UNI-FOSS: UNA PROPUESTA WEB PARA NECESIDADES DE INFORMACIÓN UNIVERSITARIA - CASO DE ESTUDIO UNIVERSIDAD DE NARIÑO

DAVID ALEXANDER CERON
JHON MARIO GETIAL

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2021

UNI-FOSS: UNA PROPUESTA WEB PARA NECESIDADES DE INFORMACIÓN UNIVERSITARIA - CASO DE ESTUDIO UNIVERSIDAD DE NARIÑO

DAVID ALEXANDER CERON
JHON MARIO GETIAL

Trabajo de grado como requisito para obtener el título de Ingeniero de Sistemas

Asesor
ING. M. G. FRANKLIN EDUARDO JIMÉNEZ GIRALDO
Co-Asesor
ING. M. G. GIOVANNI HERNANDEZ

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2021

NOTA DE RESPONSABILIDAD

Las ideas y conclusiones aportadas en el trabajo de grado son responsabilidad exclusiva de sus autores.

Artículo 1º. Del acuerdo No. 324 del 11 de octubre de 1966 emanado del Honorable Consejo Directivo de la Universidad de Nariño. “La Universidad de Nariño no se hace responsable de las opiniones o resultados obtenidos en el presente trabajo y para su aplicación priman las normas sobre el derecho de autor”.

Artículo 13, Acuerdo No. 005 de 2010 emanado del Honorable Consejo Académico.

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

Firma del Jurado

_____

Firma del Jurado

_____

Firma del Asesor

_____

Firma del Co-Asesor

San Juan de Pasto, abril 2021

“La inteligencia es la habilidad de adaptarse a los cambios”

Stephen Hawking

“Cuando se innova, se corre el riesgo de cometer errores. Es mejor admitirlo rápidamente y continuar con otra innovación”

Steve Jobs

DEDICATORIA

A mi familia por su cariño y por ser fuente de inspiración para seguir adelante. A mis padres por sus sacrificios para ponerme en donde estoy hoy. A mi hermanita, la única capaz de hacerme olvidar mi mal genio y de alegrarme los días oscuros con su compañía.

A Madrov S.A.S y mis compañeros de trabajo por los aprendizajes y colaboración durante este tiempo, y por la flexibilidad y apoyo que me brindaron durante la realización del proyecto.

A la Universidad de Nariño, por formarme como profesional, pero aún más importante, como persona, con una visión más amplia del mundo y como aportar en él.

A todos los que, con su paciencia y comprensión, apoyaron la culminación del proyecto en medio de estos tiempos de pandemia, de incertidumbre y desnaturalización de la cotidianidad y el mundo que conocíamos.

David Alexander Cerón

DEDICATORIA

Agradezco infinitamente a mis padres por el apoyo incondicional, ese apoyo que me permitió ser libre y feliz para poder formarme como profesional. Por la esperanza, admiración y la promesa de ser ante todo una persona integra en todos los aspectos de la vida. Por todos los sacrificios que silenciosamente superaron para darme esta oportunidad de salir adelante, y por el orgullo y la alegría de ser su hijo, que para mí es el combustible que me impulsa para dedicarles cada uno de mis logros.

A mi abuela Elvira que con su especial cariño y cuidado me ha dado fuerzas para salir adelante y que al igual que mis padres representa una inspiración, solo por la simple razón de ver reflejado en su rostro la alegría y el orgullo de ser su nieto, y en general a toda mi familia que me ha apoyado en los momentos que he necesitado.

A todos nuestros docentes que hacen su mejor esfuerzo para brindarnos su valioso conocimiento; especialmente a nuestros asesores que con su ayuda pudimos alcanzar este último escalón para cumplir esta meta.

A todos mis compañeros, amigos y demás personas que han influenciado en mi aprendizaje tanto en lo profesional como en lo personal.

Jhon Mario Getial

RESUMEN

Las organizaciones soportan su funcionamiento en el flujo de información, sin embargo, los requisitos para cada una pueden variar; por tanto, es indispensable cuando se habla de manejo de información, adaptarse a requisitos diversos, complejos y cambiantes. Esta investigación, realizada en la Universidad de Nariño, aplico una serie de encuestas para validar las necesidades de acceso a la información y carencias en servicios actualmente brindados en la Universidad de Nariño. En los datos recolectados se observó una gran variedad de requisitos a solventar, muchos procesos e información de diversos ámbitos que requieren ser entregados a la comunidad universitaria. Esto permitió validar la necesidad de un mecanismo de gestión de información que sea flexible y que dé respuesta a los diversos requisitos relacionados con ello. La investigación teórica en búsqueda de esa flexibilidad arrojó Graphql y arquitectura de software como los conceptos clave. Se desarrolló un administrador de información, una aplicación web que permite establecer conexiones con cualquier motor de base de datos, organizar la información y entregarla a partir de una API Graphql. Finalmente se hizo una validación del aplicativo y su aporte en el proceso de entrega de información para la Universidad de Nariño.

PALABRAS CLAVE: API, GRAPHQL, ACCESO A LA INFORMACION, DESARROLLO DE APLICACIONES, FLEXIBILIDAD, ESCALABILIDAD

ABSTRACT

Organizations support their operation in the flow of information, however, the requirements for each one may vary; Therefore, when it comes to information management, it is essential to adapt to diverse, complex and changing requirements. This research, carried out at the University of Nariño, applied a series of surveys to validate the needs for access to information and gaps in services currently provided at the University of Nariño. In the data collected, a great variety of requirements to be met, many processes and information from various fields that need to be delivered to the university community were observed. This allowed validating the need for an information management mechanism that is flexible and responds to the various requirements related to it. Theoretical research in search of that flexibility yielded Graphql and software architecture as the key concepts. An information manager was developed, a web application that allows to establish connections with any database engine, organize the information and deliver it from a Graphql API. Finally, the application was validated and its contribution to the information delivery process for the University of Nariño.

KEYWORDS: API, GRAPHQL, ACCESS TO INFORMATION, APPLICATIONS DEVELOPMENT, FLEXIBILITY, SCALABILITY

CONTENIDO

LISTADO DE TABLAS

LISTADO DE GRÁFICAS

Glosario

Endpoint: Los endpoints API son la locación digital específica a la que llegan todas las peticiones de información enviadas por un programa, y es desde donde se envía como respuesta esta información.

URI: En español significa Identificador Uniforme de Recursos, y sirve para identificar recursos en Internet. Provee un formato estándar definido y su propósito es permitir interacción entre recursos disponibles en Internet, o en alguna red de cómputo.

Query: En español significa consulta, y en tecnología hace referencia a la acción de consultar información de una fuente, por ejemplo, una API o una base de datos.

SQL: En español significa Lenguaje de Consulta Estructurada, y es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales.

HTTP: En español significa Protocolo de Transferencia de Hipertexto, es el protocolo de comunicación que permite las transferencias de información en la web.

Frontend: Es una de las dos partes de la que se componen las aplicaciones informáticas, comprende las interfaces y todo lo que es visible por parte del usuario y con lo cual este interactúa.

Backend: Es la parte trasera de las aplicaciones y aporta toda la carga de procesamiento de datos y trabajo. Se encarga de que una página funcione y de que lo haga como lo hace.

Escalabilidad: En informática hace referencia a la capacidad del sistema informático de cambiar su tamaño o configuración para adaptarse a circunstancias cambiantes.

Mantenibilidad: Propiedad de un sistema que representa la cantidad de esfuerzo requerida para conservar su funcionamiento normal o para restituirlo una vez se ha presentado un evento de falla.

Framework: En español significa entorno de trabajo y es una estructura conceptual y tecnológica, generalmente compuesta por módulos de software, que puede servir de base para la organización y desarrollo de software.

Open Source: En español significa código abierto y es un modelo de desarrollo de software que se basa en la colaboración abierta, es decir, se permite que el programa en cuestión sea modificado de manera abierta por otros usuarios.

Cliente: Es una aplicación informática o un ordenador que consume un servicio remoto en otro ordenador conocido como servidor, normalmente a través de una red de telecomunicaciones.

Servidor: Es un equipo informático que forma parte de una red y provee servicios a otros equipos cliente.

INTRODUCCIÓN

En la actualidad el uso de aplicaciones informáticas está íntimamente ligado a la vida de todos, a sus actividades, a su trabajo y en general a la forma en la que interactúan con el mundo. Las personas son cada vez más dependientes de la tecnología tanto en su vida cotidiana como laboral. En el caso de las organizaciones, la conexión entre sus integrantes e incluso sus vínculos con otras organizaciones están marcados por esta misma realidad y han tenido que adaptarse a una realidad en la que el consumo de servicios e información es fundamental. Las organizaciones han tenido que incursionar en la creación e integración de productos software, para ofrecer nuevas e innovadoras utilidades que den respuesta a las necesidades de acceso a la información en el mundo moderno.

Esta investigación inició indagando sobre conceptos y herramientas software útiles en la implementación de sistemas de información, donde surgieron algunos conceptos clave como el de API, indispensable en los procesos de flujo de información del mundo informático. Posteriormente se describe las características de la investigación, cuyo objetivo es el de aportar en el proceso de acceso a la información en la Universidad de Nariño. Este objetivo se ejecutó en tres etapas, una caracterización de las necesidades de acceso a la información, el desarrollo de un aplicativo que aporte en su resolución y un proceso de validación del nivel de aporte de ese aplicativo.

La primera etapa está contenida en el capítulo tres. Describe la aplicación de encuestas dirigidas a la comunidad universitaria de la Universidad de Nariño, y en los resultados se evidenció la existencia de diversas necesidades de acceso a información, y el interés en la ampliación de los servicios actualmente ofrecidos por la Universidad.

La segunda etapa está contenida en el capítulo cuatro. Describe el proceso de creación de una aplicación web que permite gestionar y administrar la información que la universidad necesita compartir, para que sea consumida a través de una API.

Se hizo uso de herramientas que ofrecen un marco de trabajo flexible y escalable como GraphQL y Django.

La etapa final muestra los resultados del proceso de validación del aplicativo. Mediante la aplicación de encuestas, se recoge la opinión de los desarrolladores del centro de informática de la Universidad de Nariño entorno al aplicativo propuesto, evaluando que mejoras o utilidades ofrecería, y el nivel de interés que hay en su utilización en la Universidad.

Línea de investigación

Línea Software y manejo de Información.

Alcance y delimitación

Esta investigación realizó una caracterización de las necesidades de acceso a la información académica e institucional en la comunidad universitaria de la Universidad de Nariño, se tomaron 3 grupos poblacionales, los estudiantes, los docentes y el personal administrativo, se aplicaron encuestas recolectando información que permitió conocer el uso de los servicios de acceso a la información académica e institucional brindados por la Universidad y las necesidades aún existentes. También se exploraron conceptos y herramientas útiles para el proceso de gestión de información y se hizo una comparativa y análisis de las alternativas. Con base en la exploración descrita, se desarrolló una aplicación web que permite gestionar y administrar la información que la universidad necesita compartir, para que sea consumida a través de una API. La aplicación propuesta ofrece a la Universidad de Nariño una forma flexible y escalable de desarrollar soluciones software, simplificando la creación e integración de nuevos productos, que puedan satisfacer las necesidades de acceso a la información de la comunidad universitaria. Finalmente, se hizo un proceso de validación de la aplicación, presentándolo ante

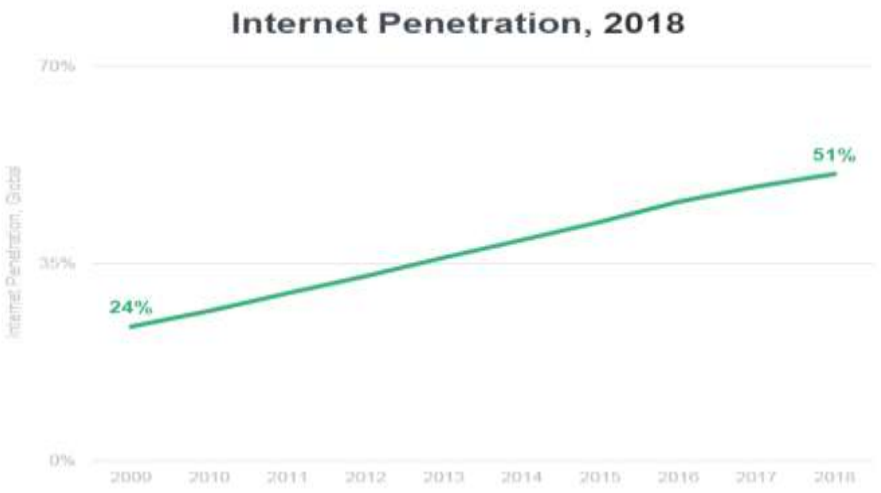
los desarrolladores del centro de informática de la Universidad de Nariño, y aplicando unas encuestas para que ellos evalúen el nivel de aporte que puede ofrecer para el proceso de desarrollo de soluciones que fortalezcan los procesos de entrega de información hacia la comunidad universitaria.

Descripción del problema

Planteamiento del problema

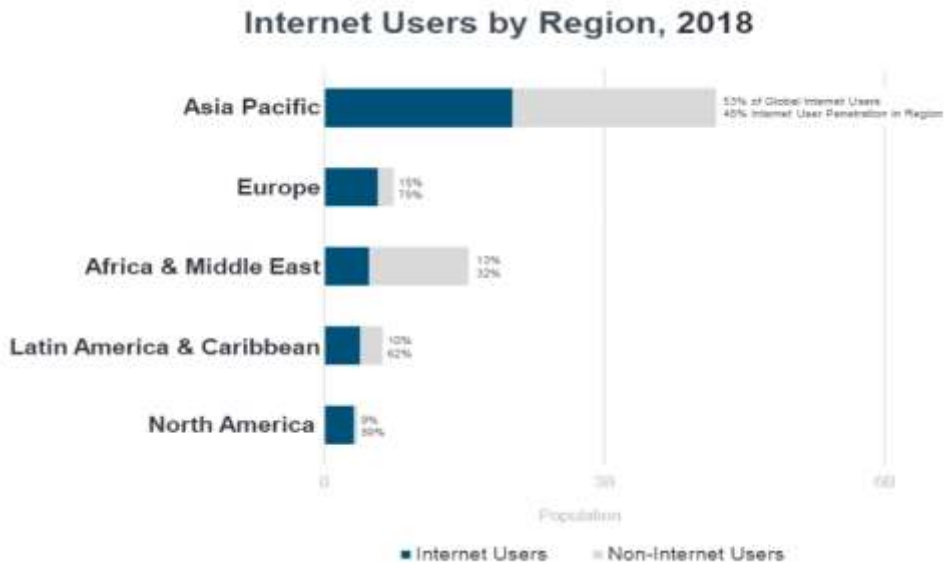
El internet se ha convertido en una de las herramientas más importantes del mundo, y su nivel de penetración ha aumentado dramáticamente (Meeker, 2019), pasando del 24% de personas en el mundo con acceso a internet en el 2009 al 51% en 2018 (Ver gráfica 1). Esto significa que el mundo digital llega cada vez a más personas, y en el caso de Latinoamérica, el porcentaje de la población con acceso ya es de 62 % para el 2018, por detrás de Europa y Norteamérica (Ver gráfica 2), ello implica que el potencial de crecimiento de usuarios aún es alto. Esta tendencia es importante porque marca el crecimiento de un nuevo mercado que tiene como base la conexión entre los usuarios, en tiempo real, con el mundo. Las personas se conectan con servidores y datos a través de internet, y la enorme mayoría de servicios que consumen involucra de algún modo una interacción con internet.

Gráfica 1 ***Porcentaje de personas en el mundo con acceso a Internet.*** *(Meeker, 2019)*



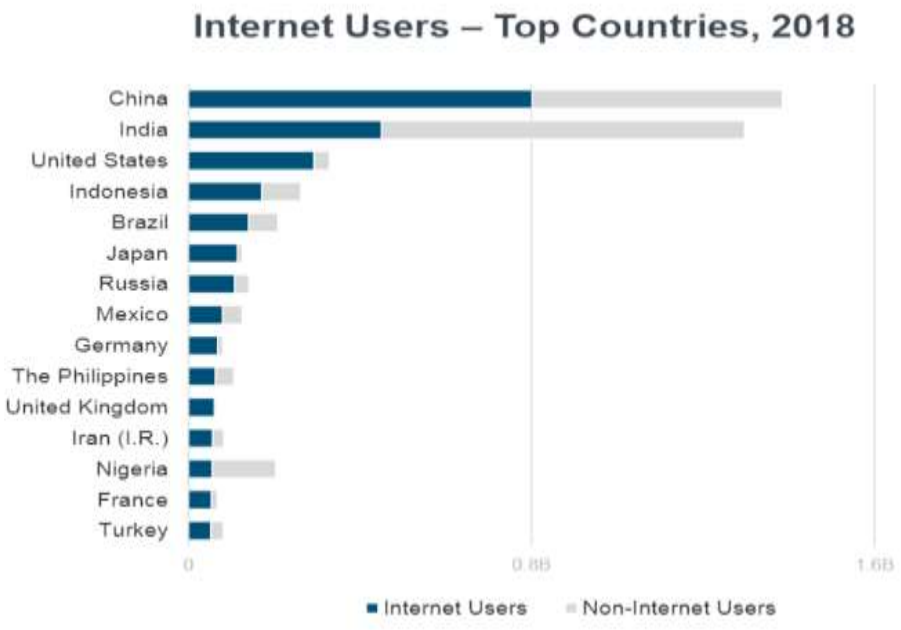
Fuente: datos de Meeker, 2019

Gráfica 2 ***Usuarios de Internet por región.*** *(Meeker,2019).*



Fuente: Meeker, 2019

Gráfica 3 ***Top usuarios de Internet por país.*** *(Meeker, 2019).*



Fuente: Datos de (Meeker, 2019).

Lo anterior se confirma si se analiza que actualmente son las empresas tecnológicas las que dominan la economía mundial, las llamadas 'Big Five': Amazon, Apple, Facebook, Microsoft, y Alphabet conjuntamente tienen ganancias que superan los 800 billones de dólares anuales (Desjardins, 2019), tienen niveles muy altos de crecimiento, y constantemente compiten entre ellas para encabezar la lista como la empresa más grande del mundo (Ver gráfica 4). El dominio por parte de estas empresas sobre la economía se ha mantenido durante ya varios años, y no parece tener indicios de cambiar (BBC Mundo, 2017).

Gráfica 4 ***Empresas de tecnología más grandes del mundo y su nivel de crecimiento económico de 2016 a 2019.*** *(Meeker, 2019).*

Rank			Market Cap Value ($B)		
2019	Company	Region	6/7/19	6/7/16	% Change
1	Microsoft	USA	$1,007B	$410B	+146%
2	Amazon	USA	888	343	+159%
3	Apple	USA	875	540	+62%
4	Alphabet	USA	741	497	+49%
5	Facebook	USA	495	340	+46%
6	Alibaba	China	402	195	+106%
7	Tencent	China	398	206	+93%
8	Netflix	USA	158	43	+266%
9	Adobe	USA	136	50	+174%
10	PayPal	USA	134	46	+190%
11	Salesforce	USA	125	56	+123%
12	Booking.com	USA	77	67	+15%
13	Uber	USA	75	--	--
14	Recruit Holdings	Japan	52	20	+167%
15	ServiceNow	USA	51	12	+316%
16	Workday	USA	48	16	+197%
17	Meituan Dianping	China	44	--	--
18	JD.com	China	39	32	+22%
19	Baidu	China	38	60	(36%)
20	Activision Blizzard	USA	35	28	+25%
21	Shopify	Canada	34	2	+1,297%
22	NetEase	China	33	23	+44%
23	eBay	USA	33	28	+19%
24	Atlassian	Australia	32	5	+509%
25	MercadoLibre	Argentina	30	6	+388%
26	Twitter	USA	29	11	+173%
27	Square	USA	29	3	+808%
28	Electronic Arts	USA	29	23	+25%
29	Xiaomi	China	28	--	--
30	Spotify	Sweden	25	--	--
	Total		$6,119	$3,064	

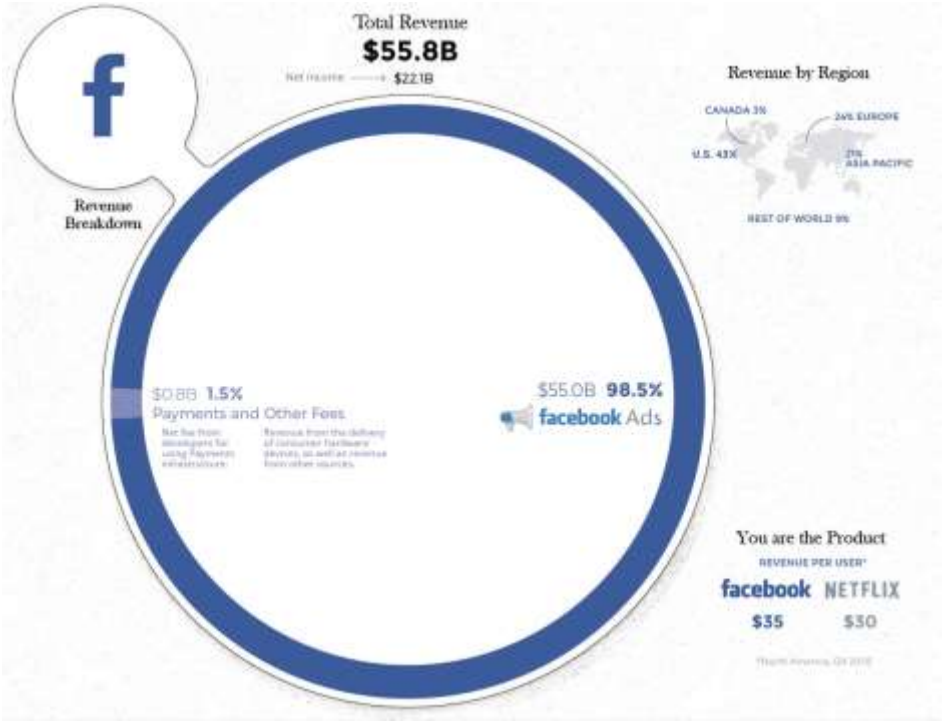
Fuente: Datos de Meeker, 2019.

