInt(cantVenta.getText().toString()))/3 ;

    Integer proGraphql = ( Integer.parseInt(cantClienteQL.getText().toSt
ring()) +
        Integer.parseInt(cantProductoQL.getText().toString()+
        Integer.parseInt(cantVentaQL.getText().toString()) )/3;

    ArrayList<BarEntry> entries = new ArrayList<>();

    entries.add(new BarEntry(proRest,0));
    entries.add(new BarEntry(proGraphql,1));

    BarDataSet barDataSet = new BarDataSet(entries,"APIS");

    ArrayList<String> labels = new ArrayList<String>();
    labels.add("REST");
    labels.add("GRAPHQL");

    BarData data = new BarData(labels,barDataSet);
    barChart.setData(data);
    barChart.setDescription("Comparativo en milisegundos.");

    barDataSet.setColors(ColorTemplate.COLORFUL_COLORS);

    barChart.animateY(2000);
}
```

ANEXO N° 4: Código fuente base de datos

Código fuente de base de datos

```

create table producto
(
    idproducto serial,
    nombre character varying(100) not null,
    preciocompra numeric(9,2) not null,
    precioventa numeric(9,2) not null,
    stock integer not null default 0,
    estado character varying(1) not null,
    constraint pk_producto primary key(idproducto),
    constraint chk_precom_producto check (preciocompra >= 0.00),
    constraint chk_preven_producto check(precioventa > 0.00),
    constraint chk_estado_producto check (estado in ('0','1')),
    constraint chk_stock_producto check(stock >= 0),
    constraint chk_nombre_producto unique(nombre)
);

create table cliente
(
    idcliente serial,
    nombres character varying(200) not null,
    direccion character varying(200) null,
    telefono character varying(20) null,
    email character varying(80) null,
    constraint pk_cliente primary key (idcliente)
);

create table venta
(
    idventa serial,
    numeroventa character varying(8) not null,
    fechaventa date not null,
    idcliente integer not null,
    total numeric(9,2) not null,
    estadoventa character varying(1) not null,
    constraint pk_venta primary key(idventa),
    constraint fk_venta_cliente foreign key(idcliente) references cliente (
idcliente),
    constraint chk_total_venta check (total > 0.00)
);

```



```

create table detalleventa
(
    iddetalleventa serial,
    idventa integer not null,
    idproducto integer not null,
    preunicompra numeric(9,2) not null,
    cantidad integer not null,
    preciounitario numeric(9,2) not null,
    preciototal numeric(9,2) not null,
    estadodetalle character varying(1) not null,
    constraint pk_detalleventa primary key(iddetalleventa),
    constraint fk_detalleventa_venta foreign key(idventa) references venta (
idventa),
    constraint fk_detalleventa_producto foreign key(idproducto) references p
roducto ( idproducto),
    constraint chk_cantidad check ( cantidad > 0 ),
    constraint chk_preciounitario check ( preciounitario > 0),
    constraint chk_preciototal check ( preciototal > 0),
    constraint chk_preciocompra_det check (preunicompra >= 0.00)
);