Estas empresas han revolucionado el modo en el que se interactúa con el mundo, y sus tecnologías han sido fuerzas de transformación que han moldeado la realidad y cotidianidad de las personas, por ejemplo, el IPhone de Apple es la base de lo que se conoce hoy como teléfonos inteligentes (Bajarin, 2017), y gracias a este dispositivo y los derivados de él, como sus equivalentes Android, las personas tienen actualmente muchas más opciones para conectarse con el mundo sin importar su ubicación. Otro de muchos ejemplos es Facebook, que ha transformado completamente la forma en la que las personas se comunican entre ellas con su red social, pero, además, ha ofrecido al mundo variedad de tecnologías y herramientas que hacen posible muchas de las aplicaciones y servicios que nos acompañan diariamente (Finley, 2019).

Los multimillonarios modelos de negocios de estas grandes empresas tienen algo en común, y es que se centran en el flujo de información, teniendo millones de personas que usan sus plataformas en el mundo, compartiendo datos e información a tal punto, que se ha llegado a convertir en armas de doble filo, con frecuentes cuestionamientos sobre el manejo que se le da a esa información (Desjardins, 2019). De forma general, los modelos de negocio de Apple, Amazon y Microsoft consisten en ofrecer productos y servicios a sus clientes a cambio de un pago, y aunque cada una tiene un enfoque diferente, han logrado cubrir muchas de las necesidades de los usuarios y el mercado, con grandes márgenes de ganancia.

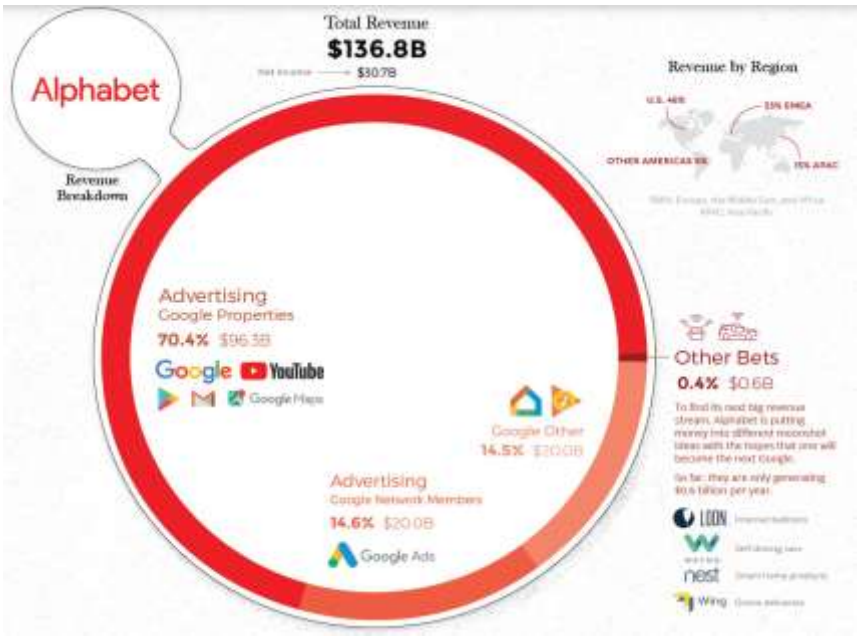
Por otro lado, en los modelos de negocio de Facebook y Google, el usuario se convierte en el producto, pues son sus datos, que surgen de las interacciones que hacen con esas plataformas, lo que venden como valiosa información a otras empresas, haciendo llegar la publicidad adecuada a cada usuario o posible comprador. Para 2018, el 98.5% de las utilidades de Facebook provenían de publicidad (Ver gráfica 5), de forma similar, Alphabet, la empresa detrás de Google y YouTube, generó el 85% de sus utilidades en publicidad para el mismo año (Ver grafica 6).

Gráfica 5 ***Fuentes de ingreso de Facebook.*** *(Desjardins, 2019).*



Fuente: Datos de (Desjardins, 2019).

Gráfica 6 ***Fuentes de ingreso de Alphabet.*** *(Desjardins, 2019).*



Fuente: datos de (Desjardins, 2019).

Así pues, el mundo está cada vez más ligado a la tecnología, y su uso está transformándolo, cambiando la forma en la que se crean negocios, los requisitos para generar valor, y hasta el modo en el que las personas trabajan, interactúan y se comunican; tecnologías como la computación en la nube y la móvil, el big data y machine learning, la manufactura inteligente y los sensores, la robótica y los drones y las tecnologías de energía limpia, entre otras tantas, están permitiendo que las cosas se hagan más rápido y de mejor forma (Cascio & Montealegre, 2016), un ejemplo de esto es la existencia de máquinas comunicándose entre ellas sin intervención humana, tomando decisiones y afectando las labores cotidianas de las personas (Chai, 2019), esta es la realidad hecha posible por el Internet de las cosas, que presenta a los usuarios la promesa de mayor eficiencia y comodidad en sus actividades.

Todas estas innovaciones, en conjunto, están provocando cambios profundos en la forma en que las organizaciones funcionan, encaminándonos a una nueva revolución industrial, y haciendo que el uso de estas nuevas herramientas sea de vital importancia para la competitividad e incluso la supervivencia de las organizaciones y negocios, que deberán adaptarse y transformarse en este nuevo contexto, o enfrentar las consecuencias de que otros lo hagan primero (Murray, 2015). Y esta conclusión no se limita al mundo de los negocios, pues hay amplia literatura que hace observaciones similares enfocadas a los campos de la medicina, la ingeniería y las ciencias exactas y sociales, en otras palabras, la tecnología se está convirtiendo rápidamente en algo tan importante como la electricidad (Cascio & Montealegre, 2016).

Las Universidades, que, debido a su estructura, la multiplicidad de fines y la intervención de múltiples actores, son un tipo de organización especialmente compleja (Miceli, 2019), no pueden ser ajenas a los desafíos del mundo tecnológico. Como ocurre con cualquier otra organización, el modelo de funcionamiento de las universidades está cada vez más ligado a las tecnologías de la información, y, de

hecho, existe una correlación entre el uso de la tecnología y la eficacia con la que las universidades pueden responder en sus procesos (García Peñalvo, 2011).

En el caso de la Universidad de Nariño, se cuenta con un sistema de información que le permite a su comunidad realizar múltiples acciones, consultando y gestionando información a través de los sitios web que están a su disposición. Estos recursos han sido desarrollados por equipos de personas destinados por la propia Universidad para ese propósito. Se invierte personal y recursos para mantener este sistema de información, proporcionando enlaces que permiten el flujo de información entre la Universidad y su comunidad. Esta investigación pretende proponer aportes que permitan contribuir en la consolidación de un sistema de información basado en el uso de la tecnología, que pueda dar respuesta a las necesidades de la comunidad universitaria de la Universidad de Nariño.

Formulación del problema

¿Cómo aportar al proceso de acceso a información académica e institucional de la Universidad de Nariño?

Sistematización del problema

¿Qué necesidades de acceso a la información presenta la comunidad universitaria de la Universidad de Nariño?

¿Cómo contribuir en el proceso de acceso a la información en la Universidad de Nariño?

¿Cómo evaluar el nivel del aporte brindado al proceso de acceso a la información en la Universidad de Nariño?

Objetivo general

Aportar al proceso de acceso a información académica e institucional de la universidad de Nariño.

Objetivos específicos

- Caracterizar las necesidades de acceso a información académica e institucional de la comunidad universitaria de la Universidad de Nariño.
- Desarrollar un aplicativo que contribuya en el proceso de acceso a información académica e institucional de la comunidad universitaria de la Universidad de Nariño.
- Validar, en un entorno de pruebas, el aporte del aplicativo y los módulos implementados.

Justificación

Las organizaciones soportan su funcionamiento en el flujo de información, es el caso de la Universidad de Nariño, sus procesos académicos y administrativos dependen de ese flujo de información: entre dependencias y facultades, entre profesores y estudiantes, entre administrativos y comunidad universitaria en general, etc. Sin embargo, es posible que cada una de las partes interesadas, en especial en organizaciones tan grandes como una Universidad, tengan necesidades diversas, complejas y cambiantes. Es por esto que, en primera instancia, es muy importante tener una idea de qué tipo de necesidades relacionadas con el acceso a la información existen en la Universidad, para así poder aportar en su resolución, lo cual podría mejorar enormemente la experiencia de la comunidad.

Dado que el entorno tecnológico evoluciona y cambia constantemente, y siempre surgen nuevas propuestas, alternativas y oportunidades, es importante hacer revisión de los sistemas de manejo de información y adaptarse a las necesidades y

condiciones impuestas por la tecnología. Es indispensable el plantearse constantemente el cómo aprovechar las nuevas herramientas disponibles para contribuir en la mejora constante de procesos, además de asegurarse de mantener la flexibilidad y escalabilidad de los sistemas, de modo que la adaptación a estos cambios sea realmente posible.

Por último, resulta útil validar el impacto y potencial que tiene la implementación de tecnologías de vanguardia en sistemas de información creados antes de su existencia. Es también una manera de validar la capacidad de adaptabilidad de estos sistemas, y las ventajas que puede proveer una tecnología nueva en comparación.

Marco de referencia

Antecedentes

La adaptación al uso de la tecnología ha sido uno de los retos más grandes de las universidades en las últimas décadas. Y, aunque ya se ha acoplado a muchos de los procesos de las mismas, la tecnología sigue avanzando a un ritmo bastante alto, con nuevas herramientas y posibilidades que asimilar, y en las que invertir. Por este motivo el proceso de adaptación se torna difícil, pudiéndose generar estancamientos y pérdida de competitividad con respecto a otras instituciones que sean capaces de adaptarse mejor (García Peñalvo, 2011). Esta problemática ha sido ampliamente abordada por diversos proyectos de investigación.

Como primer ejemplo, una investigación de la Universidad A&M de Texas en Galveston en 2015, planteó la importancia de conectar con los estudiantes desde su primer año en la universidad como estrategia para prevenir la deserción, y hace la observación de que la generación actual de jóvenes, que constituye la gran mayoría de la población estudiantil, se comunica e interactúa fundamentalmente mediante el uso de la tecnología, especialmente dispositivos móviles. Por esta razón, se propone una aplicación móvil intuitiva y fácil de usar, con el branding de la universidad, que ofrezca soporte académico y no académico para los estudiantes, especialmente los nuevos, y que además permita a los administradores acceder a reportes y análisis de la interacción de los estudiantes para hacer ajustes en las estrategias de ser necesario (Hoff, 2015). En este ejemplo se evidencia lo útil que puede ser apoyarse en herramientas tecnológicas para cumplir con los objetivos misionales de las organizaciones. Esto, como es natural, aplica también para las Universidades, en este caso, apoyando el proceso de los estudiantes nuevos para evitar su deserción, y casos similares pueden encontrarse involucrando a los diferentes actores que componen la Universidad como organización, y es algo que deberá tenerse en cuenta en la presente investigación.

Otra investigación, realizada en la Pontificia Universidad Javeriana de Colombia en 2016, resaltó la existencia de problemas relacionados con el acceso a la información de localización y la disponibilidad de servicios de la universidad, lo cual provocaba problemas de coordinación en las actividades académicas. Se propuso el desarrollo de una aplicación móvil que permitiera consultar y compartir información de disponibilidad de servicios estudiantiles de la Pontificia Universidad Javeriana, además de fortalecer lazos entre la comunidad al interconectarla (Peña Gómez & Ramírez Salamanca, 2016). En el mismo año, en la Universidad Tecnológica del Perú, se desarrolla una propuesta similar, trasladando ciertos servicios ya ofrecidos en su plataforma web, relacionados a procesos académicos, a una aplicación móvil destinada tanto a docentes como a estudiantes, con la intención de facilitar acceso y reducir tiempo en la ejecución de estos procesos (Reyes Rodriguez, 2015). Así pues, siendo la Universidad una organización de múltiples actores interactuando entre sí, se vuelve una necesidad el encontrar mejores formas de relacionarlos entre ellos, mejorando canales de comunicación y flujo de información, y la tecnología juega un papel fundamental en ello.

Finalmente, una referencia más cercana. En 2017, en la Universidad Mariana de Pasto, Colombia, se realizó una investigación sobre necesidades de acceso a la información de su comunidad universitaria, con la intención de ofrecer una propuesta software que aporte en su resolución. La investigación identifica una serie de servicios de interés para la comunidad universitaria y presenta dos aplicaciones que pretenden dar respuesta a estas necesidades. La primera consiste en un aplicativo web que permite al administrador gestionar la información que desea bridar a sus usuarios, conectando con diferentes gestores de bases de datos y ofreciendo información en forma de servicios a través de una API REST. La segunda es aplicación móvil llamada UMovil, que consume los servicios de la primera, y mediante la cual es posible hacer consultas sobre horarios, notas, listado de estudiantes, mapa de la universidad, entre otros servicios. (Muñoz & Cabrera, 2017).

El anterior ejemplo resulta especialmente interesante para la presente investigación debido a que, en el objetivo planteado en ella, evalúa directamente el problema de resolver necesidades de información en la comunidad universitaria. Además, permite diferenciar claramente las partes que generalmente componen el funcionamiento de las aplicaciones. Por una parte, se debe tener una infraestructura que permita gestionar y proporcionar información, interactuando directamente con los datos, a través de, por ejemplo, gestores de bases de datos. A esto se le conoce comúnmente como backend. Por otra parte, se debe presentar la información disponible a los usuarios finales, pero debe hacerse mediante interfaces que faciliten su comprensión y les proporcionen la mayor utilidad posible a esos usuarios. A esto se le conoce comúnmente como frontend. Comprender esto es crucial para la presente investigación, puesto que ambas partes, tanto el backend como el frontend son indispensables en los procesos de entrega y flujo de información.

Es importante señalar también que estos ejemplos no son aislados, y muchas Universidades hacen uso de la tecnología para apoyar sus procesos, y más que una innovación, el hacerlo se ha convertido en una obligación y no es de extrañar que para esto se recurra al uso de los dispositivos móviles, pues estos son cada vez más usados y acompañan a las personas cotidianamente. Además, todas estas investigaciones buscan resolver necesidades de acceso a información, aprovechando las nuevas herramientas para garantizarle a los usuarios el acceso a los servicios que necesitan de la forma más rápida y fácil. De modo que, al tener la intención de compartir información y servicios, estos deben ser provistos de alguna forma, aquí es donde entra el concepto de API.

Marco teórico

API: Interfaz de Programación de Aplicaciones

Detrás de todas las aplicaciones mencionadas hasta el momento, haciendo posible su funcionamiento, están las API. Esto se debe a que prácticamente todo el código que le da vida a las aplicaciones, de algún modo hace uso de llamados a API, ya sea por el uso de librerías, framework, servicios externos o simplemente acceder a recursos del sistema (Myers & Stylos, 2016). Desde los primeros días de la informática se ha hecho uso de las API, inicialmente como bibliotecas en entornos locales dentro de los sistemas operativos y con los años las API empezaron a extenderse más allá de estos entornos locales convirtiéndose para el año 2000 en una tecnología muy importante en la integración remota de datos (Red Hat).

¿Qué es una API?

Una API es un conjunto de definiciones y protocolos que se utilizan para desarrollar e integrar el software de las aplicaciones (Red Hat), estas definiciones y protocolos ofrecen un lenguaje común, lo que permite que dos aplicaciones puedan entenderse e intercambiar información entre sí, a forma de analogía, se puede pensar en esta interacción como algo similar a una interfaz de usuario que permite la interacción entre un software y una persona, con la diferencia de que la interacción en este caso se da entre aplicaciones o programas informáticos, para acceder, por ejemplo, a los servicios del sistema operativo, bibliotecas de software, u otros sistemas (BBVA).

En este sentido, uno de los principales aportes de las API es que permiten el uso de funciones o infraestructuras ya existentes, evitando así tener que implementar varias veces la misma funcionalidad, sabiendo que esta podrá ser satisfecha por una API ya existente (Merino, 2014). De hecho, las API permiten que distintas aplicaciones se comuniquen entre sí sin necesidad de saber cómo están

implementadas, lo que facilita la integración de productos y tecnologías de manera sencilla, permitiendo ahorrar tiempo y dinero (Red Hat). Esta característica les permite a los desarrolladores innovar y experimentar ahorrando tiempo, reutilizando, compartiendo y sin la obligación de aprender cómo funciona todo lo que usan, no necesitan saber cómo se crearon las API, ni que costo tuvieron, sino que tan fácil es hacer uso de ellas (Jensen, 2015).

Las API son utilizadas como parte estratégica para una gran variedad de modelos de negocios, incluyendo los ya mencionados, relacionados con las aplicaciones y servicios de las grandes empresas tecnológicas; tal es su utilidad y su extensión que en algunos casos se usan como modelo de negocio en sí mismas, un ejemplo claro es Twitter que a través de su API puede generar 10 veces más tráfico que su propia aplicación. La empresa decidió prestar este servicio, que sirve como mediador de publicaciones, permitiendo a cualquiera que desee hacerlo, el desarrollo de interfaces de usuario, y lo hacen porque esto crea valor no solo para el desarrollador que usa la API, sino para la propia empresa (Jensen, 2015).

Es así como, dentro del panorama empresarial, se hace indispensable el uso de las API para que las empresas puedan adaptarse de manera rápida a las necesidades de un mercado cada vez más exigente y cambiante, a tal punto que algunos empiezan a hablar del concepto de Economía de las API (Jensen, 2015). Esto se evidencia en el enorme valor comercial que tienen las API, al permitir rentabilizar los datos de una empresa, aprovechando las API públicas para compartir información con sus partners y usuarios externos, lo que resulta principalmente en que se puedan crear canales de ingresos adicionales, expandir el alcance de la marca y facilitar la innovación y desarrollo gracias a la colaboración de externos (Red Hat).

SOAP Y REST

A medida que se han difundido las API, han existido muchos esfuerzos por simplificar su diseño y facilitar su implementación. Como consecuencia de estos esfuerzos se han desarrollado especificaciones y protocolos orientados a estandarizar y establecer pautas para el desarrollo API. Una de estas especificaciones es el Protocolo de Acceso a Objetos Simples, más conocido como SOAP, que es un protocolo de comunicación que se basa en mensajes en formato XML, generalmente sobre el protocolo HTTP. Otra especificación para la implementación de las API es la Transferencia de Estado Representacional REST, a las API web que funcionan bajo esta especificación se les llama API de RESTful. A diferencia de SOAP las API RESTful representan una arquitectura que especifica la forma como están estructurados los recursos y cómo se debe interactuar con ellos a través de la API (Red Hat).