create table datarespuesta
(
    iddatarespuesta serial,
    tipo character varying(12) not null,
    metodo character varying(20) not null,
    tiempo integer not null,
    interfaz character varying(25) not null,
    indicador character varying(20) not null,
    estado character varying(1) not null,
    constraint pk_datarespuesta primary key(iddatarespuesta),
    constraint chk_tiempo check ( tiempo > 0),
    constraint chk_estado_respuesta check (estado in ('1','0'))
);

```

ANEXO N° 8: Guía completa de interfaces de las APIS REST y GRAPHQL

1.- Introducción.

En este entregable se explicará el funcionamiento e instalación de una API REST y una API GRAPHQL, además como realizar múltiples consultas hacia una base de datos POSTGRES mediante las APIS usando como cliente el navegador web (GOOGLE CHROME).

2.- Modelo Entidad-Relación de la base de datos.

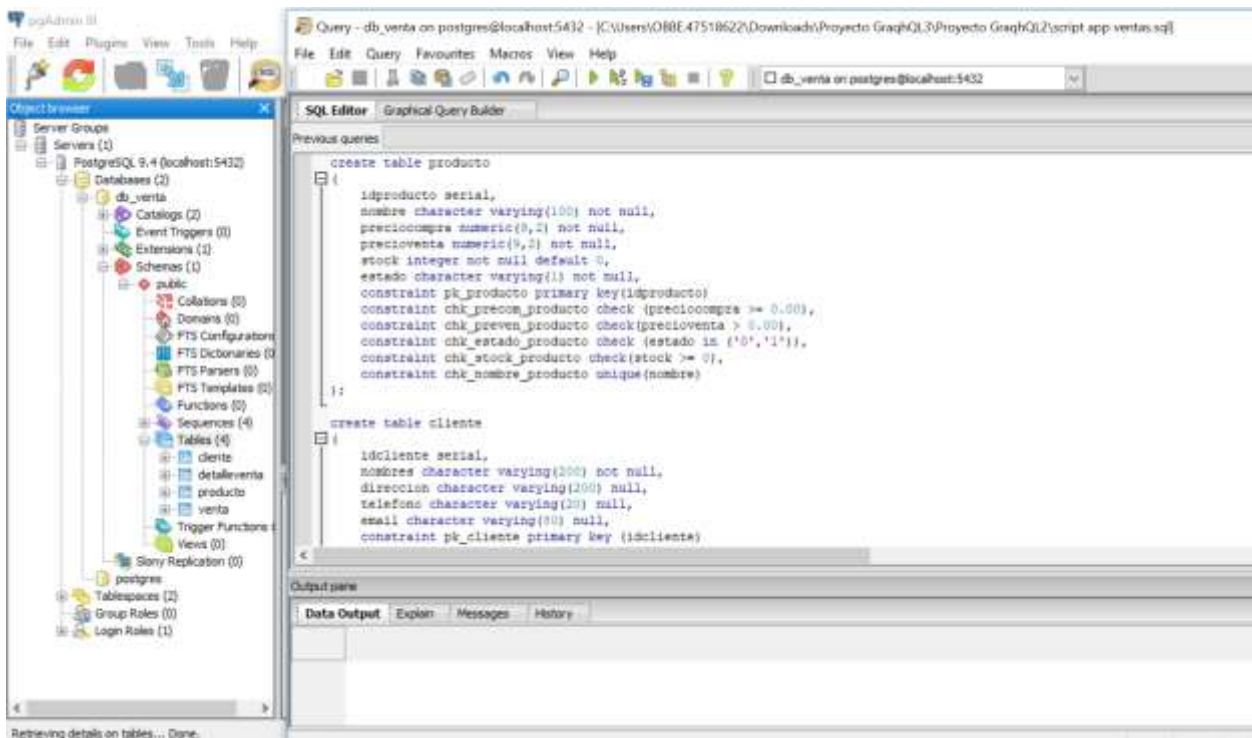
2.1.- Instalación del motor de base de datos.

Ingresa a la siguiente URL <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> y descarga el instalador que se adecue a las características de su equipo.

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
12.1	N/A	N/A	Download	Download	N/A
11.6	N/A	N/A	Download	Download	N/A
10.11	Download	Download	Download	Download	Download
9.6.16	Download	Download	Download	Download	Download
9.5.20	Download	Download	Download	Download	Download
9.4.25	Download	Download	Download	Download	Download
9.3.25 (Not Supported)	Download	Download	Download	Download	Download

2.2.- Script de base de datos.

Se detalla los comandos SQL para crear el diseño.



3.- API REST

3.1.- Instalación de servidor web.

The screenshot shows the Apache Tomcat 8 Software Downloads page. At the top left is the Apache Tomcat logo (a yellow cat) and the text 'Apache Tomcat®'. At the top right is the Apache Software Foundation logo. Below the logo is a search bar with 'GO' and a 'Save the date!' link. The main content area is titled 'Tomcat 8 Software Downloads' and contains the following text:

Welcome to the Apache Tomcat® 8.x software download page. This page provides download links for obtaining the latest versions of Tomcat 8.x software, as well as links to the archives of older releases.

Users of Tomcat 8.0.x should be aware that it has reached [end of life](#). Users of Tomcat 8.0.x should upgrade to 8.5.x or later.

Note: End of life has been announced for 8.0.x only. 8.5.x is not affected by this announcement.

Quick Navigation

[KEYS](#) | [8.5.42](#) | [Browse](#) | [Archives](#)

Release Integrity

You **must verify** the integrity of the downloaded files. We provide OpenPGP signatures for every release file. This signature should be matched against the [KEYS](#) file which contains the OpenPGP keys of Tomcat's Release Managers. We also provide [SHA-512](#) checksums for every release file. After you download the file, you should calculate a checksum for your download, and make sure it is the same as ours.

Mirrors

You are currently using <https://www-eu.apache.org/dist/>. If you encounter a problem with this mirror, please select another mirror. If all mirrors are failing, there are *backup* mirrors (at the end of the mirrors list) that should be available.

Other mirrors:

The left sidebar contains the following sections:

- Apache Tomcat**
 - Home
 - Tagfile
 - Maven Plugin
- Download**
 - Which version?
 - Tomcat 9
 - Tomcat 8
 - Tomcat 7
 - Tomcat Connectors
 - Tomcat Native
 - Tagfile
 - Archives
- Documentation**
 - Tomcat 9.0
 - Tomcat 8.5
 - Tomcat 7.0

3.2.- Ejecución del API REST.

Una vez instalado apache tomcat v8.x.x, se despliega la aplicación sobre dicho servidor web y se aloja en el puerto 8080.

Definición de API REST

API significa Interfaz de Programación de Aplicaciones. Una Interfaz de Programación de Aplicaciones te permite establecer una conexión o enlace entre 2 tipos de software diferentes. Una API facilita el intercambio de datos entre 2 tipos diferentes de software. Estás utilizando diariamente APIs, por ejemplo cuando nos logueamos a través de la cuenta de google o via Facebook estamos utilizando APIs. Por lo tanto, las APIs permiten a los desarrolladores usar contenido o características de una app, servicio o plataforma diferente en un servicio, plataforma o app propia de una manera segura.

Listad de EndPoints

- [Listar Clientes](#)
- [Listar Productos](#)
- [Listar Ventas](#)

3.3.- Comunicación con el API REST desde el navegador web.

Las respuestas contienen todos los valores de todos los atributos de la entidad solicitada, de los cuales solo usare algunos.

```
localhost:8080/ListaClientes
[{"idcliente":1,"nombres":"ERICK","direccion":"AV BOLOGNESI","telefono":"192923","email":"bernilla@hotmail.com"}, {"idcliente":2,"nombres":"BERNILLA","direccion":"CALLE JULIO C. TELLO 125","telefono":"1929292","email":"erick21_91@hotmail.com"}]
```

```
localhost:8080/ListaProductos
[{"idproducto":1,"nombre":"DASBOGA","preciocompra":10.0,"precioventa":20.0,"stock":5,"estado":"1"}, {"idproducto":2,"nombre":"CAMISA","preciocompra":20.0,"precioventa":50.0,"stock":4,"estado":"1"}]
```