SOAP: Protocolo de Acceso a Objetos Simples

Es un protocolo que permite la creación de servicios web sin estado, estableciendo un estándar que define cómo dos objetos de diferentes procesos pueden comunicarse; fue creado principalmente por IBM y Microsoft y está actualmente auspiciado por la W3C, siendo uno de los protocolos más utilizados en la creación de servicios Web (EcuRed). Para la comunicación se utilizan mensajes encapsulados en XML a través del protocolo de transporte TCP, lo cual permite que se puedan utilizar varios protocolos de aplicación como: HTTP, SMTP o JMS. Este protocolo es altamente estandarizado por lo tanto existen reglas muy específicas y concretas que definen la comunicación entre el cliente y el servidor, lo que se representa como un contrato detallado que el formato de datos que se envía a través de XML debe cumplir con exactitud (Chakray).

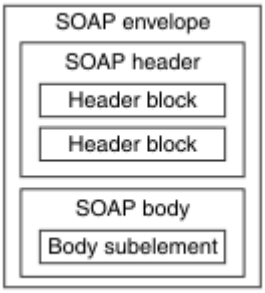
Gráfica 7 ***Ejemplo de mensaje SOAP.*** *(IBM).*

```
<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
                env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
                env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <n:name>Fred Bloggs</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2007-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2007-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference></p:seatPreference>
   </p:return>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

Fuente: SOAP

SOAP se basa en el metalenguaje XML el cual propone un formato que define qué unidades son necesarias para dar como resultado un documento XML, y es este el que se transmite como mensaje (Ionos, 2020), en la gráfica 7 se puede observar un ejemplo.

Gráfica 8 ***Estructura de un mensaje SOAP****. (IBM).*



Fuente: Elaboración propia a partir de datos de SOAP

Dentro de la estructura que compone un mensaje SOAP se pueden distinguir 3 elementos principales:

- Envelope: es el elemento de mayor jerarquía dentro del documento XML, indica la raíz de cada mensaje SOAP (IBM).
- Header: en esta sección se incluye información específica del mensaje SOAP, por ejemplo, la información de autenticación. Este elemento es opcional (AEPI).
- Body: contiene la información para el destinatario final del mensaje. SOAP define una sección secundaria para este elemento llamada Fault que es una sección que se utiliza para informar de cualquier error que suceda en el servidor.

En la gráfica 8 se puede observar conceptualmente cómo se estructura cada uno de los elementos ya mencionados dentro de un mensaje SOAP. En las gráficas 9 y 10 se puede observar un ejemplo sencillo de un mensaje SOAP, donde se muestra la estructura XML de una consulta y su respectiva respuesta.

Gráfica 9 ***Consulta XML.*** *(Sandy, 2014).*

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   </soap:Header>
   <soapenv:Body>
      <consultarProductosRequest xmlns="http://elements.blogsoa.com/Productos">
         <codigoProducto></codigoProducto>
         <nombreProducto>Silla Ejecutiva</nombreProducto>
         <categoria>Artículos de oficina</categoria>
         <almacen>Centro</almacen>
      </consultarProductosRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Fuente: ejemplo de un mensaje SOAP

Gráfica 10 ***Respuesta XML.*** *(Sandy, 2014).*

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   </soap:Header>
   <soapenv:Body>
      <consultarProductosResponse xmlns="http://elements.blogsoa.com/Productos">
        <listaProductos>
            <producto>
                <codigoProducto>1KL979GY</codigoProducto>
                <nombreProducto>Silla Ejecutiva Piel</nombreProducto>
                <categoria>Artículos de oficina</categoria>
                <almacen>Centro</almacen>
                <precio>3800.36</precio>
                <cantidadDisponible>5</cantidadDisponible>
            </producto>
            <producto>
                <codigoProducto>1KL368WS</codigoProducto>
                <nombreProducto>Silla Ejecutiva Estándar</nombreProducto>
                <categoria>Artículos de oficina</categoria>
                <almacen>Centro</almacen>
                <precio>1756.99</precio>
                <cantidadDisponible>14</cantidadDisponible>
            </producto>
        </listaProductos>
      </consultarProductosResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Fuente: ejemplo de un mensaje SOAP

La descripción de un servicio web específico se hace mediante la definición del WSDL: Lenguaje de Descripción del Servicio Web. Como se muestra en el ejemplo de la gráfica 11 este lenguaje también se basa en XML y se usa para definir al cliente todos los procedimientos y funciones a las que puede acceder, y la forma como debe llamar al servicio web (Ionos, 2020). En otras palabras, el WSDL describe la interfaz que proporciona el proveedor del servicio web y es una guía que un cliente utiliza para componer una solicitud de servicio web (IBM).

Gráfica 11 ***Ejemplo de WSDL.*** *(Thotapalli, 2019).*

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/definitions"
          xmlns:tns="http://example.com/stockquote/definitions"
          xmlns:xsd1="http://example.com/stockquote/schemas"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

   <import namespace="http://example.com/stockquote/schemas"
           location="http://example.com/stockquote/stockquote.xsd"/>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>
</definitions>
```

Fuente: ejemplo SOAP

Se puede señalar que SOAP es un protocolo con una documentación extensa, que requiere la existencia de un contrato entre el cliente y el servidor para funcionar. Este contrato también llamado WSDL como se mencionó anteriormente describe de forma detallada cómo el servicio web está implementado y los estándares que el cliente debe cumplir para poder acceder a los servicios (IBM). Lo anterior le otorga cierta rigidez y complejidad, que podría tener consecuencias contraproducentes en la adaptabilidad, costos de desarrollo y mantenimiento.

REST: Transferencia de Estado Representacional

Este es un estilo de arquitectura que se establece a la hora de realizar una comunicación entre un cliente y un servidor a través del protocolo de comunicación HTTP, y permite compartir información o hacer cualquier tipo de operación sobre los datos en cualquier formato posible. Generalmente los formatos más utilizados para compartir información son XML y JSON (BBVA, 2016).

Este enfoque en el desarrollo de proyectos y servicios web fue establecido por Roy Fielding, e inició una nueva era en el intercambio y manipulación de datos en servicios de internet. El término REST se originó por primera vez en el año 2000 en la tesis doctoral de Fielding llamada “Estilos arquitectónicos y el diseño de arquitecturas de software basadas en la red” (BBVA, 2016).

Al tratarse de un estilo arquitectónico y no de un protocolo, no existe ningún estándar oficial para una API web REST (Red Hat). Las API que funcionan bajo este estilo de arquitectura se denominan API RESTful y deben cumplir con 6 limitaciones, que definen si una API es RESTful o no (BBVA, 2016):

- Arquitectura cliente-servidor: la arquitectura REST se basa principalmente en un sistema cliente servidor, donde ambos son independientes uno del otro o están débilmente asociados. En otras palabras, al cliente no le interesa conocer los detalles de cómo está implementado el servidor, y al servidor no le interesa saber cómo son utilizados los recursos que comparte al cliente (Ordoñez).
- Sin estado: cada petición debe ser independiente una de la otra, y los datos de sesión siempre deben ser almacenados por el cliente y no en el servidor (Ordoñez).
- Capacidad de cache: el sistema debe admitir el almacenamiento en caché, evitando generar conexiones nuevas entre el cliente y el servidor para obtener el mismo recurso (Red Hat).
- Sistema en capas: la interacción entre el cliente y el servidor puede estar mediada por capas, que ofrecen diferentes funcionalidades que pueden mejorar la escalabilidad, rendimiento y seguridad (Red Hat).
- Interfaz uniforme: define una interfaz genérica que administra cada interacción entre el cliente y el servidor de manera uniforme separando la arquitectura, permitiendo que ambas partes evolucionen de manera independiente.

- Código de demanda: esta es una característica opcional que ofrece la posibilidad de extender las capacidades que tiene el cliente transfiriendo código desde el servidor que puede ser ejecutado por el cliente (Red Hat).

Los servicios web que funcionan bajo estas especificaciones están fuertemente ligados al protocolo HTTP, pues utiliza todos los métodos que proporciona este protocolo de comunicación: GET, POST, PUT, DELETE, además de sus códigos de respuesta nativos. De esta manera REST se ha convertido en una alternativa más simple a los protocolos de intercambio de datos más comunes como SOAP, que si bien son estándares que disponen de una gran capacidad al mismo tiempo son muy complejos. Por esta razón las API de RESTful son cada vez más frecuentes que SOAP ya que las empresas buscan contar con una solución más sencilla para poder compartir y manipular datos (BBVA, 2016).

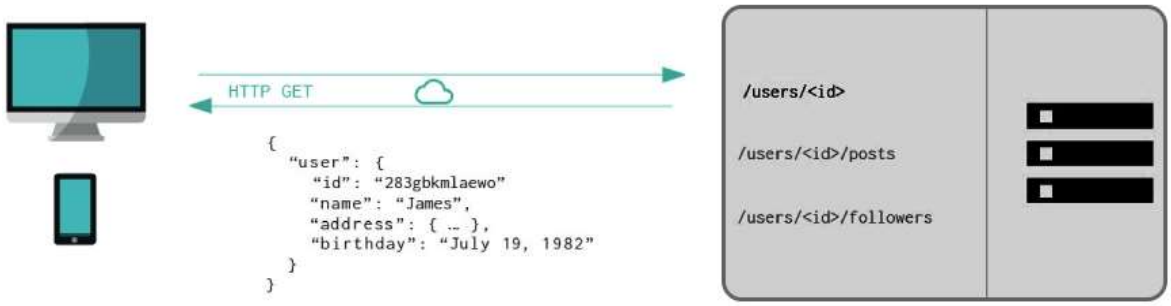
Uno de los principios que define a una API RESTful consiste en utilizar URI para identificar cada recurso disponible dentro de la arquitectura REST. Una URI además de permitir identificar de manera única un recurso también permite localizarlo para acceder a dicho recurso y poder realizar cualquier acción permitida sobre los datos (Marqués, 2013).

Gráfica 12 ***Ejemplo de API RESTful****. (Álvarez, 2019).*

Fuente: Elaboración propia a partir de datos de API

En la gráfica 12 se puede observar un ejemplo de un recurso compartido por una API RESTful en donde cada petición que el cliente realiza a través de HTTP va acompañada de un verbo que responde a una operación específica.

Gráfica 13 ***Ejemplo de petición GET a API REST.*** *(Williams, 2019).*



Fuente: Consulta de un usuario específico

Como un ejemplo de recuperación de datos se presenta la gráfica 13 que describe de manera sencilla una petición donde un cliente consulta información de un usuario específico accediendo al recurso /users/<id> utilizando el método GET, y la respectiva respuesta en formato JSON que el servidor REST devuelve al cliente.

REST es una alternativa más simple, que se basa en definir un estilo de arquitectura antes de ceñir a las empresas a un estándar más complejo como el que propone SOAP para el desarrollo de servicios web. A pesar de que SOAP es un protocolo robusto que ha sido aceptado por muchas empresas y que tiene mucho tiempo de ser utilizado, REST ha ganado fuerza precisamente por ser más sencillo y flexible, por esta razón se utilizan servicios REST con más frecuencia para gestionar datos de manera masiva, siendo actualmente una de las tecnologías más populares para el desarrollo API (Chakray).

GRAPHQL

GraphQL es un lenguaje de queries que sirve para manejar la comunicación entre aplicaciones y servidores. Fue creado por Facebook, y usado por primera vez en su

aplicación móvil para IOS en el año 2012; su documentación fue posteriormente liberada en el año 2015, como una versión Open Source modificada. Actualmente es usado por muchas compañías alrededor del mundo debido a su gran utilidad y ventajas con respecto a las tecnologías pasadas, brindando una alternativa a los protocolos y especificaciones que hasta el momento no brindaban una solución óptima en la transmisión de información entre el cliente y el servidor (McGarvie & Tracey, 2020).

Según testimonios de Lee Byron, Dan Schafer y Nick Schrock, los cocreadores de GraphQL, este surge a raíz de la necesidad de una herramienta que permita un desarrollo de versiones móviles nativas para la red social Facebook, algo que el equipo de desarrolladores del gigante tecnológico había fallado en lograr, pues la transición de la implementación web a la móvil involucraba adaptarse a cosas para las cuales el código y las tecnologías de las que disponían no estaban diseñadas (McGarvie & Tracey, 2020). Facebook evidenció la necesidad de contar con un modelo de búsqueda que permitiera optimizar la obtención de datos de una API y que ayude a mejorar la latencia de las solicitudes entre los clientes móviles y la infraestructura de backend de Facebook. Además, evidenciaron que la cantidad de conexiones que se estaban realizando para mostrar cada vista de sus aplicaciones eran excesivas, lo cual no era viable pensando en la cantidad de usuarios que se manejaban.

Como se ha mencionado anteriormente, durante muchos años REST se estableció como la forma más popular para exponer los datos de un servidor, sin embargo, este protocolo se definió en condiciones diferentes a las actuales, y estas evolucionan constantemente, así pues, las aplicaciones son cada vez más complejas y de igual forma el ritmo de desarrollo ha aumentado. En otras palabras, el contexto en el que se utilizan las API ha cambiado en los últimos años, y por ello, el diseño de estas ha tenido que cambiar conforme evoluciona el desarrollo de las aplicaciones. Entre las principales razones que obligaron a estos cambios destacan

el aumento del uso de los dispositivos móviles, la variedad de diferentes marcos y plataformas frontend y la expectativa de un desarrollo ágil (Prisma, s.f.).

En primer lugar, el aumento del uso de los dispositivos móviles hizo necesario optimizar al máximo los datos que se transfieren por la red para no afectar el rendimiento de estos dispositivos de baja potencia. Por otra parte, la heterogeneización de los marcos de trabajo para clientes y las diferentes plataformas donde se ejecutan, ha dificultado la creación y el mantenimiento API que puedan ajustarse a los requerimientos de todos. Y, por último, la expectativa de un desarrollo ágil se convirtió en una característica muy importante para las empresas, pues les permite ser competitivas mediante iteraciones rápidas de desarrollo, actualizaciones frecuentes y despliegues continuos de nuevos productos (Prisma, s.f.).

Es en este contexto, y pensando en estas necesidades es que surge GraphQL como propuesta, ofreciendo a los clientes una manera más directa, sencilla y potente para obtener exactamente los datos que necesita a través de un protocolo potente y dinámico (Aplyca). Utilizando este lenguaje un cliente obtiene datos de forma declarativa y los resultados son predecibles, es decir, el cliente especifica que es lo que necesita obtener de una API y cómo lo quiere, y el servidor siempre va a componer sus respuestas respetando la estructura enviada por el cliente (Prisma, s.f.). En la gráfica 14 se puede ver un ejemplo de este lenguaje de consulta y su respectiva respuesta.

Gráfica 14 ***Ejemplo de consulta GraphQL con su respuesta.***

Fuente: ***consulta GraphQL***

Especificación GraphQL

Como se ha dicho anteriormente GraphQL consiste en una especificación que describe de manera estándar como implementar este lenguaje de consulta y motor de ejecución, estableciendo los requisitos normativos y de conformidad que deben cumplirse (GraphQL Specification Versions, 2015). Dado que GraphQL no es una librería sino más bien un manual de instrucciones esta es una especificación que puede ser implementada en cualquier lenguaje de programación (McGarvie & Tracey, 2020). A continuación, se describirán los elementos que conforman la columna vertebral de esta especificación.

Esquema GraphQL

La columna vertebral de un servidor GraphQL se basa en el diseño de un esquema que es el que define las capacidades del servicio. En la gráfica 15 se muestra un ejemplo de un esquema que será utilizado para describir y entender de mejor manera cada uno de los elementos que lo componen. Este esquema está definido bajo el lenguaje de definición de esquemas GraphQL también conocido como SDL. Un esquema define principalmente los modelos de datos que serán expuestos a los clientes a través de tipos, las directivas que admite y los tipos de operación raíz para cada operación: consulta, mutación y suscripción las cuales determinan el lugar donde empiezan las operaciones (GraphQL Specification Versions, 2015).

Gráfica 15 ***Esquema GraphQL.*** *(Helfer, 2016).*

```
type Author {
  id: Int!
  name: String
  posts: [Post]
}

type Post {
  id: Int!
  title: String
  text: String
  author: Author
}

type Query {
  getAuthor(id: Int): Author
  getPostsByTitle(titleContains: String): [Post]
}

schema {
  query: Query
}
```

Fuente: este proyecto

Tipos

Es la unidad fundamental que compone un esquema GraphQL y se pueden agrupar de manera jerárquica. Los tipos de clasifican en: tipos de envoltura, tipos de entrada y salida. Los tipos de envoltura son los tipos que otorgan a otros tipos una capa de validación o que modifican su naturaleza (GraphQL Specification Versions, 2015).

En GraphQL se especifican dos tipos de envoltura. En el primero GraphQL puede definir un campo que representa una lista de un tipo, en cuyo caso se proporciona el tipo List que envuelve al tipo que será representado como una lista. Otro tipo de envoltura es el tipo no nulo, que es un tipo que envuelve a otro tipo para denotar que el valor resultante nunca será nulo. Los tipos que no son de envoltura se denominan tipos con nombre (GraphQL Specification Versions, 2015).

Los tipos de entrada y salida son los tipos que se utilizan en GraphQL para describir los valores que se aceptan para argumentos, variables y valores de respuesta. GraphQL especifica seis tipos de definiciones de tipos con nombre y dos tipos de ajuste que son los tipos de envoltura (GraphQL Specification Versions, 2015).

Escalar

Son tipos que retornan valores de datos primitivos. Tomando el concepto de que las respuestas de un servidor GraphQL toman la forma de un árbol jerárquico, los escalares serían las hojas del árbol. GraphQL proporciona escalares incorporados que se usan para devolver primitivos dependiendo del tipo de dato que se quiera representar en el esquema. Estos tipos de escalares que ofrece la especificación GraphQL inicialmente son: Int, Float, String, Boolean, ID. En la gráfica 15 se identifica a este tipo en los campos o atributos que componen el tipo Author los cuales son id y name, y en el tipo Post id, title y text (GraphQL Specification Versions, 2015).

Objetos

Las consultas GraphQL son jerárquicas y compuestas las cuales describen un árbol de información y como se dijo anteriormente mientras que los tipos escalares describen los valores de las hojas en estas consultas jerárquicas, los objetos representan un elemento de un nivel más superior o intermedio. Estos objetos representan una lista de campos con nombres, se puede decir que son los atributos del objeto, los cuales producen un valor específico. Las respuestas de estos objetos se serializan a manera de un diccionario donde los nombres de los campos se utilizan como llaves (key), y el resultado que se devuelve para dicho campo es el valor, ordenados de acuerdo a como se especifica en la consulta (GraphQL Specification Versions, 2015).

Tomando nuevamente el ejemplo de la gráfica 15 el tipo Author es un tipo objeto donde id es un campo tipo escalar que devuelve un Int valor, name es un campo tipo escalar que devuelve un String valor y posts es un campo tipo objeto cuyos campos se declaran en el tipo objeto Post. Los campos de un tipo objeto pueden

ser de cualquier tipo que se especifique dentro del esquema GraphQL (GraphQL Specification Versions, 2015).

Argumentos de campo. Los campos de un objeto conceptualmente son funciones que devuelven un valor, y son utilizados para especificar aún más el valor de retorno de un campo. En la imagen x se puede observar el argumento *id* de tipo escalar Int del campo getAuthor en el tipo objeto raíz Query que se utiliza para pedir un autor específico por id. Estos argumentos pueden ser opcionales u obligatorios dependiendo como se lo defina en el esquema (GraphQL Specification Versions, 2015).

Desuso de campo. Puede haber ocasiones en que un campo de un objeto se vuelva obsoleto y que para no afectar la integridad de aplicaciones que aún pueden seguir usándolo siguen siendo campos válidos para consulta, pero por temas de documentación en la definición de esquema de tipos se usa la directiva *@deprecated* para indicar a los usuarios que ese campo está en desuso (GraphQL Specification Versions, 2015), como se muestra en la gráfica 16.

Gráfica 16 ***Ejemplo de campo en desuso en esquema GraphQL.*** *(GraphQL Specification Versions, 2015).*

```
type ExampleType {
  oldField: String @deprecated
}
```

Fuente: Este proyecto

Interfaces

Representa una lista de campos con nombres y argumentos que los objetos GraphQL pueden implementar para heredar los campos de dichas interfaces (GraphQL Specification Versions, 2015), como se muestra en la gráfica 17.

Gráfica 17 ***Ejemplo de interfaces en esquema GraphQL****. (GraphQL Specification Versions, 2015).*

```
interface NamedEntity {
  name: String
}

interface ValuedEntity {
  value: Int
}

type Person implements NamedEntity {
  name: String
  age: Int
}

type Business implements NamedEntity & ValuedEntity {
  name: String
  value: Int
  employeeCount: Int
}
```

Fuente: datos del proyecto

Enumeraciones

Las enumeraciones representan al igual que los escalares las hojas en el árbol jerárquico, pero a diferencia es que estas definen un conjunto de valores posibles (GraphQL Specification Versions, 2015). La grafica 18 representa un ejemplo de la definición de una numeración.