```
localhost:8080/ListaVentas
[{"idventa":1,"numeroventa":"P-000001","fechaventa":"2019-10-10","cliente":{"idcliente":1,"nombres":"ERICK","direccion":"AV BOLOGNESI","telefono":"192923","email":"bernilla@hotmail.com"},"total":100.0,"estadoventa":"1"}, {"idventa":2,"numeroventa":"P-000002","fechaventa":"2019-11-10","cliente":{"idcliente":2,"nombres":"BERNILLA","direccion":"CALLE JULIO C. TELLO 125","telefono":"1929292","email":"erick21_91@hotmail.com"},"total":100.0,"estadoventa":"1"}]
```

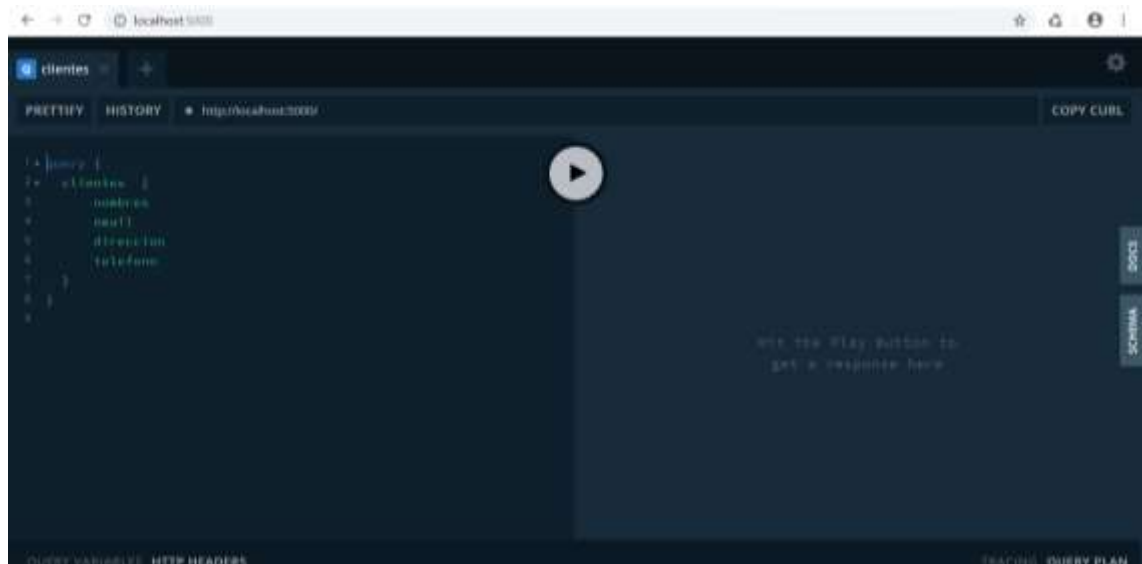
4.- API GRAPHQL

4.1.- Instalación de NODEJS.

Descargar e instalar, luego ingrese a la carpeta del proyecto GRAPHQL y ejecute el comando npm install, espere a que se instalen todas las dependencias del proyecto y luego inicie la aplicación con el comando npm start.



4.2.- Ejecución del API GRAPHQL.



4.2.- Comunicación con el API GRAPHQL desde el navegador web.

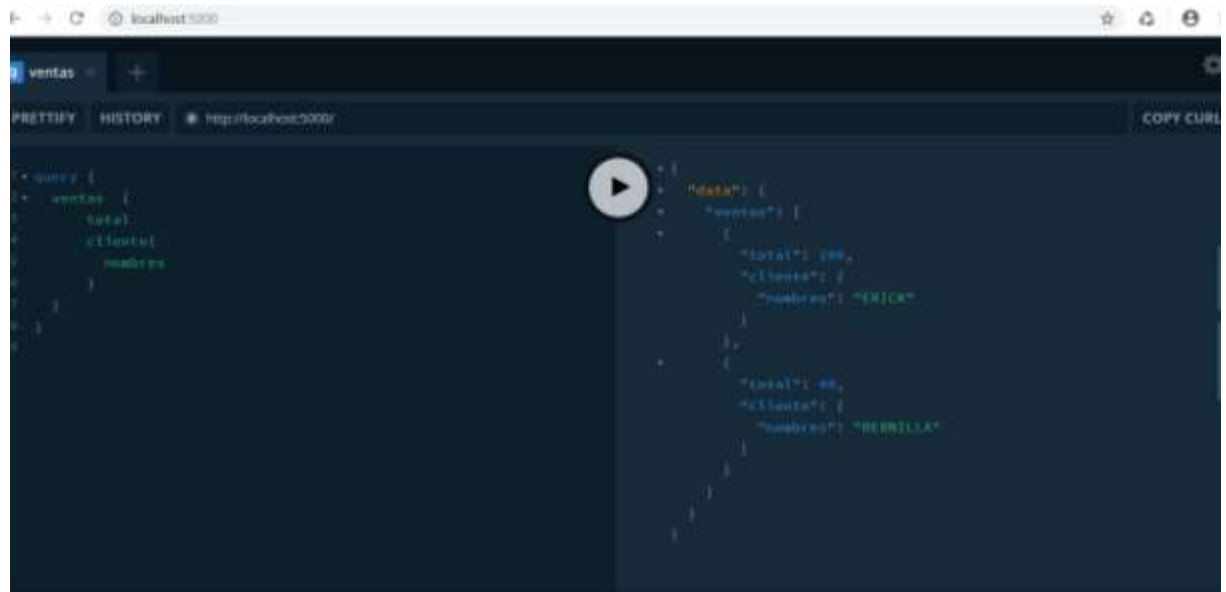
El beneficio de esta tecnología es que puede especificar qué datos necesito, así evito obtener datos que quedaran en el aire, además que la velocidad de mi respuesta es mucho mayor a REST, ya que me llegan menos datos.