Gráfica 18 ***Ejemplo de enumeración en esquema GraphQL****. (GraphQL Specification Versions, 2015).*

```
enum Direction {
  NORTH
  EAST
  SOUTH
  WEST
}
```

Fuente: datos del proyecto

Objetos de entrada

Los campos admiten argumentos para configurar el comportamiento de su respuesta, y por lo general estos argumentos son escalares o numeraciones, pero hay situaciones en que se requiere crear argumentos más complejos y para ello se utilizan objetos de entrada que definen un conjunto de campos de entrada los cuales pueden ser escales, numeraciones y otros objetos de entrada, permitiendo obtener argumentos más complejos (GraphQL Specification Versions, 2015). Como ejemplo se muestra la gráfica 19.

Gráfica 19 ***Ejemplo de objeto de entrada en esquema GraphQL****. (GraphQL Specification Versions, 2015).*

```
input Point2D {
  x: Float
  y: Float
}
```

Fuente: datos del proyecto

Listas

Son un tipo de colección especial que declara el tipo de cada elemento en la lista. Para denotar que un campo es de tipo lista, se debe envolver su tipo base entre corchetes, así posts: [Post] (GraphQL Specification Versions, 2015). En el ejemplo de la gráfica 15 el campo posts se define como lista, el cual retorna la lista de posts de cada autor.

No nulo

Por defecto todos los tipos en GraphQL son anulables, es decir que el valor nulo es una respuesta válida para cualquiera de los tipos. Para definir un campo no nulo en el esquema se utiliza un signo de exclamación (¡) de la siguiente manera: <name:String!>. En el ejemplo de la gráfica 15 se puede observar que el campo que define el id tanto del tipo objeto Author y Post son no nulos.

Tipos de operaciones de raíz

Un esquema GraphQL debe definir el tipo de operación raíz para cada tipo de operación a través de las cuales el usuario va a interactuar con el servicio. Las operaciones que admite GraphQL son: consultas, mutación y suscripción.

- Para consultas se debe proveer al esquema del tipo raíz query y debe definirse como un tipo objeto. En la gráfica 15 se tiene el tipo objeto Query que es la raíz que se especifica en el esquema GraphQL con el nombre del campo query, el cual indica el inicio para la consulta de autores y posts.
- Las mutaciones se utilizan para realizar operaciones de escritura seguidas de una búsqueda, este tipo es opcional, si se proporciona debe ser un tipo objeto.
- Las suscripciones son solicitudes de larga duración que se mantienen para recibir datos en respuesta a eventos de origen. Esta operación establece una conexión estable entre el cliente y el servidor y a diferencia de las consultas y

mutaciones las respuestas se envían al cliente al momento de suceder un evento particular en el esquema en tiempo real.

Funciones de resolución

GraphQL hace una clara distinción entre la estructura de su API y su comportamiento. En cuanto a su estructura, como se revisó anteriormente se trata de definir el esquema que describirá de manera abstracta las capacidades del servidor. Las funciones de resolución o resolutores son componentes clave que determinan el comportamiento del servidor (Burk, GraphQL Server Basics: GraphQL Schemas, TypeDefs & Resolvers Explained: Prisma, 2017).

De manera general un servidor GraphQL debe implementar una función de resolución por cada campo que defina su esquema. Estas funciones serán las encargadas de obtener los datos que serán devueltos por su respectivo campo, dado que GraphQL en esencia es un conjunto de campos, para resolver la información que se necesita retornar de una consulta, el servidor GraphQL se encarga de invocar todas las funciones de resolución para cada uno de los campos que se especifica en la consulta (Burk, GraphQL Server Basics: GraphQL Schemas, TypeDefs & Resolvers Explained: Prisma, 2017).

Ventajas de GraphQL

Como se mencionó antes, el anuncio de Facebook de sus planes para abrir el código fuente de GraphQL fue en el 2015, por lo tanto, es una tecnología que aún es relativamente nueva, en especial en comparación con otros protocolos y arquitecturas como SOAP y REST. Sin embargo, GraphQL fue ampliamente aceptado en la comunidad, y desde que fue liberado, la propia comunidad se ha encargado de crear librerías open source para múltiples lenguajes de programación, todas ellas basadas en las especificaciones de GraphQL. Incluso muchas compañías identificaron el potencial de esta herramienta, y la integraron en el

desarrollo de sus API en muy poco tiempo. Este fue el caso de grandes empresas como GitHub, Twitter, Yelp y Shopify, entre muchas otras (McGarvie & Tracey, 2020). Esto se debe a que GraphQL ofrece características muy favorables para el desarrollo de aplicaciones.

Las API GraphQL se basan en un esquema fuertemente tipado

Uno de los principales problemas que tienen los desarrolladores con el uso de las API es que no cuentan con un contrato sólido que defina de manera clara el uso de sus operaciones, provocando que muchas veces tengan que trabajar con documentaciones desactualizadas desconociendo si las funciones que están usando son compatibles con las operaciones que ofrece una API. Sin embargo, la columna vertebral de una API GraphQL se basa en la especificación inicial de un esquema el cual es un contrato infalible que especifica las capacidades de una API, definiendo todos los elementos al que puede tener acceso un cliente como por ejemplo las operaciones permitidas, modelos de datos (tipos), argumentos de entrada y posibles respuestas (Burk, Top 5 Reasons to Use GraphQL: Prisma, 2018).

Las API GraphQL manejan un tipado fuerte, definido desde el principio en su esquema, esto significa que en este tipo de API no se permiten violaciones en los tipos de datos de entrada como argumentos ni de salida en las respuestas. Gracias a esta característica los desarrolladores pueden obtener muchos beneficios que no son posibles con API que no manejan esquema. Por ejemplo, las herramientas de compilación podrían validar cualquier error existente en la comunicación con la API en tiempo de compilación, incluso durante el desarrollo en tiempo real desde su editor. Otro beneficio derivado, que cambia por completo la manera como se desarrolla las API, es que los desarrolladores ya no necesitan documentar manualmente una API si no que pueden generar automáticamente esta

documentación directamente del esquema que la define (Burk, Top 5 Reasons to Use GraphQL: Prisma, 2018).

El cliente define lo que recibe

Esta es una de las ventajas principales que ofrece GraphQL, el cliente puede recuperar exactamente los datos que necesita, ya que puede declarar a través del lenguaje de consulta la forma de los objetos que serán devueltos como respuesta por la API. Esta característica, le da el control al cliente sobre los datos que necesita obtener de la API y de este modo evita caer en dos problemas comunes en el uso de API, generalmente aquellas basadas en REST: overfetching y underfetching (Burk, Top 5 Reasons to Use GraphQL: Prisma, 2018).

Como se mencionó anteriormente, una API RESTful ofrece diferentes puntos finales a los que el cliente puede acceder, sin embargo, está condicionado a recibir exactamente la estructura de datos definida por esta URI (Burk, Top 5 Reasons to Use GraphQL: Prisma, 2018). En la gráfica 20, se observan tres puntos de entrada de una API RESTful, cada uno de ellos devuelve una información específica. Se analizarán dos ejemplos de posibles consultas hechas por un cliente.

En el primer ejemplo, el cliente necesita consultar el id y el nombre de un usuario, para ello deberá hacer una petición al punto de entrada 1, pero al hacerlo, no va a recibir solo los datos que necesita, sino que, en adición, obtendrá otros datos de cada usuario, como dirección, fecha de cumpleaños, etc. A esto se le conoce como sobrecarga o overfetching, significa que un cliente está obteniendo más datos de los que realmente necesita al consultar los recursos de una API. Esto tiene un impacto negativo en el rendimiento de la aplicación (Burk, Top 5 Reasons to Use GraphQL: Prisma, 2018).

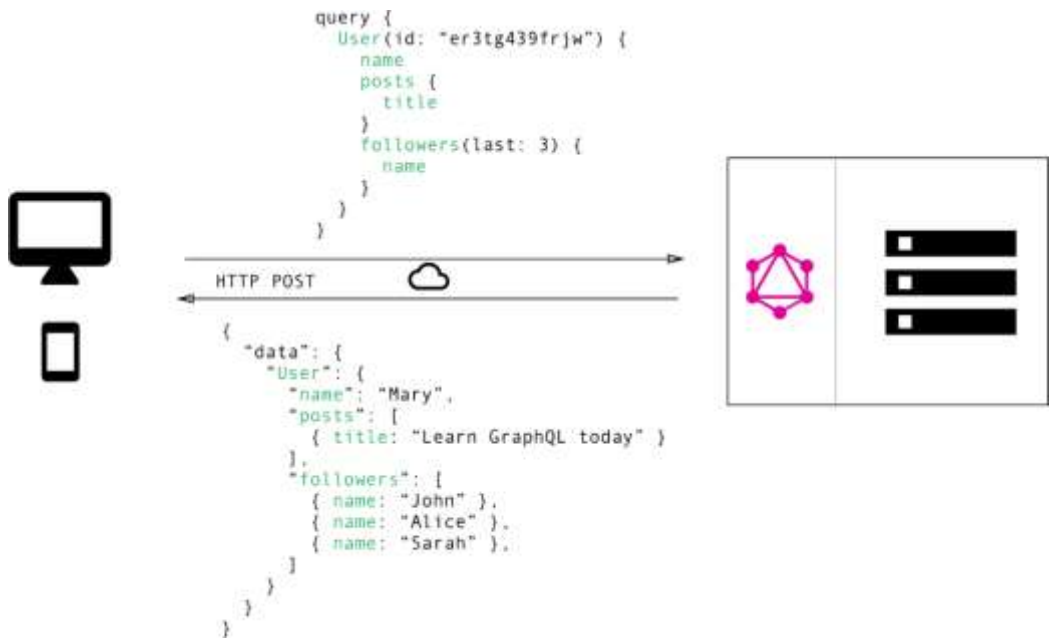
En el segundo ejemplo, el cliente necesita consultar el nombre de un usuario y una lista de los nombres de sus seguidores, para ello, una sola consulta no será suficiente, por ejemplo, al hacer una petición al punto de entrada 1, no se obtendría la información de los seguidores del usuario. A esto se le conoce como captación insuficiente o underfetching, en este caso el cliente no está recibiendo toda la información que necesita de una sola respuesta lo que lo obliga a realizar más peticiones a la API para completar sus requisitos de datos para llenar su interfaz actual (Burk, Top 5 Reasons to Use GraphQL: Prisma, 2018). Para el presente ejemplo el cliente debería hacer dos peticiones, una al punto de entrada 1 y otra al 2, nótese que en ambas peticiones también se incurriría en el problema del overfetching.

Gráfica 20 ***Ejemplo de peticiones a API RESTful****. (Prisma, s.f.)*

Fuente: datos del proyecto

En contraposición, una API GraphQL expone un solo punto de acceso a los datos y además el cliente puede especificar todos los datos que necesita del servidor en una única solicitud utilizando el lenguaje de consulta, como se puede observar en el ejemplo de la gráfica 21.

Gráfica 21 ***Ejemplo de petición a API GraphQL****. (Prisma, s.f.)*



Fuente: datos del proyecto

GraphQL permite el desarrollo rápido de productos

GraphQL proporciona un ambiente propicio para aumentar la productividad de los desarrolladores frontend, y por lo tanto agiliza el desarrollo de productos. Una de las razones de esto es la existencia de variedad de bibliotecas para clientes GraphQL como Apollo, Relay o Urql, desarrollados por la comunidad, que ofrecen herramientas con características de alto nivel de forma gratuita, evitando la necesidad de equipos enteros dedicados a trabajar en la creación de la estructura

necesaria para hacer uso de GraphQL (Burk, Top 5 Reasons to Use GraphQL: Prisma, 2018).

Por otra parte, el desarrollo de una API GraphQL se centra básicamente en la definición de un esquema GraphQL, lo cual significa que cualquier elemento que se necesite adicionar se define primero en el esquema y después se implementan las funciones que se encargaran de resolver la información que se requiere para dicho elemento. Esta característica de un desarrollo basado en esquemas les permite a los desarrolladores frontend ser productivos una vez se haya definido el esquema sin necesidad de contar con una API GraphQL totalmente terminada, por lo cual los desarrolladores frontend y backend pueden trabajar de forma independiente (Burk, Top 5 Reasons to Use GraphQL: Prisma, 2018).

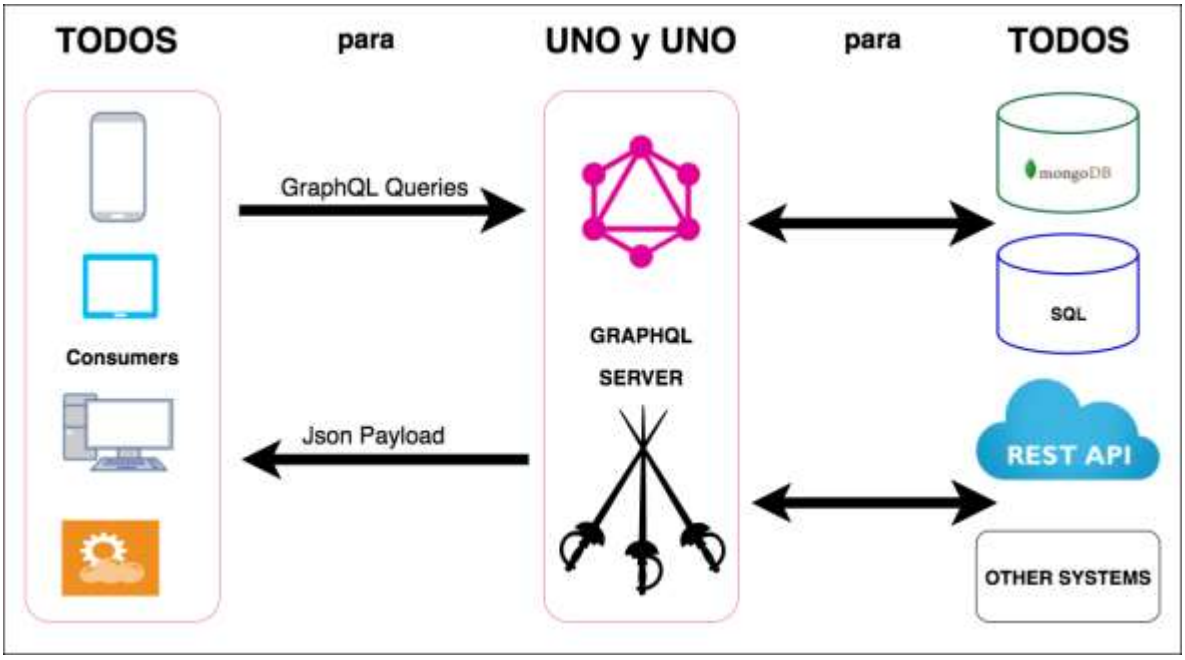
Además, GraphQL admite iteraciones rápidas de un producto, debido a su naturaleza flexible, que le permite a un cliente realizar cambios en alguna interfaz sin la necesidad de realizar ajustes en el backend ya que el cliente es el que especifica los datos exactos que requiere para una interfaz, a diferencia de algunas API REST que ofrecen al cliente toda la información requerida para un vista a través de un punto final que es estático y que no tiene la capacidad de adaptarse a nuevas necesidades de una vista sin realizar los respectivos ajustes en el backend lo que puede disminuir considerablemente la productividad (Prisma, s.f.).

Arquitectura de una API GraphQL

El lema “Todos para uno y uno para todos” de “Los tres mosqueteros” define de cierta manera la esencia de GraphQL, partiendo de la base de que existe un solo punto de acceso a los datos. Analizando la primera parte “Todos para uno” se muestra el “Todos” como los clientes que acceden a las operaciones a través del único punto de entrada, trátese de aplicaciones móviles, web, de escritorio incluso

otras API, y el “uno” sería el servidor GraphQL que respondería a dichas operaciones. En la segunda parte “Uno para todos” el “uno” siempre será el servidor GraphQL y el “Todos” corresponde a las distintas fuentes de recursos de datos que el servidor utilizará para resolver las consultas y operaciones (Garcia Rozas, 2017).

Gráfica 22 ***Todos para uno y uno para todos.*** *(Garcia Rozas, 2017).*



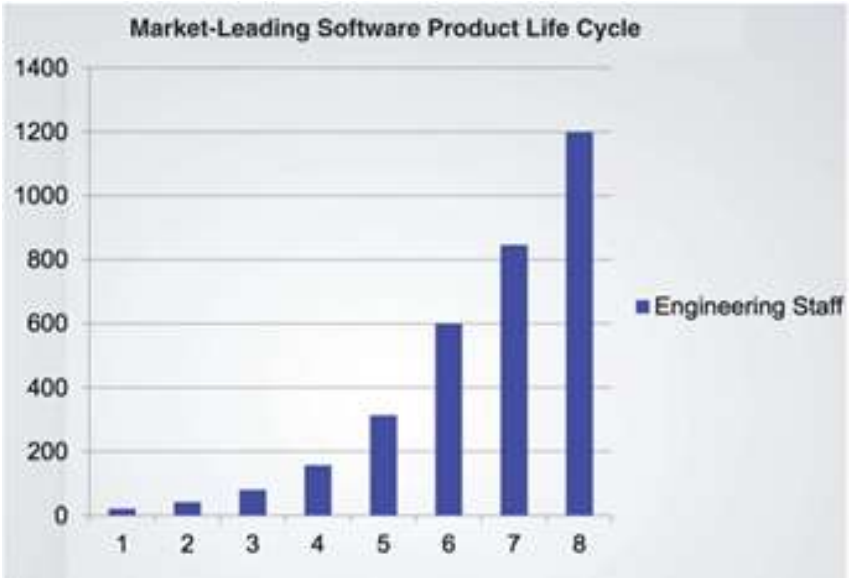
Fuente: datos del proyecto

Entonces GraphQL es un elemento que permite estandarizar la manera cómo se obtienen datos y cómo se realizan diferentes operaciones sobre ellos, ya sea desde al cliente hacia el servidor GraphQL a través del lenguaje de consulta o desde el servidor GraphQL hacia las diferentes fuentes de datos a través de los “resolvers”, y ya que no se trata de una librería o framework propone las pautas de cómo se debe implementar en cualquier lenguaje de programación (Garcia Rozas, 2017), por lo tanto, la integración de aplicaciones y servicios puede llegar a ser mucho más sencilla si se sigue una única forma de hacer las cosas.

Arquitectura de software

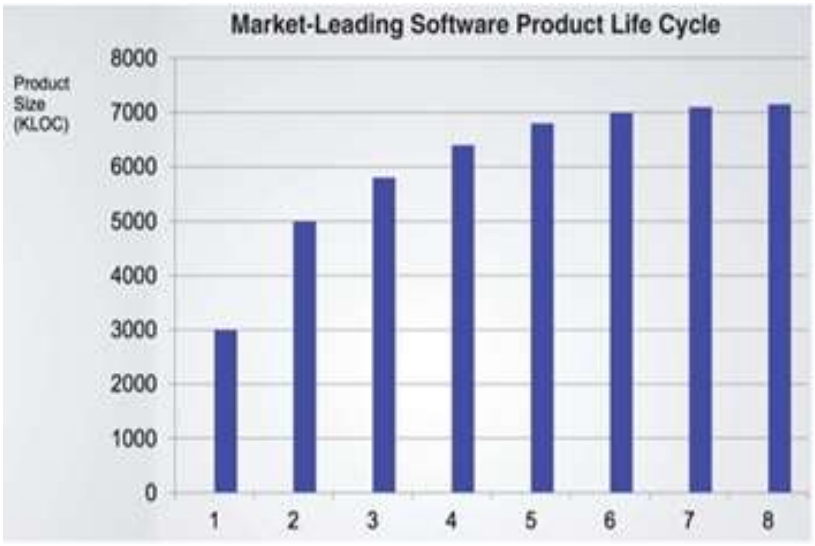
Uno de los problemas más desafiantes a los que se enfrentan las grandes organizaciones es el de escalar y mantener su software, un caso de estudio de una empresa anónima (Martin, 2016) permite evidenciar esto con claridad, en la gráfica 23 se muestra el crecimiento del personal de ingeniería, el encargado de mantener el software de la empresa. Podría entenderse como algo positivo, como un signo de que la empresa está creciendo, sin embargo, si se analiza las siguientes figuras ocurre algo inquietante. En la gráfica 24 la productividad del personal de ingeniería no crece al mismo ritmo, entre más ingenieros, menor productividad, lo cual suena contradictorio. Finalmente, la gráfica 25 indica el costo por línea de código en el mismo periodo de tiempo, cada vez es más costoso para la empresa mantener su software, e implementar nuevas funcionalidades, esto imposibilita la viabilidad de cualquier modelo de negocio.