```
GET http://localhost:5000/clientes
```

```
{
  "data": [
    {
      "cliente": {
        "nombre": "BERNARDO",
        "email": "bernard@hotaxil.com",
        "direccion": "Av. BOLSOBARI"
      }
    },
    {
      "cliente": {
        "nombre": "BERNILLA",
        "email": "bernilla32@hotaxil.com",
        "direccion": "CALLE BOLSO C. TELLO 125"
      }
    }
  ]
}
```

```
GET http://localhost:5000/productos
```

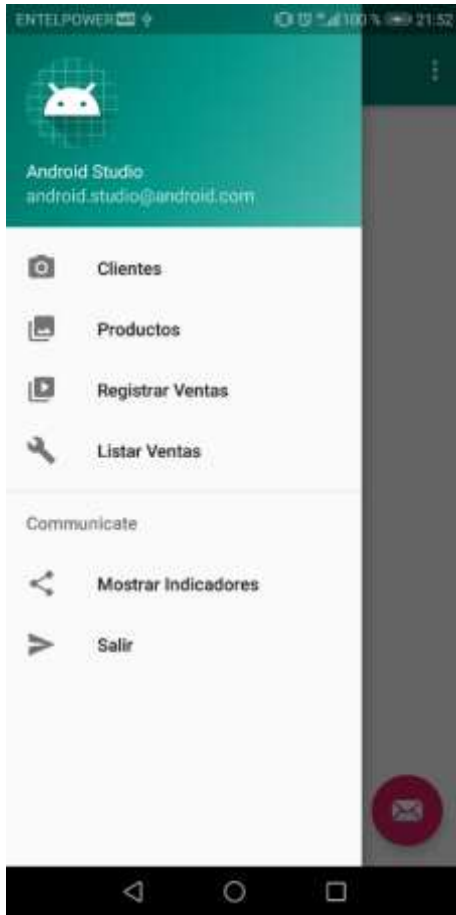
```
{
  "data": [
    {
      "producto": {
        "nombre": "CAMISA",
        "precio_venta": 30
      }
    },
    {
      "producto": {
        "nombre": "PANTALON",
        "precio_venta": 10
      }
    },
    {
      "producto": {
        "nombre": "ZAPATOS",
        "precio_venta": 40
      }
    },
    {
      "producto": {
        "nombre": "BOLSO",
        "precio_venta": 20
      }
    }
  ]
}
```

5. Módulos que componen la API.

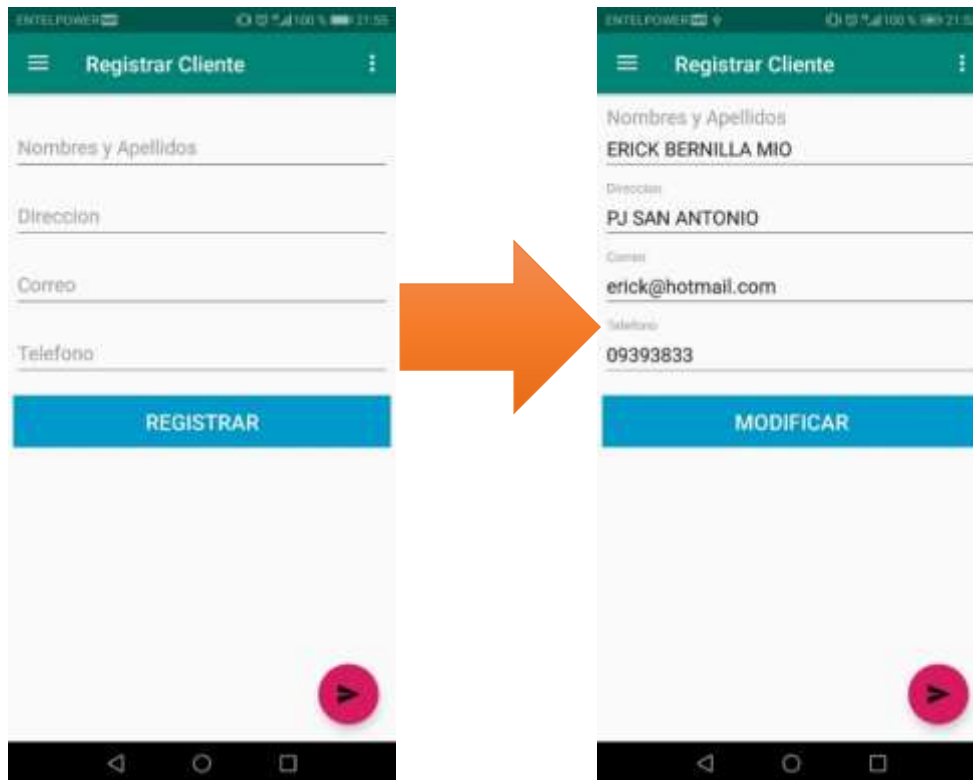
Se muestran las pantallas de los módulos propios de la aplicación, entre ellos están los módulos de clientes, productos y ventas.