Gráfica 23 ***Crecimiento del personal de ingeniería****. (Martin, 2016).*



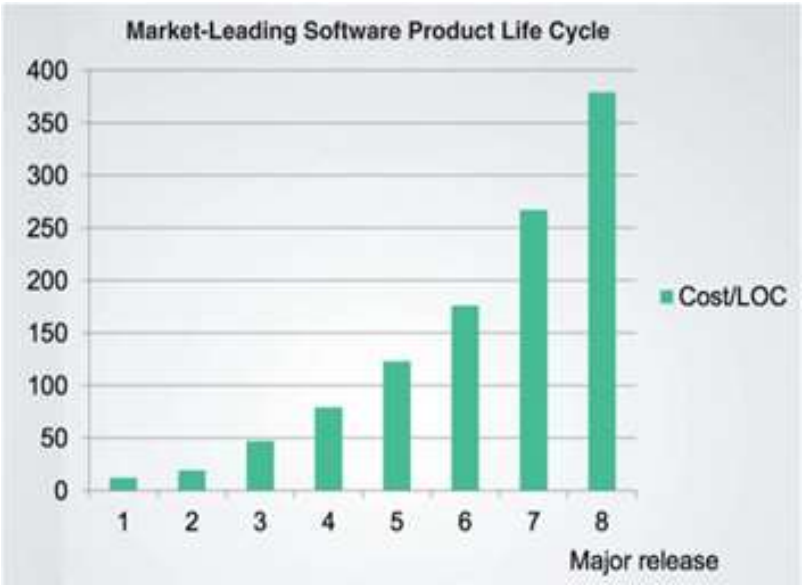
Fuente: datos del proyecto

Gráfica 24 ***Productividad sobre el mismo periodo de tiempo.*** *(Martin, 2016).*



Fuente: datos del proyecto

Gráfica 25 ***Costo por línea de código.*** *(Martin, 2016).*



Fuente: datos del proyecto

Este tipo de escenarios revela la importancia de lograr sistemas flexibles y adaptables, con una buena proyección a futuro; un producto software que funciona bien, pero es imposible de cambiar o el costo de cambiarlo excede sus beneficios,

será inútil en el momento en el que alguna de las partes interesadas exponga un nuevo requisito a solventar.

La Universidad de Nariño no es ajena a este desafío, y la adaptabilidad ha sido una falencia en más de una ocasión, al menos desde el punto de vista de usuarios finales. Un ejemplo claro fue el cambio de ciertos servicios de la antigua plataforma web universitaria, como el de consulta de notas, que inicialmente eran públicos, accesibles a todos, y que fueron trasladados a la consola de administración privada de cada estudiante. En este caso la nueva implementación consistía en una simple redirección al sitio antiguo, sin ningún cambio ni control añadido, completamente vulnerable.

Este tipo de problemas pueden surgir cuando no se hace una buena planeación, teniendo la flexibilidad y escalabilidad del sistema en mente al construir su arquitectura. Y si se deja este problema sin atender, pueden desencadenarse escenarios aún más caóticos en el futuro, incluso algunos en los que la nueva funcionalidad requerida sea tan difícil de implementar que se decida hacer un nuevo sistema para ella, conllevando a muchos gastos innecesarios.

En el libro Clean Architecture (Martin, 2016), se señala que todo producto software está conformado por estructuras de alto nivel que definen lo que hará, y por detalles de bajo nivel, fragmentos específicos de código que soportan y le dan la funcionalidad a la estructura definida; esto forma un tejido, un todo, que es en definitiva lo que el software es y todo ello se construye a partir de decisiones, que tienen como finalidad satisfacer una necesidad específica. Hacer que algo funcione es relativamente fácil, las decisiones necesarias para ello son simples, se puede lograr que algo funcione bien una vez, por mera fuerza de voluntad, pero esto no significa que esté bien hecho, o, en otras palabras, que vaya a funcionar bien o a ser de utilidad en el futuro.

Para evitar este tipo de escenarios catastróficos, es indispensable la aplicación de conceptos de arquitectura de software y código limpio, si el equipo de programadores no lo hace, será imposible lograr sistemas con una buena proyección a futuro; un producto software que funciona bien, pero es imposible de cambiar o el costo de cambiarlo excede sus beneficios, será inútil en el momento en el que alguna de las partes interesadas exponga un nuevo requisito a solventar.

¿Qué es la arquitectura de software?

La base de la arquitectura de software es el código; desde los inicios de la programación informática con Alan Turing en 1938 hasta la actualidad, el código ha atravesado numerosas etapas y cambios, incluyendo la aparición de nuevos lenguajes de programación, y, entre otras cosas, el surgimiento de algo muy relevante para la arquitectura, los paradigmas de programación: programación estructurada, programación orientada a objetos y programación funcional. Estos paradigmas establecen una serie de reglas para la programación, ninguno ofrece nada nuevo, que no existiera antes, por el contrario, limita o para ser más claros, disciplina la escritura de código. Justo esto es lo que define la arquitectura de software, son una serie de reglas, conceptos que invitan a disciplinar el desarrollo de software. Estos conceptos surgen a partir del aprendizaje y las lecciones de muchos años de fracasos y éxitos, y de largos debates en el ámbito de la programación informática, y que permiten entender qué tipo de prácticas funcionan y cuales son perjudiciales en el desarrollo de proyectos y productos software (Martin, 2016).

Principios de Diseño SOLID

Estos principios son una recopilación de Robert C. Martin, y ofrecen una guía sobre cómo acoplar e interconectar los componentes de un sistema de software, se trata de lecciones y conocimientos aprendidos por diversos programadores gracias a su

experiencia, y que fueron unificados en una teoría que se expone a detalle en el libro Clean Architecture (Martin, 2016), donde los desarrolla, equilibra y aplica en diferentes niveles de detalle.

En este sentido, el autor afirma que los “ladrillos” de un sistema de software son sus agrupaciones acopladas de código; si se lo analiza a través de los lentes de la programación orientada a objetos, esas agrupaciones estarían representadas por las clases, sin embargo, este concepto se extiende a cualquier sistema de software. De modo que se puede definir a cada uno de estos ladrillos como un componente, que es la unidad mínima de código que se puede implementar en un sistema, desarrollándose de forma independiente, como una clase, un plugin, etc.

La aplicación de estos principios puede y debe hacerse adaptándolos a los diferentes niveles de complejidad en la estructura de un sistema de software, ya sea en agrupaciones de código más grandes o más pequeñas, y su objetivo es permitir que estas agrupaciones, sean tolerantes al cambio, fáciles de comprender y puedan ser utilizadas como base para construir otros sistemas de software. Se pretende hacer uso de ellos para la construcción de una propuesta de software que goce de esas características y que constituya un aporte en la forma en que la universidad de Nariño gestiona y entrega su información a la comunidad universitaria.

- **Principio de responsabilidad única:** Cada componente de software debe obedecer a las necesidades de un solo “actor”, entendiendo este como las partes interesadas en el proyecto. Por ejemplo, en el caso de una universidad, si se entiende a los estudiantes y a los docentes como dos partes interesadas con necesidades diferentes, sería un error permitir que la definición de una clase intente acoplarlos a ambos al tiempo (Martin, 2016).

- **Principio de apertura y cierre:** Añadir nuevas funcionalidades debería implicar añadir nuevo código, sin modificar el existente. Retomando el ejemplo anterior, si se definió desde el principio una clase independiente para los estudiantes, entonces cuando se requiera implementar una para los profesores, bastaría con agregar nuevo código, sin embargo, sí se intentó unificar ambas partes interesadas en una sola clase, entonces una nueva funcionalidad en una de ellas, implicaría cambios en la clase, que pueden ser incompatibles con la otra (Martin, 2016).

- **Principio de sustitución de Liskov**: Los módulos de software deberían ser sustituibles entre sí (herencia, polimorfismo, versionamiento). Una de las formas para abordar el ejemplo expuesto sería la definición de interfaces, si tanto docentes como estudiantes comparten un núcleo común, una interfaz permitirá definir este núcleo, y mediante la herencia, se puede reutilizar, no sólo en estas, sino en nuevas partes interesadas si se requiere (Martin, 2016).

- **Principio de segregación de la interfaz:** Los módulos de software no deben depender de elementos que no usan. Se debe separar en interfaces, e importar sólo lo necesario. Al definir esas interfaces, es indispensable hacerlo como elementos generales y estables, que no tengan más de lo que requiera para funcionar, de lo contrario perderían su utilidad (Martin, 2016).

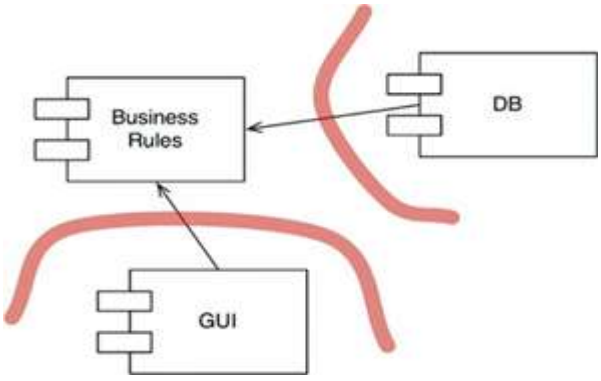
- **Principio de inversión de dependencias:** La dependencia entre módulos de software debe evitarse en la vía de la volatilidad, y favorecerse en la vía de la estabilidad. Una interfaz bien definida es un elemento estable en el sistema, pues no sería necesario cambiarlo. Sin embargo, las clases que usan esa interfaz, entre más específicas sean, pueden convertirse en elementos sumamente volátiles, los nuevos elementos de software deben depender de las

cosas estables y no de aquellas que son muy propensas al cambio (Martin, 2016).

¿Para qué sirve la arquitectura de software?

"La meta de la arquitectura de software es minimizar los recursos humanos necesarios para crear y mantener el sistema requerido" (Martin, 2016). De acuerdo al autor, mediante la aplicación de principios y reglas de arquitectura de software, es posible establecer límites arquitectónicos que permitan la construcción de un software escalable y flexible, cuyas partes puedan intercambiarse por otras sin que haya necesidad de cambiar todo el sistema, por ejemplo, si existe un límite arquitectónico bien definido entre las reglas de negocio y la base de datos usada, entonces se podrá cambiar de motor de base de datos sin afectar el núcleo del sistema y su funcionalidad, esto es extremadamente útil en organizaciones grandes y longevas como una universidad, pues el cambio hacia un motor de base de datos más apropiado para los nuevos desafíos que enfrentan, o una simple actualización en las interfaces de usuario que ofrecen en su software, puede ser un proceso verdaderamente traumático y riesgoso si estas partes están tan intrínsecamente conectadas que al cambiar una línea de código en una, se requiere cambiar otras 100 o mil en otra, y dependiendo del tamaño del sistema, esto puede ser tan costoso que incluso sería más sensato hacer un nuevo sistema.

Gráfica 26 ***Conexión a las reglas de negocio****: (Martin, 2016)****.***



Fuente: datos del proyecto

Un ejemplo muy apropiado de la utilidad de una buena arquitectura es el del sistema Strato de Twitter. Cuando esta empresa decidió hacer uso de la nueva propuesta de consumo de API llamada GraphQL, tuvieron que migrar sus antiguos sistemas basados en REST. Para hacerlo, decidieron adoptar una metodología consistente en generar una capa intermedia entre GraphQL y la data; una arquitectura que flexibiliza el desarrollo de nuevas funcionalidades. Esta capa se llama Strato, y contiene un esquema GraphQL, en base al cual puede devolver los datos en la estructura definida, de modo que cualquier frontend o sistema que los requiera, puede obtener los datos con una consulta GraphQL a Strato. Por su parte, los datos que Strato ofrece, definidos en su esquema, los puede obtener a partir de diversas fuentes, ya sean bases de datos, datos planos o incluso otras API, entre ellas las API REST que se usaban previamente en las aplicaciones. Esto permitió migrar estas API poco a poco, para hacer conexiones directas a las bases de datos si lo requerían, además de agregar nuevas fuentes de información; toda la data agregada quedaría automáticamente esquematizada para GraphQL y seria accesible para cualquier frontend (Beyang, 2017). Twitter marca como uno de sus principales objetivos el ayudar y facilitar a sus desarrolladores la construcción de productos y funcionalidades nuevas de forma rápida y fácil, y mediante la aplicación de estos conceptos de arquitectura de software, definitivamente lo logran.

Python

Python es un lenguaje de programación potente de alto nivel, que ofrece estructuras de datos eficientes y un enfoque simple para la programación orientada a objetos y otros paradigmas, por esta razón es fácil de aprender. La sintaxis que maneja Python, su escritura dinámica y su naturaleza interpretada lo convierte en un lenguaje de programación ideal para el desarrollo rápido de aplicaciones en la mayoría de plataformas (Python).

Entonces una de las principales características de Python es que es de código abierto lo que permite ser utilizado en cualquier escenario y preferido por muchas

escuelas y universidades para ser aprendido. Además, es utilizado de manera intensa por muchas empresas como por ejemplo Google, YouTube Facebook para automatizar procesos y ejecutar tareas tanto en el lado del servidor como en el del cliente aprovechando las múltiples librerías, programas y herramientas que están disponibles de manera gratuita (Robledano, 2019).

Graphene

Graphene es una librería para Python que implementa la especificación GraphQL y ofrece herramientas para la gestión de una API GraphQL. A diferencia de implementaciones realizadas en otros lenguajes de programación donde el eje central para construir una API GraphQL gira entorno a la definición del esquema, en Python se sigue la política de "código primero", es decir, que se escribirá código Python para describir los datos que serán proporcionados por el servidor, representando los tipos que conforman el esquema a través de clases (Graphene Python).

Gráfica 27 ***Ejemplo de API GraphQL con Graphene****. (Graphene Python).*

```python
from graphene import ObjectType, String, Schema

class Query(ObjectType):
    # this defines a Field `hello` in our Schema with a single Argument `name`
    hello = String(name=String(default_value="stranger"))
    goodbye = String()

    # our Resolver method takes the GraphQL context (root, info) as well as
    # Argument (name) for the Field and returns data for the query Response
    def resolve_hello(root, info, name):
        return f'Hello {name}!'

    def resolve_goodbye(root, info):
        return 'See ya!'

schema = Schema(query=Query)
```

Fuente: datos del proyecto

La grafica 27 muestra un ejemplo sencillo de la definición de una API GraphQL utilizando Graphene, donde tipo raíz query está definido por el tipo objeto Query que es una clase que hereda de ObjectType, la cual ofrece Graphene para definir un

tipo objeto GraphQL. Los campos que definen el tipo objeto Query son sus atributos de clase, y los métodos de clase corresponde a las funciones de resolución de cada uno de los campos. Cada función de resolución recibirá tres parámetros: una raíz o padre, información de la solicitud y argumentos del campo.

El parámetro root o raíz hace referencia al valor inicial que la función puede utilizar para resolver la información de su respectivo campo. Cuando GraphQL ejecuta las funciones de resolución para recoger toda la información que necesita lo hace por niveles, es decir, dependiendo de la profundidad del esquema y de los campos a los que se quiere llegar GraphQL los hace nivel por nivel en el árbol jerárquico. Por lo tanto, el parámetro root recibe el resultado de la llamada anterior, y es un valor nulo en caso de ser el primer nivel de ejecución.

El parámetro info en las funciones de resolución de Graphene, contiene el objeto AST de la consulta entrante y gracias a este parámetro los resolutores saben qué campos deben devolver (Burk, GraphQL Server Basics: Demystifying the info Argument in GraphQL Resolvers: Prisma, 2018).

Por último, se tiene el argumento que recibe el diccionario de parámetros que serán utilizados en la consulta (Burk, GraphQL Server Basics: Demystifying the info Argument in GraphQL Resolvers: Prisma, 2018).

Gráfica 28 ***Elementos de API GraphQL utilizando SDL.*** *(Graphene Python).*

```
type Query {
  hello(name: String = "stranger"): String
  goodbye: String
}
```

Fuente: datos del proyecto

Gráfica 29 ***Ejemplos de queries GraphQL con y sin parámetros****. (Graphene Python).*

```
# we can query for our field (with the default argument)
query_string = '{ hello }'
result = schema.execute(query_string)
print(result.data['hello'])
# "Hello stranger!"

# or passing the argument in the query
query_with_argument = '{ hello(name: "GraphQL") }'
result = schema.execute(query_with_argument)
print(result.data['hello'])
# "Hello GraphQL!"
```

Fuente: datos del proyecto

La gráfica 28 muestra los elementos de la API representados en el esquema GraphQL utilizando el SDL. En la gráfica 29 se tiene un ejemplo de la consulta realizada al servidor GraphQL mostrando dos casos posibles, con y sin envió de parámetros al campo hello.

Gráfica 30 ***Consulta GraphQL.*** *(Graphene Python).*

```
{
  hello(name: "friend")
}
```

Fuente: datos del proyecto

Gráfica 31 ***Respuesta GraphQL****. (Graphene Python).*

```
{
  "data": {
    "hello": "Hello friend!"
  }
}
```

Fuente: datos del proyecto

Las gráficas 30 y 31 se muestra el mismo ejemplo anterior de la consulta que recibe el servidor y su respuesta respectivamente. Esto elementos son una muestra de cómo se representa de manera general la especificación GraphQL en Python utilizando Graphene, lo cual para la implementación de la herramienta que se plantea desarrollar en este proyecto es una librería que puede agilizar el proceso de gestión de una API GraphQL aprovechando las características que ofrece Python para el desarrollo.

GraphQL y la metaprogramación en Python

La metaprogramación consiste en escribir programas que manipulan otros programas u otra concepción más habitual es que los metaprogramas tengan la capacidad de generar otros programas, como, por ejemplo: analizadores, interpretadores, compiladores etc (Sturtz).

(Hernandez, 2018) define la metaprogramación como “la habilidad de usar código para generar código”. Python admite una forma de metaprogramación para clases llamada metaclases (Sturtz). Normalmente en Python y en cualquier otro lenguaje de programación entendemos el concepto de clase como una plantilla que se utiliza para crear objetos, pero Python tienen una característica muy especial donde las clases también son consideradas como objetos (Cómo funcionan las metaclases en Python). En Python todo es considerado como un objeto, y como consecuencia una clase debe tener un tipo o metaclase de la cual es creada.

Como se puede observar en la gráfica 32, el tipo del objeto instanciado x es Foo, pero el tipo de la clase Foo, es type. Por lo tanto, type es la metaclase de la cual sus instancias son clases.

Gráfica 32 ***Ejemplo de clases y sus tipos.*** *(Sturtz).*

Python

```
>>> class Foo:
...     pass
...
>>> x = Foo()

>>> type(x)
<class '__main__.Foo'>

>>> type(Foo)
<class 'type'>
```

Fuente: datos del proyecto

En la gráfica 33 se observar la instancia de la clase myClass a partir de su metaclase type y su equivalente en la instancia de la clase de manera tradicional en Python. Como se puede observar el primer parámetro que recibe la metaclase type corresponde al nombre de la clase que se va a instanciar, segundo, recibe como parámetro una tupla donde se debe especificar las clases de las que hereda y por último se envía todos los atributos a manera de un diccionario, con atributos se hace referencia a atributos de clase y métodos.

Gráfica 33 ***Instancia de la metaclase type.*** *(Cómo funcionan las metaclases en Python).*

```
myClass = type('myClass',(),{'a':True})
```

```
class myClass(object):
    a = True
```

Fuente: datos del proyecto

La intención de incluir el concepto de la metaprogramación en el desarrollo del proyecto es tener la posibilidad de integrar nuevos servicios a la API GraphQL de manera dinámica, reduciendo al máximo la necesidad de digitar código y ofrecer una plataforma que permita la gestión de estos servicios. Entonces, aprovechando las características que ofrece Python, la implementación de una API GraphQL utilizando Graphene y la metaprogramación sería un proceso que se puede automatizar ya que todos los tipos que definirán el esquema están compuestos por clases que, como ya se mostró anteriormente son instancias de la metaclase type.

Django

Django es un framework de alto nivel basado en Python utilizado para el desarrollo de aplicaciones web, es gratuito y de código abierto. Fue desarrollado con el fin de facilitar y agilizar el desarrollo de sitios web, teniendo en cuenta aspectos de seguridad, escalabilidad, mantenibilidad y portabilidad con una comunidad muy amplia que contribuye a su desarrollo y mantenimiento (Django)

Es importante resaltar que Django como framework para el desarrollo de aplicaciones web respeta el patrón de diseño MVC (Model - View - Controller), con una pequeña diferencia en conceptos ya que en Django se utiliza el patrón MVT (Model - View -Template) donde el modelo hace referencia al manejo de datos, la vista corresponde a la lógica del negocio y la plantilla (template) es la parte que se encarga de la comunicación con el usuario final.