Se nos mostrará un menú con las siguientes opciones:

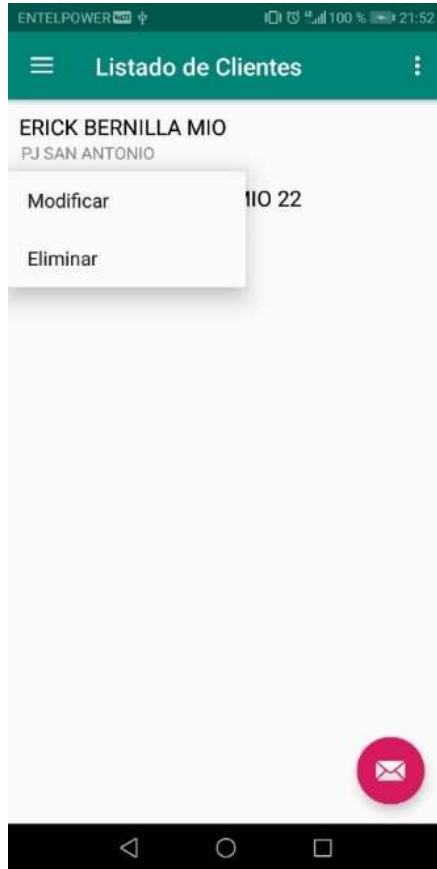


5.1.- Módulo Clientes

A. Registro. Para empezar con el uso de la aplicación se deben registrar los clientes y se deben llenar los siguientes campos:



B.- Modificación y eliminación. Con esta opción se puede modificar o eliminar.



C.- Listar clientes: Se muestra una lista de todos los clientes registrados



5.2.- Módulo productos

A. Registro. Se debe ingresar todos los campos para registrar un producto

The screenshot displays a mobile application interface for product registration. The title bar at the top is green and contains a hamburger menu icon, the text "Registrar Producto", and a vertical ellipsis icon. The status bar above shows "ENTELPOWER" and "100% 21:02". The form consists of several input fields: "Nombres" with the value "ZAPATOS", "Precio Venta" with the value "23.0", "Precio Compra" with the value "10.0", and "Stock" with the value "10". Below these fields is a dropdown menu currently showing "ACTIVO". At the bottom of the form is a prominent blue button labeled "MODIFICAR". The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps icons.

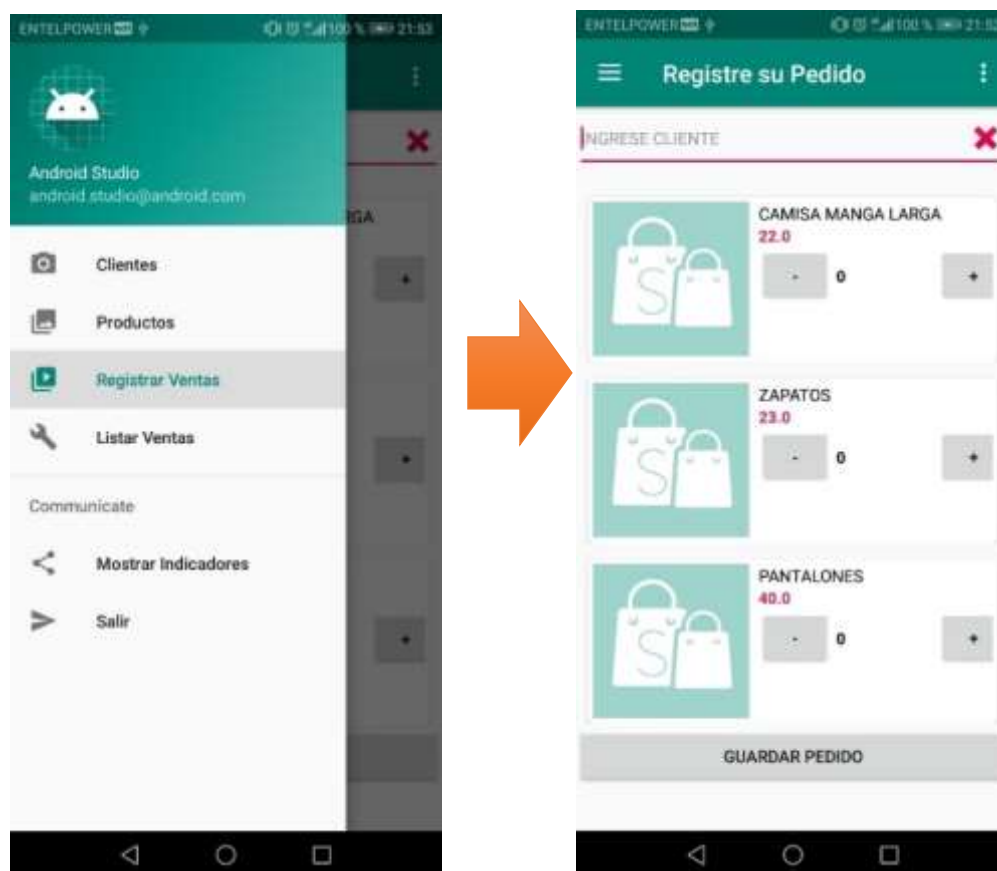
B. Modificación y Eliminación. Dando clic en la lista se puede modificar o eliminar.

C. Listar. Se mostrará una lista detallada de todos los productos registrados.

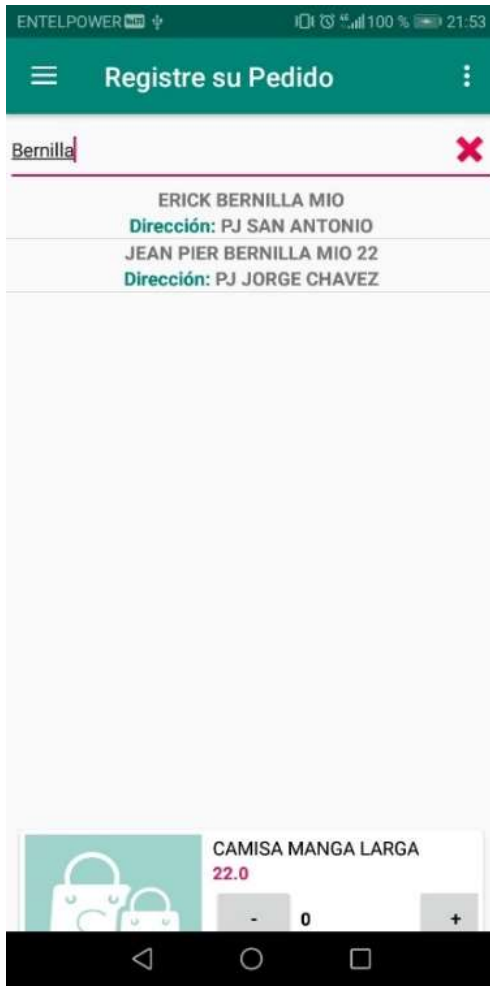


5.3.- Módulo ventas

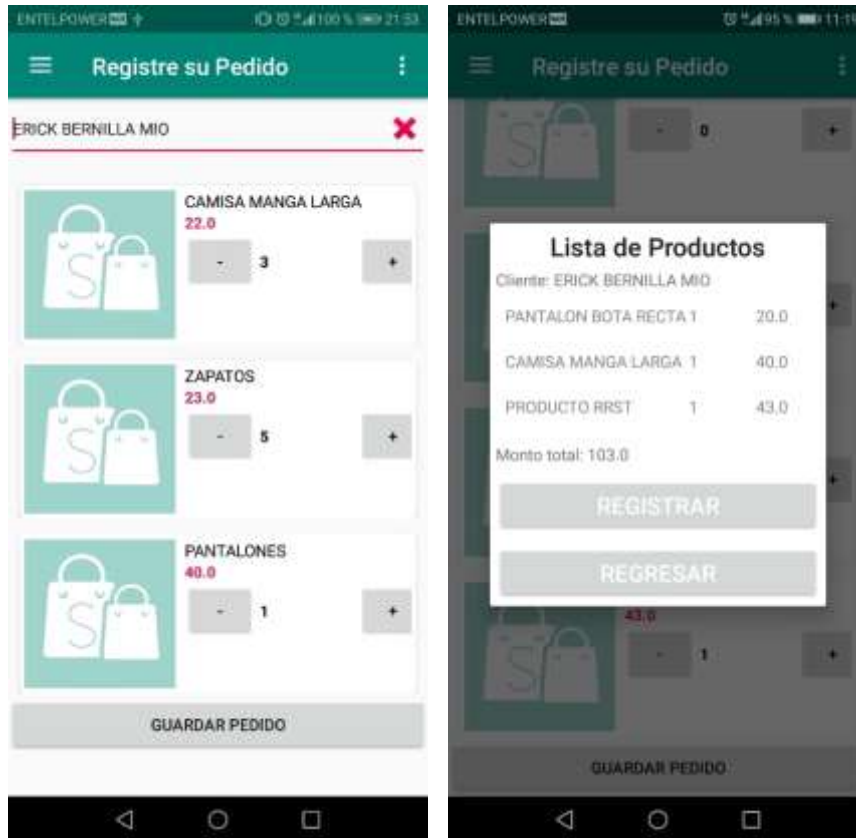
A. Registro de pedidos. Se listan los productos con la cantidad de y el tipo de producto a pedir



B Selección de cliente: Se asigna el cliente para proceder a agregar los productos.



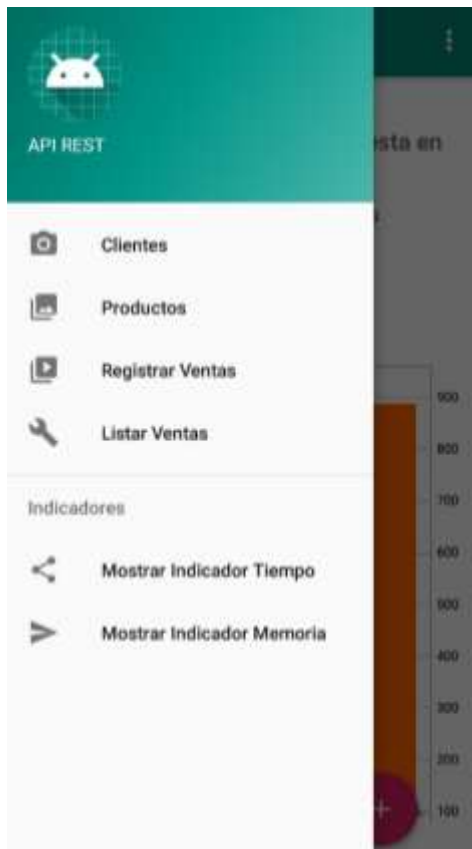
C. Agregar producto: Se lista los productos y se selecciona aquellos que se desea adquirir.



6. API REST y GRAPHQL.

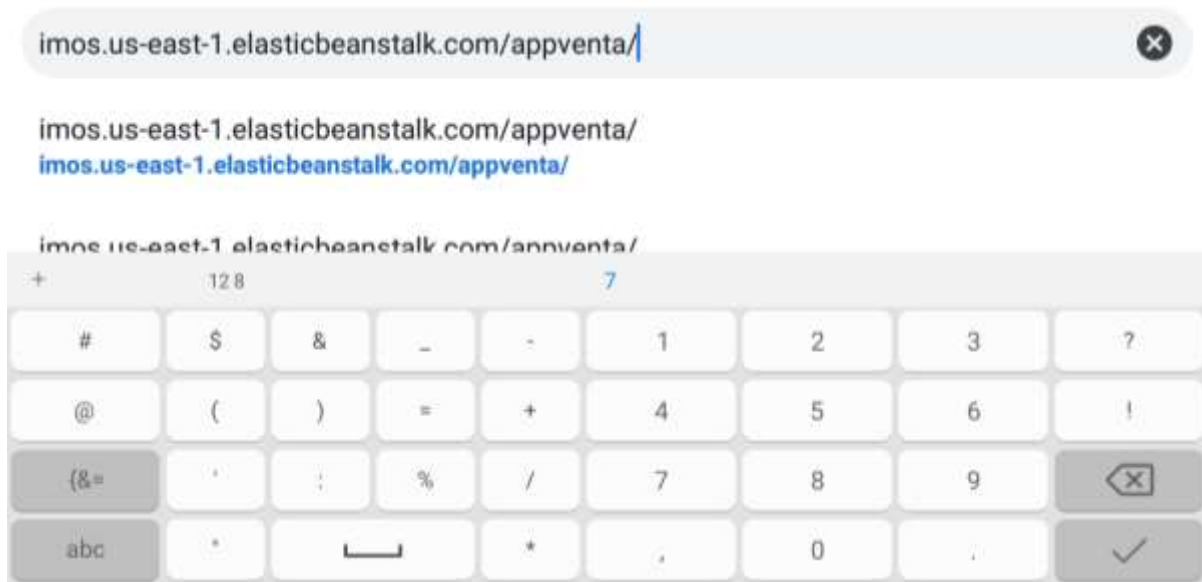
Se mostrará los módulos de indicadores tiempo y memoria expresados en graficas de barras, los datos usados son los que se recopilan al usar los módulos de clientes, productos y ventas.

En la siguiente imagen se puede ver los módulos agregados.

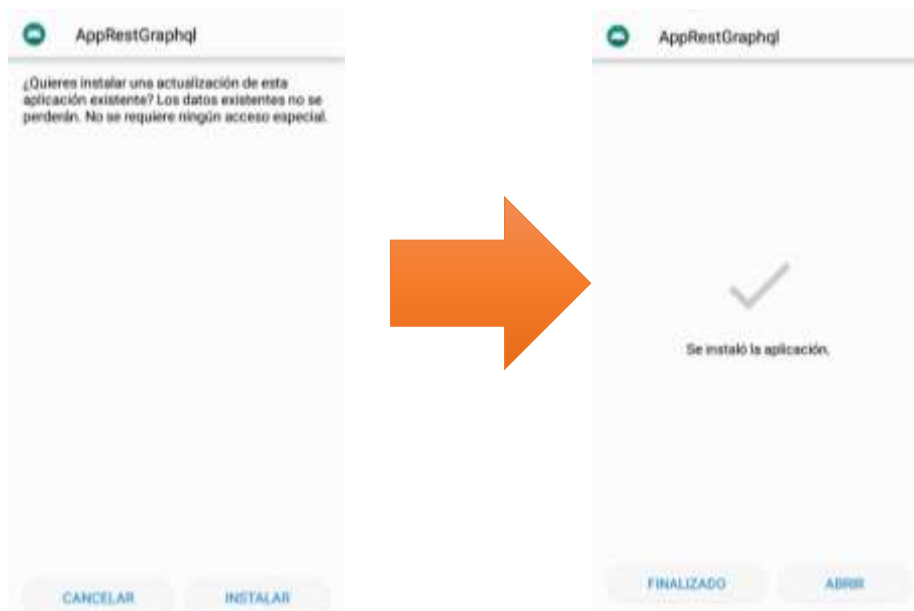