Existen numerosas librerías open source que permiten incorporar herramientas como REST o GraphQL en un proyecto Django de forma rápida; además, el propio framework ofrece una estructura que organiza y separa la lógica del proyecto, haciendo fácil la implementación de ciertos principios de arquitectura de software.

Metodología de desarrollo *Extreme Programming*

Es una metodología de desarrollo ágil especialmente útil cuando se abordan requisitos de software que cambian constantemente. Esta metodología recomienda el trabajo en pares (Laurence & Mancl, 2002), está centrada en el desarrollador y consiste en periodos de trabajo llamados iteraciones, en los cuales se busca resolver historias de usuario, cada iteración implica la implementación de un producto software funcional básico, estos se validan con usuarios dando pie a la retroalimentación, que luego es usada para las historias de usuario de la siguiente iteración; cada una de ellas requiere una documentación básica, pero no es el fuerte de esta metodología, pues su objetivo es el desarrollo constante. Se orientada por una serie de reglas:

- Comunicación: el trabajo en equipo es fundamental, compartir conocimientos, discutiendo la mejor forma de hacer determinada actividad y complementando las habilidades de los demás (Agile Alliance).

- Simpleza: se propone trabajar con el objetivo de obtener productos mínimos viables, manteniendo el diseño del sistema en las cosas absolutamente necesarias (Agile Alliance).

- Retroalimentación: es esencial revisar los resultados de cada una de las implementaciones logradas, de modo que se pueda identificar áreas de mejora y mejores prácticas para aplicar (Agile Alliance).

- Coraje: se requiere una discusión honesta y directa sobre las cosas que se están haciendo mal, de modo que se pueda corregirlas (Agile Alliance).

- Respeto: los miembros del equipo deben respetarse mutuamente para que las prácticas mencionadas previamente se ejecuten con éxito (Agile Alliance).

Marco contextual

Esta investigación se lleva a cabo en la Universidad de Nariño, una universidad pública del sur de Colombia, cuyo domicilio principal se encuentra ubicado en la ciudad de Pasto, pero con diferentes sedes en el departamento de Nariño. La Universidad de Nariño establece como su función misional el brindar servicios educativos para el desarrollo regional y nacional, a través de la producción de conocimientos científicos, tecnológicos, artísticos y humanísticos. En la actualidad cuenta con 11 Facultades, 50 programas académicos de pregrado, con 15 acreditados en alta calidad, 19 programas académicos de postgrado propios y 7 en Convenio. Según la Universidad, busca constantemente mejorar en los campos de la investigación, la docencia y proyección social, para responder adecuadamente a los retos que la modernidad le impone (Universidad de Nariño, s.f.).

Entre las dependencias académicas y administrativas adscritas a la Universidad de Nariño, existen algunas cuyo propósito es abordar este desafío desde el uso de herramientas tecnológicas. Por ejemplo, el Aula de Informática e Infraestructura

Tecnológica de la Universidad de Nariño, que depende de la Vicerrectoría Académica, funciona como el laboratorio donde se desarrollan las actividades académicas de la Universidad que tienen que ver con la informática, la ingeniería de sistemas y las telecomunicaciones, y cuenta con un equipo de trabajo cuya función es la de prestar una serie de servicios, en general asociados al uso de herramientas tecnológicas, en pro de colaborar con el desarrollo de las actividades de la Universidad (Universidad de Nariño, s.f.). Por su parte, la Oficina de Tecnologías de la Información y la Comunicación para la Educación de la Universidad de Nariño se enfoca en la formación docente entorno al uso de recursos educativos digitales, plataformas educativas y en general en el uso de las tecnologías de la información TIC para la investigación, interacción social y otros servicios (Universidad de Nariño, s.f.).

Entre estas dependencias se destaca el Centro de Informática de la Universidad, que se encarga de gestionar los requerimientos de información y automatización de los procesos de apoyo a funciones misionales de la Universidad. Hace parte del Proceso de Gestión de Información y Tecnología de la Universidad, y se encarga de administrar su sistema integrado de Información y Tecnología, fundamental para el desarrollo de los procesos y actividades en la Universidad, ofreciendo herramientas orientadas a satisfacer las necesidades de información de la institución y entidades externas (Universidad de Nariño, s.f.). Sus funciones incluyen:

- Análisis, diseño y desarrollo de software de apoyo para el cumplimiento de las funciones misionales de la Universidad.
- Soporte técnico a los aplicativos de software en producción.
- Asesoría y consultoría a la administración central en el ámbito tecnológico.
- Apropiación y despliegue de tecnología para el almacenamiento, procesamiento y transmisión de información.

Marco metodológico

Paradigma enfoque y tipo

Esta investigación sigue el paradigma cuantitativo, porque se enfoca en la recolección de datos medibles y cuantificables, que permitan mediante resultados estadísticos descriptivos, sacar conclusiones sobre qué tipo de servicios institucionales son los más usados por la comunidad universitaria de la Universidad de Nariño, y que nivel de interés hay en la implementación de estos en nuevas plataformas (Herrera, Pacheco, & Suazo).

Esta es una investigación aplicada tecnológica, pues pretende, a partir de conceptos, estudios e investigaciones previas, generar un producto software, que sea de provecho para la Universidad de Nariño, capaz de permitir la creación de servicios y mejorar la competitividad del sistema de información de la Universidad de Nariño y sus procesos (Investigación aplicada: qué es, características y ejemplos).

El tipo de investigación es exploratoria porque pretende proporcionar un acercamiento inicial a las necesidades de acceso a la información existentes en la comunidad universitaria de la Universidad de Nariño, además de explorar herramientas novedosas, con poca o inexistente literatura o estudios que indiquen su impacto, y usarla para aportar en el proceso de acceso a la información en la Universidad de Nariño. (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2010).

Población y muestra

Para el primer objetivo de caracterización de las necesidades de acceso a la información, la población corresponde la comunidad universitaria de la Universidad de Nariño, en específico a los estudiantes, los docentes y el personal administrativo.

Se toman los datos de la segunda mitad del año 2018, esto se debe a que la recolección de datos para la investigación se inicia en este periodo. Debido a que en el proceso de construcción de los instrumentos de recolección de información se evidencio que cada uno de los grupos de estudio tendrían intereses y necesidades particulares y diferentes entre ellos, se decidió aplicar 3 encuestas diferentes para cada uno. La población de estudiantes corresponde a los matriculados en el Semestre II de 2018 en la Universidad, y es de 13841 (Universidad de Nariño, 2018). La población de docentes corresponde a los vinculados a la Universidad en el año 2018, y es de 863 (Universidad de Nariño, 2018). La población del personal administrativo corresponde al vinculado a la Universidad en el Semestre II de 2018, y es de 521 (Universidad de Nariño, 2018).

Para esta primera etapa de recolección de información, se planteó un nivel de confianza del 95% y margen de error del 5% y se tomó una muestra no probabilística de los diferentes grupos poblacionales en estudio. Los instrumentos se aplicaron, por conveniencias logísticas, de forma virtual principalmente, haciendo uso del sistema de correos institucionales de la Universidad, a través del cual se envió las encuestas a los diferentes grupos poblacionales. Para el caso de los estudiantes, que era el grupo poblacional más grande, se requirió hacer una recolección extra de datos para cumplir con el margen de error deseado, para ello se hicieron visitas aleatorias a las aulas de informática del bloque tecnológico de la Universidad de Nariño, y se aplicaron las encuestas a los estudiantes presentes. Los resultados obtenidos fueron de 397 respuestas para la encuesta dirigida a estudiantes, 286 respuestas para la encuesta dirigida a docentes y 176 respuestas para la encuesta dirigida al personal administrativo.

El segundo objetivo involucro también una etapa de análisis documental entorno a las comparativas en términos de rendimiento entre las arquitecturas de API: GraphQL y REST. Para esto se hizo uso de fichas de análisis de contenido aplicadas a las investigaciones ‘GRAPHQL vs REST: una comparación desde la

perspectiva de eficiencia de desempeño' (Guillen-Drija, Quintero, & Kleiman, 2018), 'Experiencias migrando servicios web RESTful a GraphQL' (Vogel, Weber, & Zirpins, 2018) y 'Mejorando la interoperabilidad del ecosistema tecnológico basado en datos de la OEEU (Observatorio de Empleabilidad y Empleo Universitarios) con GraphQL' (Vazquez Ingelmo, Cruz Benito, & Garcia Peñalvo, 2017).

Por su parte, en el tercer objetivo, como parte del proceso de validación del aplicativo resultante de esta investigación, y teniendo en cuenta que este consiste en un administrador de información que facilite y ahorre trabajo en la implementación de aplicaciones backend, se aplicaron unas encuestas de caracterización y de nivel de conformidad e interés de los desarrolladores del centro de informática de la Universidad de Nariño entorno al aplicativo propuesto, esto debido a que serían ellos los principales beneficiarios al hacer uso de la propuesta, de ser aceptada. Dentro del proceso de recolección de datos se hizo una presentación de la aplicación, explicando su funcionalidad y posible utilidad para la Universidad. También se aplicaron unas encuestas, orientadas a la caracterización del nivel de conocimientos de los desarrolladores respecto a la temática de la investigación, y a la evaluación del nivel de aceptación de la propuesta planteada, y su nivel de utilidad, en opinión de los propios desarrolladores. Según datos oficiales de la Universidad, hay 31 profesionales asociados a labores de desarrollo de software en el centro de informática (Universidad de Nariño, s.f.). En la presentación del aplicativo, 27 de los desarrolladores estuvieron presentes y se obtuvieron 10 encuestas.

Proceso de Investigación

Tabla 1 ***Operacionalización del primer objetivo de la investigación****. Elaboración propia.*

Objetivo específico	Fuente	Técnica de recolección	Instrumento	Técnica de Procesamiento	Resultado
Caracterizar las necesidades de acceso a información académica e institucional de la comunidad universitaria de la Universidad de Nariño.	Comunidad universitaria de la Universidad de Nariño.	Encuesta	Escala de actitudes	Estadística descriptiva	Capitulo con la Caracterización de las necesidades de acceso a información académica e institucional de la comunidad universitaria de la Universidad de Nariño.

Tabla 2 ***Operacionalización del segundo objetivo de la investigación****. Elaboración propia.*

Objetivo específico	Fuente	Técnica de recolección	Instrumento	Técnica de Procesamiento	Resultado
Desarrollar un aplicativo que contribuya en el proceso de acceso a información académica e institucional de la comunidad universitaria de la Universidad de Nariño.	GraphQL vs REST	Investigación documental	Ficha de contenido	Análisis documental	Capitulo comparativo entre las herramientas GraphQL y REST, desde la perspectiva de cual es más apropiado para la construcción de un aplicativo que aporte en el proceso de acceso a la información en la Universidad de Nariño.

Tabla 3 ***Operacionalización del tercer objetivo de la investigación****. Elaboración propia.*

Objetivo específico	Fuente	Técnica de recolección	Instrumento	Técnica de Procesamiento	Resultado
Validar, en un entorno de pruebas, el aporte del aplicativo y los módulos implementados.	Desarrolladores del centro de informática de la Universidad de Nariño.	Encuesta	Escala de actitudes	Estadística descriptiva	Capitulo con la validación del nivel de utilidad del aplicativo presentado a los desarrolladores del centro de informática de la Universidad de Nariño.

Fuente: datos del proyecto

Tabla 4 ***Variables del primer objetivo de la investigación****. Elaboración propia.*

Nombre	Tipo	Descripción
Frecuencia de uso de servicios web institucionales	Independiente	Frecuencia de uso (escala Likert), por parte de estudiantes, docentes y personal administrativo, de ciertos servicios web ofrecidos en la página de la Universidad de Nariño.
Nivel de interés en la implementación de servicios web institucionales	Independiente	Nivel de interés (escala Likert), por parte de estudiantes, docentes y personal administrativo, en la implementación de ciertos servicios por parte de la Universidad de Nariño.

Fuente: datos del proyecto

Tabla 5 ***Variables del segundo objetivo de la investigación****. Elaboración propia.*

Nombre	Tipo	Descripción
Ventajas de uso: GraphQL vs REST	Independiente	Ventajas de uso existentes al comparar las arquitecturas API: REST y GraphQL.

Fuente: datos del proyecto

Tabla 6 ***Variables del tercer objetivo de la investigación****. Elaboración propia.*

Nombre	Tipo	Descripción

Nivel de conocimiento entorno a los conceptos API, REST y GraphQL	Independiente	Nivel de conocimiento (escala Likert), por parte de los desarrolladores del centro de informática de la Universidad de Nariño a cerca de los conceptos API, REST y GraphQL.
Frecuencia de uso de las herramientas REST y GraphQL	Independiente	Frecuencia de uso (escala Likert), por parte de los desarrolladores del centro de informática de la Universidad de Nariño a cerca de las arquitecturas API: REST y GraphQL.
Nivel de impacto del aplicativo en tiempo de desarrollo	Independiente	Nivel de impacto (escala Likert) que tendría el aplicativo en tiempo de desarrollo y facilidad de desarrollo, en opinión de los desarrolladores del centro de informática de la Universidad de Nariño.
Interés en la implementación del aplicativo	Independiente	Interés de los desarrolladores del centro de informática de la Universidad de Nariño en la utilización del aplicativo para el proceso de desarrollo.

Fuente: datos del proyecto

1. DESARROLLO DEL PROYECTO

1.1 CARACTERIZACIÓN DE NECESIDADES DE LA COMUNIDAD UNIVERSITARIA

1.1.1 Elaboración de los instrumentos de recolección de datos

La construcción de los instrumentos se hizo tomando como base la investigación “Arquitectura orientada a servicios para clientes móviles con Android” (Muñoz & Cabrera, 2017), realizada en la Universidad Mariana de la ciudad de Pasto, Colombia. En esta investigación se realiza una caracterización de las necesidades de información de la comunidad universitaria de la universidad en cuestión, y se enfoca particularmente en clientes móviles. El instrumento utilizado para la recolección de datos fue de encuestas dirigidas a su comunidad. Se decidió usar la investigación mencionada como referente debido a su similitud en el área de estudio.

Sin embargo, se encontró que las características y los servicios de la Universidad Mariana no son iguales a los de la Universidad de Nariño, por lo cual se hizo un replanteamiento de los instrumentos, teniendo en cuenta los servicios y particularidades de la Universidad de Nariño, se hizo validación con investigadores de la Universidad y se hicieron los ajustes correspondientes. Desde el proceso de construcción de los instrumentos se evidencio que cada uno de los grupos de estudio tendrían intereses y necesidades particulares y diferentes entre ellos, esta premisa se corroboró con el proceso de validación de los instrumentos y se evidencio su validez con los resultados de las encuestas aplicadas.

En consecuencia, el resultado del proceso fue de 3 encuestas dirigidas cada una a un sector particular de la comunidad universitaria de la Universidad de Nariño, siendo estos los estudiantes, los docentes y el personal administrativo, estas corresponden a los anexos 1, 2 y 3 respectivamente. Sin embargo, a pesar de que los instrumentos tienen particularidades que obligaron a separarlos, estos si comparten una estructura similar.

1.1.1.1 Estructura de los instrumentos de recolección de datos

La primera sección de las encuestas consistió en preguntas de caracterización de las personas encuestadas, con preguntas como edad, genero, áreas de vinculación, carrera, etc. En la siguiente sección se hicieron dos preguntas de selección múltiple usando la escala Likert con 5 niveles de respuesta incluyendo variable intermedia (Matas, 2018). Finalmente se definió una sección para una pregunta abierta, que permita complementar la información obtenida.

La primera pregunta de selección múltiple fue: '¿Con qué frecuencia hace uso de los siguientes servicios web institucionales?' Para la cual las opciones de respuesta fueron: Siempre, Casi siempre, Ocasionalmente, Casi nunca y Nunca. La intención de esta primera pregunta era la de verificar que tanto hace uso la comunidad de los servicios provistos por la Universidad.

La segunda pregunta de selección múltiple fue: '¿Está de acuerdo con la implementación de los siguientes servicios en una aplicación móvil?' Para la cual las opciones de respuesta fueron: Muy de acuerdo, De acuerdo, Indiferente, En desacuerdo y Muy en desacuerdo. La intención de esta pregunta era validar el interés de la comunidad en la implementación de determinados servicios en un dispositivo móvil, esto se debe a la hipótesis inicial de la investigación, que planteaba que el aporte al proceso de acceso a la información para la Universidad

podría ser la creación de una aplicación móvil que dé acceso a los servicios más usados por la comunidad universitaria.

En la sección de pregunta abierta, el enunciado fue: ‘A continuación, describa qué servicio(s) que no fue(ron) mencionado(s) anteriormente debería(n) ser implementado(s) en una aplicación móvil’. Nuevamente esto se debe a la hipótesis inicial de la investigación, que proponía la creación de una aplicación móvil, sin embargo, la utilidad de esta información no se limita a saber qué servicios se desea tener en una aplicación móvil, sino que servicios en general pueden ser de interés para la comunidad, y desean tener un acceso rápido a ellos.

1.1.1.2 Ítems evaluados en las preguntas de selección múltiple

Para la primera pregunta de selección múltiple, ‘¿Con qué frecuencia hace uso de los siguientes servicios web institucionales?’, los ítems evaluados fueron:

Para estudiantes:

- Consulta de notas.
- Consulta del plan de estudios del programa (pre requisitos, número de créditos).
- Consulta de horarios de clase.
- Consulta de horarios de atención de docentes.
- Consulta de noticias institucionales.
- Consulta de calendarios académicos.
- Consulta de datos de docentes (teléfono, correo).

Para docentes:

- Gestión de notas de estudiantes.
- Consulta de listados de estudiantes.
- Consulta de calendarios académicos.

- Consulta de horarios de clase.
- Consulta de desprendibles de pago.
- Consulta de material bibliográfico.

Para personal administrativo:

- Consulta de hojas de vida.
- Consulta de directorio telefónico y extensiones de las dependencias de la Universidad.
- Consulta de calendarios académicos.
- Consulta de oferta académica.
- Solicitud de certificado laboral.
- Consulta de noticias institucionales.
- Consulta de desprendibles de pago.

Para la segunda pregunta de selección múltiple, '¿Está de acuerdo con la implementación de los siguientes servicios en una aplicación móvil?', los ítems evaluados fueron:

Para estudiantes:

- Cálculo de notas.
- Consulta de notas.
- Consulta de horarios de clase.
- Consulta del plan de estudios del programa (prerrequisitos, número de créditos).
- Consulta de datos de docentes (teléfono, correo).
- Consulta de horario de atención de docentes.
- Recepción de notificaciones y comunicados de docentes.
- Consulta de calendarios académicos.
- Consulta de noticias institucionales.
- Ubicación de bloques y dependencias de la Universidad de Nariño.
- Consulta y publicación de objetos extraviados.

Para docentes:

- Consulta y gestión de notas de estudiantes.
- Consulta de listados de estudiantes.
- Consulta de noticias institucionales.
- Envío de notificaciones a estudiantes (cancelación de clases, cambio de aula).
- Consulta de calendarios académicos.
- Solicitud de certificados laborales.
- Ubicación de bloques y dependencias de la Universidad de Nariño.
- Consulta de horarios.
- Consulta de desprendibles de pago.
- Consulta de material bibliográfico.
- Consulta y publicación de objetos extraviados.

Para personal administrativo:

- Consulta de hojas de vida.
- Consulta de directorio telefónico y extensiones de las dependencias de la Universidad.
- Consulta de calendarios académicos.
- Consulta de oferta académica.
- Solicitud de certificado laboral.
- Consulta de noticias institucionales.
- Consulta de desprendibles de pago.
- Ubicación de bloques y dependencias de la Universidad de Nariño.
- Consulta y publicación de objetos extraviados.

1.1.1.3 Posibles mejoras a los instrumentos de recolección de datos

Una pregunta complementaria a la información obtenida podría ser: '¿Como calificaría el nivel de utilidad de los siguientes servicios web institucionales?' Con las opciones de respuesta: Muy importante, Importante, Relativamente importante, Poco importante, Nada importante. Esta pregunta permitiría la comparación entre la frecuencia de uso y el nivel de importancia, de modo que se pueda establecer causalidades, o establecer incongruencias que inviten a encontrar las razones por las cuales un servicio catalogado con alto nivel de importancia tiene baja frecuencia de uso.