6.1.- Configuración y uso de la aplicación.

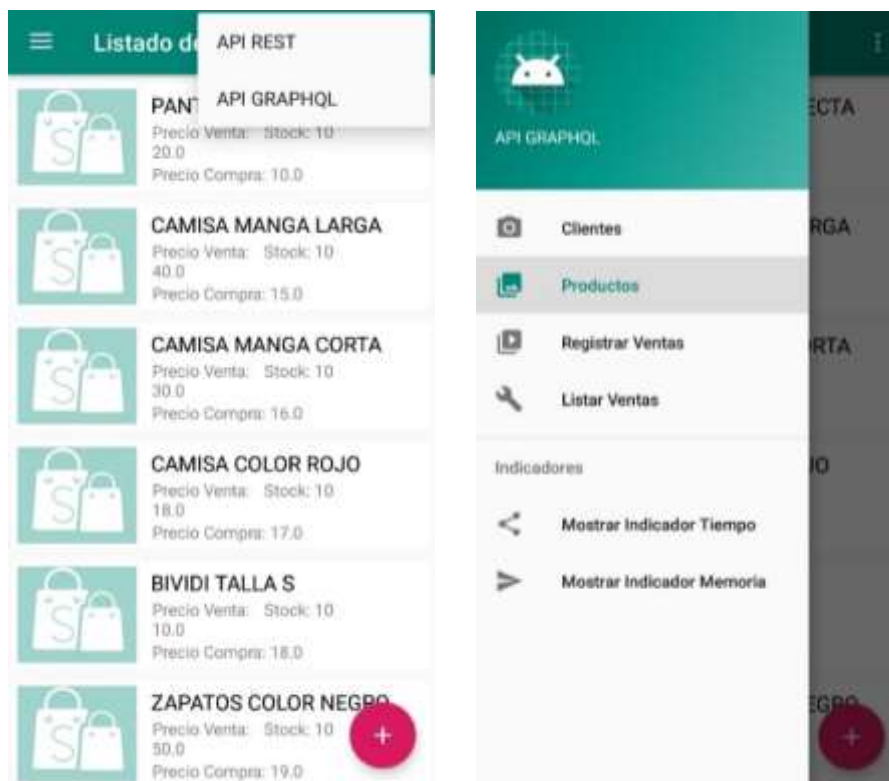
A.- Descarga. Para descargar la aplicación debe ingresar al siguiente enlace <http://imos.us-east-1.elasticbeanstalk.com/appventa/> desde el navegador de su celular



Mostrará la opción de instalar.



C.- Funcionamiento: La interfaz general muestra en la esquina derecha una opción para cambiar el tipo de petición REST o GRAPHQL que realizaran los módulos de clientes, productos y ventas, las operaciones que realizan cada módulo ya fue explicado en el informe número 2.



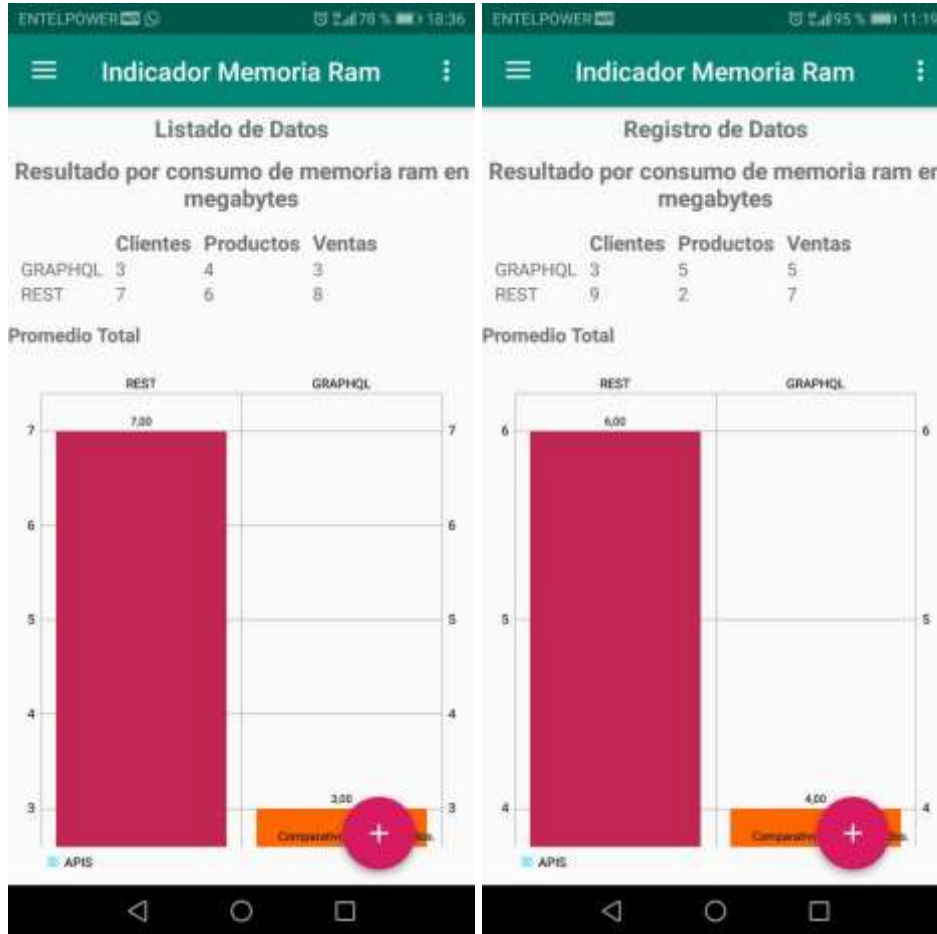
6.2.- Indicadores

A.- Mostrar indicador tiempo. El siguiente modulo nos muestra un promedio de tiempo en milisegundos de las ultimas 10 peticiones de cada API, a partir de dichos promedios se hace un promedio general y dichos datos se muestran en una gráfica de barras.

En la parte inferior derecha hay un botón flotante que nos da a elegir entre ver el promedio de las peticiones de tipo listado o las peticiones de tipo registro, se creyó conveniente separarlas por tipo de peticiones para mantener la fiabilidad de los datos.

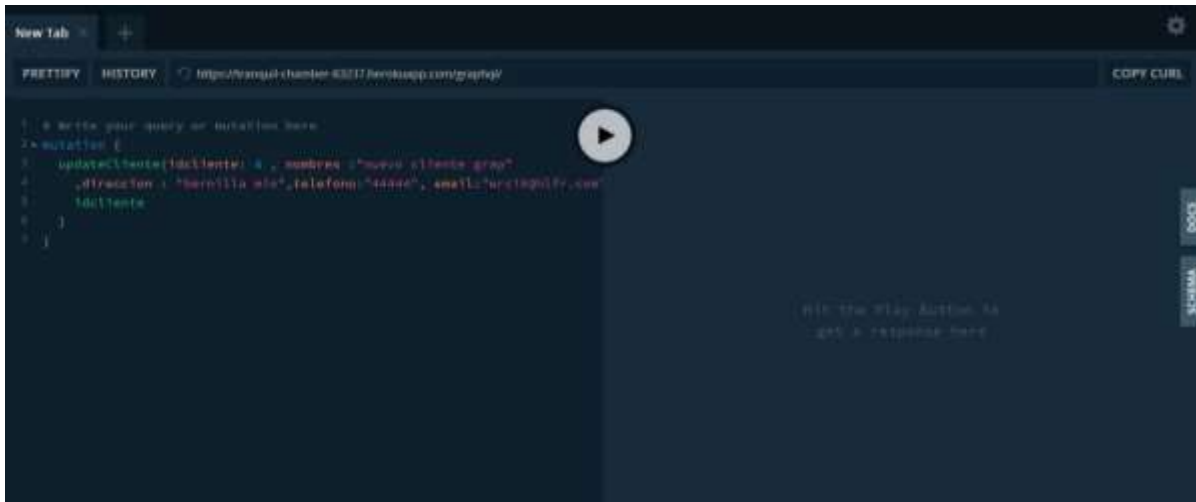


B.- Mostrar indicador memoria. De la misma manera en este módulo nos muestra el promedio, pero esta vez expresado en megabytes.



6.3.- Pruebas en la nube de las APIS.

A. API GRAPHQL. <https://tranquil-chamber-83237.herokuapp.com/graphql/>



B. API REST. <http://imos.us-east-1.elasticbeanstalk.com/>

Definicion de API REST

API significa Interfaz de Programación de Aplicaciones. Una Interfaz de Programación de Aplicaciones te permite establecer una conexión o enlace entre 2 tipos de software diferentes. Una API facilita el intercambio de datos entre 2 tipos diferentes de software. Estás utilizando diariamente APIs, por ejemplo cuando vas logueando a través de la cuenta de google o via Facebook estamos utilizando APIs. Por lo tanto, las APIs permiten a los desarrolladores usar contenido o características de una app, servicio o plataforma diferente en un servicio, plataforma o app propia de una manera segura.

Listad de EndPoints

- [Lista Clientes](#)
- [Lista Productos](#)
- [Lista Ventas](#)