Adicionalmente, la modificación obvia a las preguntas planteadas es la remoción del componente 'móvil' de ellas. Se debería centrar la investigación inicial en las necesidades de acceso a la información de forma general, y una vez cubierto este aspecto, ya se podría analizar el nivel de interés de una aplicación móvil y que servicios concretos pueden ser preferibles en este tipo de aplicación. Sin embargo, las preguntas preservan validez, a pesar de su especificidad entorno al componente móvil de la solución, pues aún recogen datos útiles sobre las necesidades y preferencias que la comunidad universitaria tiene entorno al acceso a la información de la Universidad.

1.1.2 Recolección de datos

Por conveniencia de los investigadores, se decidió inicialmente hacer uso del sistema de correos institucionales de la Universidad de Nariño para aplicar las encuestas. Este sistema cuenta con una base de datos de los estudiantes, docentes y personal administrativo de la universidad, por lo cual resultaba muy conveniente. Se hizo solicitud al centro de informática y se recibió autorización para enviar la encuesta a los diferentes grupos poblacionales en estudio. Para hacerlo se hizo uso también de la herramienta Google Forms, subiendo los instrumentos a esta plataforma para enviar el link resultante a través de mensajes de correo electrónico.

Para el caso de los estudiantes, que era el grupo poblacional más grande, la cantidad de respuestas obtenidas no fue satisfactoria, y se vio necesario hacer una recolección extra de datos para cumplir con el margen de error deseado. Por ello se tomó la decisión de hacer visitas aleatorias a las aulas de informática del bloque tecnológico de la Universidad de Nariño, y se aplicaron las encuestas a los estudiantes presentes. Se hizo de esta forma porque estas aulas cuentan con computadores y acceso a internet, lo cual facilita enormemente la aplicación de los instrumentos que, como se mencionó anteriormente, estaban subidos en una plataforma en línea. Una vez se obtuvo el número de respuestas suficiente para cumplir con el margen de error para la muestra de estudiantes, se dio por terminado el proceso de recolección de datos.

Finalmente, los resultados obtenidos fueron de 397 respuestas para la encuesta dirigida a estudiantes, 286 respuestas para la encuesta dirigida a docentes y 176 respuestas para la encuesta dirigida al personal administrativo. Los datos planos de las encuestas aplicadas a estudiantes, docentes y personal administrativo corresponden a los anexos 6, 7 y 8 respectivamente.

1.1.3 Preparación de datos para el análisis

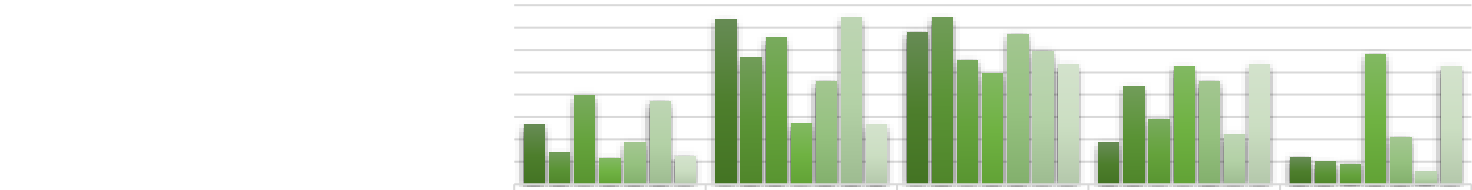
En el caso de las preguntas de selección múltiple, los datos se tabularon cuantificando el número de apariciones de cada opción de la escala Likert planteada en las preguntas. Para mostrar los datos se hizo uso de gráficos de barras para comparar niveles de aparición de cada opción, de modo que se pueda observar fácilmente en que espectro de la escala esta la mayoría de respuestas obtenidas.

Para el caso de las preguntas abiertas, estas fueron más difíciles de preparar, puesto que al ser en su mayoría textos largos, y al haber gran cantidad de ellos, el analizarlos uno por uno resultó muy extenso, por esta razón se decidió separar estas respuestas en archivos de texto planos y graficar frecuencia de aparición de

palabras, de modo que se pueda observar que es lo que los encuestados mencionan más en sus respuestas.

1.1.3.1 Datos obtenidos de estudiantes

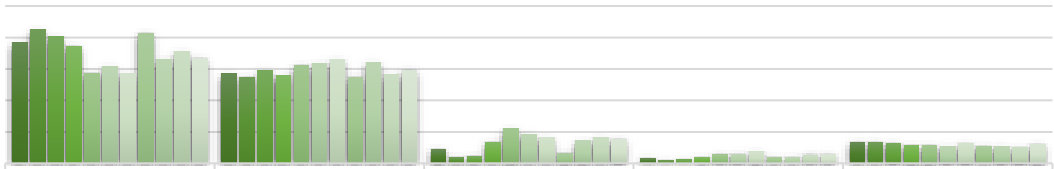
Gráfica 34 ***Frecuencia de uso de servicios por parte de estudiantes.*** *Elaboración propia.*



	Siempre	Casi siempre	Ocasionalmente	Casi nunca	Nunca
Consulta de notas	53	147	136	37	24
Consulta del plan de estudios del programa (pre requisitos, número de créditos)	28	113	149	87	20
Consulta de horarios de clase	79	131	111	58	18
Consulta de horarios de atención de docentes	23	54	99	105	116
Consulta de noticias institucionales	37	92	134	92	42
Consulta de calendarios académicos	74	149	119	44	11
Consulta de datos de docentes (teléfono, correo)	25	53	107	107	105

Fuente: datos del proyecto

Gráfica 35 ***Interés de implementación de servicios por parte de estudiantes.*** *Elaboración propia.*



	Muy de acuerdo	De acuerdo	Indiferente	En desacuerdo	Muy en desacuerdo
Cálculo de notas	192	142	22	8	33
Consulta de notas	212	137	10	5	33
Consulta de horarios de clase	201	147	11	6	32
Consulta del plan de estudios del programa (pre requisitos, numero de créditos)	186	139	33	10	29
Consulta de datos de docentes (teléfono, correo)	143	156	55	14	29
Consulta de horario de atención de docentes	154	158	45	14	26
Recepción de notificaciones y comunicados de docentes	142	165	41	18	31
Consulta de calendarios académicos	207	137	16	10	27
Consulta de noticias institucionales	165	160	36	10	26
Ubicación de bloques y dependencias de la Universidad de Nariño	177	141	41	13	25
Consulta y publicación de objetos extraviados	167	148	38	14	30

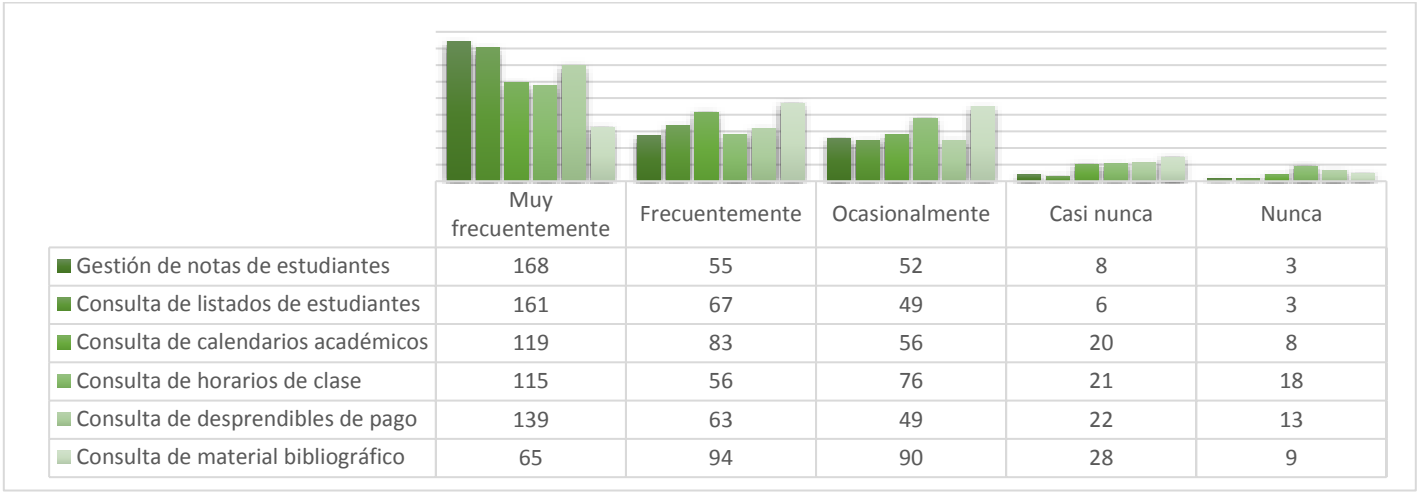
Fuente: datos del proyecto

Gráfica 36 ***Nube de palabras de servicios deseados por estudiantes.*** *Elaboración propia*

Fuente: datos del proyecto

1.1.3.2 Datos obtenidos de docentes

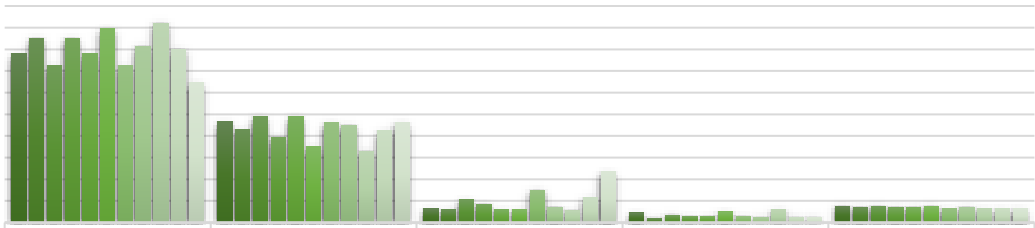
Gráfica 37 ***Frecuencia de uso de servicios por parte de docentes.*** *Elaboración propia.*



	Muy frecuentemente	Frecuentemente	Ocasionalmente	Casi nunca	Nunca
Gestión de notas de estudiantes	168	55	52	8	3
Consulta de listados de estudiantes	161	67	49	6	3
Consulta de calendarios académicos	119	83	56	20	8
Consulta de horarios de clase	115	56	76	21	18
Consulta de desprendibles de pago	139	63	49	22	13
Consulta de material bibliográfico	65	94	90	28	9

Fuente: datos del proyecto

Gráfica 38 ***Interés de implementación de servicios por parte de docentes.*** *Elaboración propia.*



	Muy de acuerdo	De acuerdo	Indiferente	En desacuerdo	Muy en desacuerdo
Consulta y gestión de notas de estudiantes	156	93	13	9	15
Consulta de listados de estudiantes	170	86	12	4	14
Consulta de noticias institucionales	145	98	21	7	15
Envío de notificaciones a estudiantes (cancelación de clases, cambio de aula)	170	79	17	6	14
Consulta de calendarios académicos	156	98	12	6	14
Solicitud de certificados laborales	179	70	12	10	15
Ubicación de bloques y dependencias de la Universidad de Nariño	145	92	30	6	13
Consulta de horarios	163	90	14	5	14
Consulta de desprendibles de pago	184	66	11	12	13
Consulta de material bibliográfico	160	85	23	5	13
Consulta y publicación de objetos extraviados	129	92	47	5	13

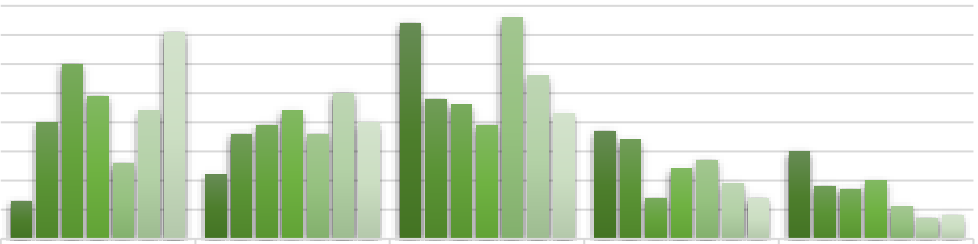
Fuente: datos del proyecto

Gráfica 39 ***Nube de palabras de servicios deseados por docentes.*** *Elaboración propia.*

Fuente: datos del proyecto

1.1.3.3 Datos obtenidos de personal administrativo

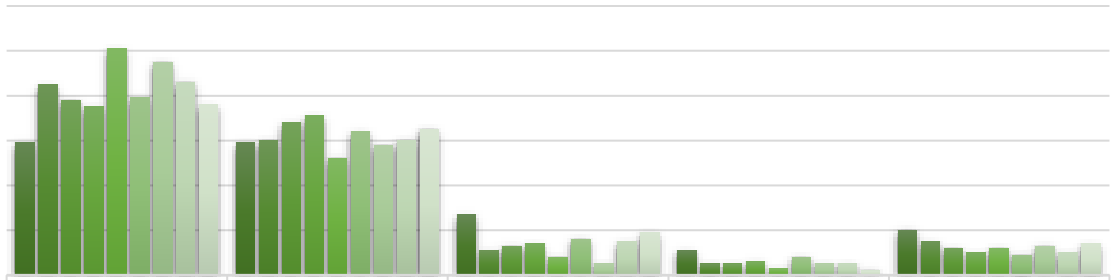
Gráfica 40 ***Frecuencia de uso de servicios por parte de personal administrativo.*** *Elaboración propia.*



	Siempre	Casi siempre	Ocasionalmente	Casi nunca	Nunca
Consulta de hojas de vida	13	22	74	37	30
Consulta de directorio telefónico y extensiones de las dependencias de la Universidad	40	36	48	34	18
Consulta de calendarios académicos	60	39	46	14	17
Consulta de oferta académica	49	44	39	24	20
Solicitud de certificado laboral	26	36	76	27	11
Consulta de noticias institucionales	44	50	56	19	7
Consulta de desprendibles de pago	71	40	43	14	8

Fuente: datos del proyecto

Gráfica 41 ***Interés de implementación de servicios por parte de personal administrativo.*** *Elaboración propia.*



	Muy de acuerdo	De acuerdo	Indiferente	En desacuerdo	Muy en desacuerdo
Consulta de hojas de vida	59	59	27	11	20
Consulta de directorio telefónico y extensiones de las dependencias de la Universidad	85	60	11	5	15
Consulta de calendarios académicos	78	68	13	5	12
Consulta de oferta académica	75	71	14	6	10
Solicitud de certificado laboral	101	52	8	3	12
Consulta de noticias institucionales	79	64	16	8	9
Consulta de desprendibles de pago	95	58	5	5	13
Ubicación de bloques y dependencias de la Universidad de Nariño	86	60	15	5	10
Consulta y publicación de objetos extraviados	76	65	19	2	14

Fuente: datos del proyecto

Gráfica 42 ***Nube de palabras de servicios deseados por personal administrativo.*** *Elaboración propia.*

Fuente: datos del proyecto

1.1.4 Análisis de los datos

1.1.4.1 Análisis de los datos de estudiantes

Para la mayoría de servicios evaluados, hay una frecuencia considerable de uso, lo que indica que hay ciertos servicios que son relevantes para las actividades cotidianas de los estudiantes. Por otra parte, si analizamos el nivel de interés por la implementación de estos servicios en una aplicación móvil, este es bastante alto, incluso para servicios marcados con una frecuencia de uso baja. Esto podría significar que hay dificultades en la accesibilidad de estos servicios, y a pesar de que son importantes o útiles para los estudiantes, lo cual se demuestra en su interés por ellos, estos no están disponibles para ellos, o al menos no de un modo rápido o conocido.

1.1.4.2 Análisis de los datos de docentes

En este caso hay incluso la frecuencia de uso de los servicios evaluados es aún más alta, esto es un indicio de lo importante que es para los docentes el tener herramientas que les permitan cumplir con sus funciones, por ejemplo, gestionar sus actividades evaluativas, contactar a sus estudiantes, etc. En este sentido, no es de extrañar que el nivel de interés por la implementación de estos servicios en una aplicación móvil, sea igual de alto, pues poder acceder a estas herramientas que son tan importantes para sus labores cotidianas, desde su dispositivo móvil, es algo muy conveniente.

1.1.4.3 Análisis de los datos de personal administrativo

La mayoría de servicios evaluados para el personal administrativo, muestran una frecuencia considerable de uso, pues la mayoría de datos están en la parte positiva de la escala usada. Esto indica que, en términos generales, estos servicios son relevantes y útiles para las actividades del personal, aunque algunos de ellos quizá

de un modo más esporádico. Por otra parte, si analizamos el nivel de interés por la implementación de estos servicios en una aplicación móvil, al igual que en los casos anteriores, este es bastante alto, incluso para servicios marcados con una frecuencia de uso baja. Esto refuerza la idea de que, para las personas, resulta muy conveniente tener acceso a los servicios y la información que necesitan de la forma más rápida y cómoda posible, y el hacerlo a través del dispositivo móvil ofrece estas características.

1.1.4.4 Análisis de datos general

En general, los resultados demuestran que si existe un interés por parte de la comunidad universitaria de la Universidad de Nariño por el uso de sistemas de información que les provean con los servicios necesarios para el cumplimiento de sus actividades. Una muestra clara de ello es que, en todas las gráficas de nubes de palabras, información, consulta y servicios tienen un nivel de relevancia bastante alto, es decir que están en la mente y los deseos de la mayoría. Además, hay interés en que se desarrolle más servicios, algunos de los cuales también se evidencian en las palabras de alta frecuencia de estas gráficas al igual que en las respuestas abiertas, un ejemplo es el envió de notificaciones.

Al mismo tiempo, los resultados reflejan que se hizo un análisis relativamente certero de los servicios que podrían ser de interés para cada grupo poblacional estudiado, pues la frecuencia de uso de estos ítems, en todos los casos se encuentra en su mayoría en la parte positiva de la escala usada. En este sentido, también se confirma que, aunque hay intereses comunes entorno a información y ciertos servicios, también se evidencia que cada grupo poblacional tendrá necesidades diferentes, y esto es de esperarse puesto que cada uno tiene funciones diferentes dentro de la universidad, y requieren de herramientas diferentes para su cumplimiento.

En consecuencia, estos resultados también sugieren que, si se desea ayudar a la universidad con el proceso de conectar a su comunidad con información y servicios a través de los medios tecnológicos, será indispensable el adaptarse a requisitos diversos, y estar preparado a responder a nuevas necesidades y problemáticas específicas de cada sector.

1.1.5 Síntesis

En este capítulo se describe el proceso de caracterización de las necesidades de acceso a la información de la comunidad universitaria de la Universidad de Nariño. Se ejecuta mediante la aplicación de encuestas a los tres grupos poblacionales escogidos, siendo estos los estudiantes, docentes y el personal administrativo de la Universidad. Los resultados arrojan que la comunidad universitaria de la Universidad de Nariño hace un uso extensivo de los servicios ofrecidos de forma virtual por parte de la Universidad, y que hay interés en que se desarrolle más servicios, y en la expansión de estos a plataformas como la de los dispositivos móviles.

1.2 DESARROLLO DEL APLICATIVO

La aplicación se desarrolló mediante la metodología *Extreme Programming,* pues sus características, revisadas anteriormente, se ajustan muy bien a las necesidades del presente proyecto.

1.2.1 Definición de roles

Tabla 7 ***Roles asignados para el cumplimiento del objetivo.*** *Elaboración propia*

Rol	Integrante
Cliente	Franklin Jimenez, John Getial, David Ceron
Desarrolladores	John Getial, David Ceron
Gestor (Big boss)	Franklin Jiménez
Entrenador	Franklin Jiménez
Tester	John Getial, David Ceron
Rastreador (Tracker)	John Getial
Consultor	Giovanni

Fuente: datos del proyecto

1.2.2 Recolección de requisitos

El análisis de los resultados de la caracterización de las necesidades de acceso a la información de la comunidad universitaria de la Universidad de Nariño, además de la revisión teórica, y de los antecedentes, hace evidente que los requisitos y necesidades potenciales a ser resueltos son muy diversos y podrían evolucionar en el futuro, por lo cual es imperativo ofrecer la posibilidad de integrar nuevas aplicaciones y servicios que puedan dar respuesta a cualquier necesidad que se presente dentro de la comunidad universitaria y externos. Para lograrlo, se propone la creación de una API, que ponga a disposición de la Universidad los servicios

necesarios para el desarrollo y cumplimiento de los diferentes requisitos que puedan surgir de parte de la comunidad.

Esta API permitiría la comunicación entre las diferentes entidades dentro de la universidad, utilizando una arquitectura que facilite compartir información y servicios desde la Universidad hacia su comunidad, incluyendo estudiantes, profesores, administrativos y público en general. Sin embargo, debe cumplir con ciertos requisitos para que sea de la mayor utilidad posible en el proceso de compartir información y consumir estos recursos por parte la población destino, por ende, se debe evaluar cuál de las tecnologías de implementación de API disponibles es la más apropiada. Para ello se acude a revisión documental de investigaciones comparativas de estas herramientas.

1.2.2.1 Análisis documental

Debido a que GraphQL surge como una alternativa principalmente a la arquitectura REST, suele ser natural la comparativa entre las dos, sin embargo, dado que GraphQL es, en términos relativos, una arquitectura nueva, la literatura académica relacionada con sus características y rendimiento aún es muy escasa (Vogel, Weber, & Zirpins, 2018). Para este apartado se han escogido tres investigaciones que ofrecen una comparativa directa entre las arquitecturas en cuestión: 'GRAPHQL vs REST: una comparación desde la perspectiva de eficiencia de desempeño' (Guillen-Drija, Quintero, & Kleiman, 2018), 'Experiencias migrando servicios web RESTful a GraphQL' (Vogel, Weber, & Zirpins, 2018) y 'Mejorando la interoperabilidad del ecosistema tecnológico basado en datos de la OEEU (Observatorio de Empleabilidad y Empleo Universitarios) con GraphQL' (Vazquez Ingelmo, Cruz Benito, & Garcia Peñalvo, 2017).

Tabla 8 ***Ficha de contenido 1****. Elaboración propia.*

"Experiencias migrando servicios web RESTful a GraphQL"

Las diferencias en la arquitectura para una API REST y GraphQL hacen difícil la comparación de rendimiento en base a migración de una misma API, por lo que esta debe hacerse con cuidado, para preservar la validez de los datos obtenidos. **(Vogel, Weber, & Zirpins, 2018)** Ficha número 1

Fuente: datos del proyecto

Tabla 9 ***Ficha de contenido 2****. Elaboración propia.*

"GraphQL vs REST: una comparación desde la perspectiva de eficiencia de desempeño"
El estilo REST impone que debe utilizarse el caché para así dar menor uso al lado del servidor y así aumentar la velocidad de las interacciones entre el cliente y el servidor, mientras que GraphQL como tecnología no ofrece de forma natural este middleware. **(Guillen-Drija, Quintero, & Kleiman, 2018)** Ficha número 2

Fuente: datos del proyecto

Tabla 10 ***Ficha de contenido 3****. Elaboración propia.*

“GraphQL vs REST: una comparación desde la perspectiva de eficiencia de desempeño”
El comportamiento de ambas API bajo estrés y en llamadas únicas, arroja que la API REST posee mayor velocidad de respuesta que la API GraphQL. En este caso la API REST tiene cache y la API GraphQL no. **(Guillen-Drija, Quintero, & Kleiman, 2018)** Ficha número 3

Fuente: datos del proyecto

Tabla 11 ***Ficha de contenido 4****. Elaboración propia.*

“Experiencias migrando servicios web RESTful a GraphQL”
En las pruebas de rendimiento, para el caso de consulta a un solo modelo de datos, a través de un solo endpoint REST y una sola query Graphql, obteniendo los mismos datos, se concluye que GraphQL y REST tienen un rendimiento equivalente, sin ventajas para ninguno en la velocidad de respuesta. **(Vogel, Weber, & Zirpins, 2018)** Ficha número 4

Fuente: datos del proyecto

Tabla 12 ***Ficha de contenido 5****. Elaboración propia.*

“Experiencias migrando servicios web RESTful a GraphQL” Para consulta de datos conectados a varios modelos, a través de varios endpoints REST y una sola query Graphql, obteniendo los mismos datos, se concluye que GraphQL tiene ventaja en la velocidad de respuesta, debido a que las consultas consecutivas requeridas para obtener la misma información mediante REST, aumentan significativamente el tiempo que le toma para estar disponible en el cliente. **(Vogel, Weber, & Zirpins, 2018)** Ficha número 5

Fuente: datos del proyecto

Tabla 13 ***Ficha de contenido 6****. Elaboración propia.*

“GraphQL vs REST: una comparación desde la perspectiva de eficiencia de desempeño”
Las peticiones dinámicas de GraphQL, que permiten especificar exactamente los campos que se requieren, además de la posibilidad de obtener campos de diferentes modelos que estén relacionados, en una sola consulta, hace que las respuestas de una API GraphQL sean considerablemente más reducidas que las que genera una API REST. En consecuencia, GraphQL tiene un impacto positivo en el rendimiento de los clientes, pues reciben los datos exactos que necesita, disminuyendo la carga y mejorando rendimiento. **(Guillen-Drija, Quintero, & Kleiman, 2018)** Ficha número 6

Fuente: datos del proyecto

Tabla 14 ***Ficha de contenido 7****. Elaboración propia.*

"Experiencias migrando servicios web RESTful a GraphQL" Un servidor GraphQL proporciona una estructura que es muy útil en la implementación de microservicios, al permitir encapsularlos y entregarlos en una API web unificada, de este modo se puede evitar efectos negativos en rendimiento debidos a llamados múltiples a diferentes endpoints. **(Vogel, Weber, & Zirpins, 2018)** Ficha número 7

Fuente: datos del proyecto

Tabla 15 ***Ficha de contenido 8****. Elaboración propia.*

"Mejorando la interoperabilidad del ecosistema tecnológico basado en datos de la OEEU con GraphQL" Gracias a la capacidad de GraphQL de devolver los datos dependiendo de la query, y no del endpoint como en REST, existe mayor libertad para la adición de nuevos componentes sin necesitar cambios en la API, puesto que solo se requiere una nueva query en el cliente. Además, la adición de campos y nuevos datos en la API GraphQL tampoco afecta los ya existentes, de modo que esta flexibilidad aplica para frontend y backend. **(Vazquez Ingelmo, Cruz Benito, & Garcia Peñalvo, 2017)** Ficha número 8

Fuente: datos del proyecto

Tabla 16 ***Ficha de contenido 9****. Elaboración propia.*

“Mejorando la interoperabilidad del ecosistema tecnológico basado en datos de la OEEU con GraphQL” Dado que los datos llegan de forma unificada mediante una query GraphQL, ya no es necesario gastar tiempo y código en implementar funciones que transformen los datos devueltos por múltiples endpoints REST en un solo objeto que se ajuste a las necesidades del cliente. Esto mejora la experiencia del desarrollador. **(Vazquez Ingelmo, Cruz Benito, & Garcia Peñalvo, 2017)** Ficha número 9

Fuente: datos del proyecto

Tabla 17 ***Ficha de contenido 10****. Elaboración propia.*

“Mejorando la interoperabilidad del ecosistema tecnológico basado en datos de la OEEU con GraphQL” En entornos con constantes cambios de requerimientos, con componentes que deban ajustarse y evolucionar respondiendo a estos cambios, GraphQL facilita enormemente el trabajo del desarrollador, puesto que, si los datos requeridos por el componente cambian, basta con modificar la query, sin necesidad de ajustar la API. Esto no sería posible con REST. **(Vazquez Ingelmo, Cruz Benito, & Garcia Peñalvo, 2017)** Ficha número 10

Fuente: datos del proyecto

Tabla 18 ***Ficha de contenido 11****. Elaboración propia.*

“Mejorando la interoperabilidad del ecosistema tecnológico basado en datos de la OEEU con GraphQL” La independencia, escalabilidad, flexibilidad y mantenibilidad ofrecidas por una API GraphQL permite que organizaciones diferentes trabajen en base a una API de la que no tienen control directo, sin tener que preocuparse constantemente por ajustar sus componentes a los cambios hechos en ella. Esto también aplica para diferentes dependencias/departamentos de una misma organización. **(Vazquez Ingelmo, Cruz Benito, & Garcia Peñalvo, 2017)** Ficha número 11

Basados en los resultados de la revisión documental, se opta por GraphQL como la tecnología para la API debido a la flexibilidad y escalabilidad que ofrece, y sus ventajas ya discutidas en apartados anteriores. Se decide hacer uso de criterios de arquitectura de software y se adiciona como parte del aporte de la investigación la sugerencia para adoptar estos criterios en futuros desarrollos para la Universidad.

1.2.2.2 Definición de requisitos

Se planifica el desarrollo de una aplicación web que permita:

- Establecer conexiones con diferentes bases de datos para obtener y organizar información.
- Entregar la información en forma de servicios a partir de una API GraphQL.
- La API GraphQL debe poder consumirse por diferentes clientes.
- Se debe poder controlar el acceso a los servicios de la API con credenciales de usuario.

Se elije Django como el framework de desarrollo de la aplicación dado que trabaja con una estructura que ya cumple con los criterios de arquitectura mencionados,

proponiendo limites arquitectónicos útiles, además de variedad de librerías y documentación para el trabajo con diferentes bases de datos y GraphQL.

1.2.3 Historias de usuario

Tabla 19 ***Historia de usuario gestión de conexiones a bases de datos.*** *Elaboración propia.*

Historia de usuario	
Número: 01	**Usuario:** Administrador del sistema
Nombre Historia: Gestión de conexiones a bases de datos	
Prioridad en Negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos Estimados: 1	**Iteración Asignada:** 1
Programador Responsable:	
Descripción: El punto de partida para compartir información es una consulta SQL, por lo cual se requiere poder contar con los datos de una conexión a una base de datos para verificar su estructura y obtener los datos de su ejecución.	
Observaciones:	

Fuente: datos del proyecto

Tabla 20 ***Historia de usuario administración de servicios SQL.*** *Elaboración propia.*

Historia de usuario	
Número: 02	**Usuario:** Administrador del sistema
Nombre Historia: Administración de servicios SQL.	
Prioridad en Negocio: Alta	**Riesgo en desarrollo:** Alto

Puntos Estimados: 1	**Iteración Asignada:** 1
Programador Responsable:	
Descripción: Permite la gestión de las consultas SQL que serán representadas posteriormente como servicios, además se debe permitir validar y ejecutar la consulta a través de la conexión que se especifique.	
Observaciones: Cada tipo que será expuesto a través de la API GraphQL para su consulta se lo denominará como servicio.	

Fuente: datos del proyecto

Tabla 21 ***Historia de usuario agrupación de servicios.*** *Elaboración propia.*

Historia de usuario	
Número: 03	**Usuario:** Administrador del sistema
Nombre Historia: Agrupación de servicios.	
Prioridad en Negocio: Media	**Riesgo en desarrollo:** Alto
Puntos Estimados: 1	**Iteración Asignada:** 1
Programador Responsable:	
Descripción: Se necesita contar con una estructura que permita crear grupos de servicios teniendo la posibilidad de anidar sin importar el nivel de profundidad de la agrupación. Cada grupo de servicios estará compuesto por servicios de tipo SQL o servicios que forman otros grupos de servicios.	
Observaciones: Un grupo de servicios debe definirse como un tipo GraphQL para que pueda ser representado en el esquema, cuyos campos van a corresponder a otros tipos GraphQL ya sea que representen un servicio de una consulta SQL o que se creen de manera manual como grupo.	

Fuente: datos del proyecto

Tabla 22 ***Historia de usuario modelado de servicios.*** *Elaboración propia.*

Número: 04	**Usuario:** Administrador del sistema
Nombre Historia: Modelado de servicios.	
Prioridad en Negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos Estimados: 3	**Iteración Asignada:** 2
Programador Responsable:	
Descripción: Cada servicio gestionado debe poder construirse de manera automática para que pueda ser accedido a través de la API. Los datos de cada servicio se deben modelar para representar sus respectivos tipos en un esquema GraphQL.	
Observaciones: Se debe evitar al máximo la necesidad de escribir código para poder crear, modificar o eliminar un servicio de la API.	

Fuente: datos del proyecto

Tabla 23 ***Historia de usuario resolución de servicios.*** *Elaboración propia.*

Historia de usuario	
Número: 05	**Usuario:** Administrador del sistema
Nombre Historia: Resolución de servicios.	
Prioridad en Negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos Estimados: 2	**Iteración Asignada:** 2
Programador Responsable:	
Descripción: Las funciones de resolución que se encargan de devolver la información de un servicio, al igual que el modelado de los tipos se deben generar de manera automática tomando como fuente la información de la consulta SQL de su respectivo servicio.	

Observaciones:

Fuente: datos del proyecto

Tabla 24 ***Historia de usuario autenticación de usuarios.*** *Elaboración propia.*

Número: 06	**Usuario:** Administrador del sistema
Nombre Historia: Autenticación de clientes GraphQL.	
Prioridad en Negocio: Media	**Riesgo en desarrollo:** Alto
Puntos Estimados: 1	**Iteración Asignada:** 3
Programador Responsable:	
Descripción: Se necesita exponer un servicio a través de la API que permita la autenticación de usuarios. Esta autenticación debe soportar dos tipos de recursos diferentes, ya sea autenticando usuarios con información de una base de datos o utilizando un directorio activo (LDAP).	
Observaciones:	

Fuente: datos del proyecto

Tabla 25 ***Historia de usuario gestión de roles y permisos.*** *Elaboración propia.*

Historia de usuario	
Número: 07	**Usuario:** Administrador del sistema
Nombre Historia: Gestión de roles y permisos.	
Prioridad en Negocio: Media	**Riesgo en desarrollo:** Alto
Puntos Estimados: 1	**Iteración Asignada:** 3
Programador Responsable:	

Descripción: Se debe permitir al usuario administrador poder crear roles de usuario para limitar el acceso a los servicios. Estos roles deben poderse validar dependiendo del tipo de autenticación que se maneje.
Observaciones:

Fuente: datos del proyecto

Tabla 26 ***Historia de usuario autorización a servicios****. Elaboración propia.*

Historia de usuario	
Número: 08	**Usuario:** Administrador del sistema
Nombre Historia: Autorización a servicios.	
Prioridad en Negocio: Media	**Riesgo en desarrollo:** Alto
Puntos Estimados: 2	**Iteración Asignada:** 3
Programador Responsable:	
Descripción: Por seguridad se requiere que cada servicio pueda restringirse solo para un grupo de usuarios a través del rol que se especifique para dicho servicio.	
Observaciones:	

Fuente: datos del proyecto

Tabla 27 ***Historia de usuario interfaz para conexiones****. Elaboración propia.*

Historia de usuario	
Número: 09	**Usuario:** Administrador del sistema
Nombre Historia: Interfaz de usuario para conexiones.	
Prioridad en Negocio: Media	**Riesgo en desarrollo:** Medio

Puntos Estimados: 1	**Iteración Asignada:** 4
Programador Responsable:	
Descripción: Se debe brindar una interfaz de usuario para gestionar conexiones.	
Observaciones:	

Fuente: datos del proyecto

Tabla 28 ***Historia de usuario interfaz de seguridad****. Elaboración propia.*

Historia de usuario	
Número: 10	**Usuario:** Administrador del sistema
Nombre Historia: Interfaz de seguridad.	
Prioridad en Negocio: Media	**Riesgo en desarrollo:** Medio
Puntos Estimados: 1	**Iteración Asignada:** 4
Programador Responsable:	
Descripción: Se debe brindar una interfaz de usuario que permita configurar los parámetros de autenticación y gestión de roles para controlar el acceso a la información de los servicios.	
Observaciones:	

Fuente: datos del proyecto

Tabla 29 ***Historia de usuario interfaz de servicios****. Elaboración propia.*

Historia de usuario	
Número: 11	**Usuario:** Administrador del sistema
Nombre Historia: Interfaz de servicios.	
Prioridad en Negocio: Media	**Riesgo en desarrollo:** Medio
Puntos Estimados: 1	**Iteración Asignada:** 4
Programador Responsable:	
Descripción: Se debe brindar una interfaz usuario que permita gestionar servicios y configurar parámetros de seguridad.	
Observaciones: Cada servicio puede estar asociado a uno o varios roles para la autorización de los datos de su consulta.	

1.2.4 Plan de iteraciones

Tabla 30 ***Plan de entrega de desarrollo****. Elaboración propia.*

No. de Historia	Iteración	Prioridad	Puntos estimados
01	1	Alta	1
02	1	Alta	1
03	1	Alta	1
04	2	Alta	3
05	2	Alta	2
06	3	Media	1
07	3	Media	1

08	3	Media	2
09	4	Media	1
10	4	Media	1
11	4	Media	1

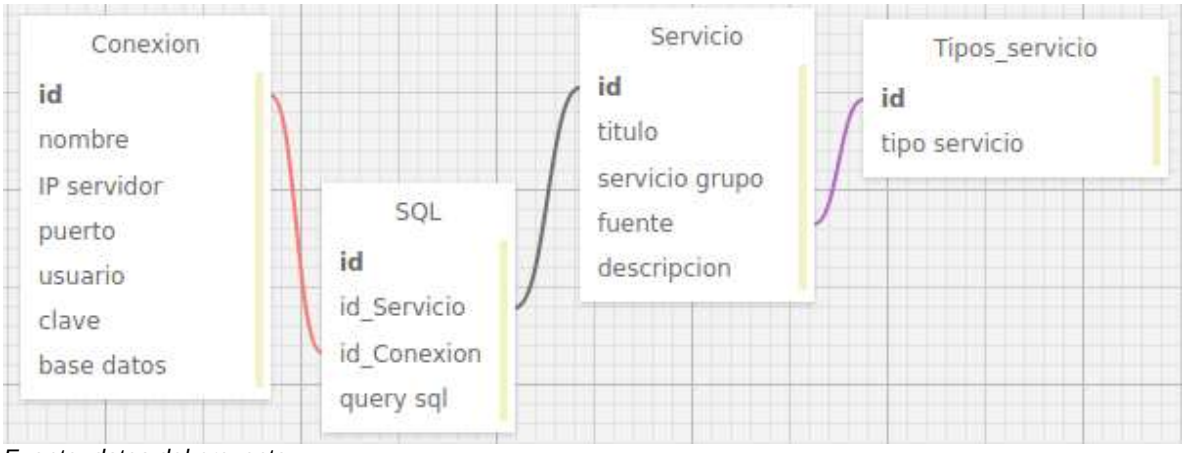
Fuente: datos del proyecto

1.2.5 Desarrollo de iteraciones

1.2.5.1 Primera iteración.

El punto de partida para esta iteración es la creación de un nuevo proyecto de Django que es una aplicación base que ofrece una parametrización, configuración y funcionalidades iniciales sobre la cual se continuará el desarrollo. Para esta primera fase se obtendrá la estructura base sobre la cual se apoyará el desarrollo de las demás iteraciones, definiendo principalmente el modelo de la base de datos que dará solución a la captura de los recursos o información básica para la construcción de servicios.

Gráfica 43 ***Estructura de base de datos: Primera Iteración.*** *Elaboración propia*



Fuente: datos del proyecto

Gráfica 44 ***Modelo conexión.*** *Elaboración propia.*

```
class Connection(models.Model):
    # Atributos del modelo conexión
    connection_name = models.CharField(max_length=50, unique=True)
    host = models.CharField(max_length=100)
    port = models.IntegerField()
    user = models.CharField(max_length=50)
    passwd = models.CharField(max_length=50, blank=True)
    dbname = models.CharField(max_length=50)
```

Fuente: datos del proyecto

Gráfica 45 ***Modelo Servicio.*** *Elaboración propia.*

```
class Service(models.Model):
    sources = (
        ('1', 'Consulta SQL'),
        ('3', 'Grupo de servicios'),
    )
    title = models.CharField(max_length=100, unique=True)
    group = models.ForeignKey("self", limit_choices_to={"source": "3"},
    source = models.CharField(choices=sources, max_length=20)
    description = models.TextField(blank=True)
```

Fuente: datos del proyecto

Gráfica 46 ***Modelo SQL.*** *Elaboración propia.*

```
# Modelo de configuracion de servicios de consulta SQL
class SQLQuery(models.Model):
    service = models.OneToOneField(Service, on_delete=models.CASCADE,
                                   limit_choices_to={'source': '1'},
                                   related_name="query", related_query_name="query")
    connection = models.ForeignKey(
        Connection, on_delete=models.CASCADE, blank=True, null=True)
    query_sql = models.TextField(blank=True)
```

Fuente: datos del proyecto

Las historias de usuario para esta iteración son:

- Gestión de conexiones a bases de datos.
- Gestión de servicios SQL.
- Agrupación de servicios.

1.2.5.1.1 Tareas de ingeniería: Primera iteración

Tabla 31 ***Tarea de ingeniería 01.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 01	**Número de Historia:** 01
Nombre de tarea: Análisis y diseño de tablas para guardar datos de conexiones.	
Tipo de tarea: Estructura	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita una estructura que permita guardar los datos de cada una de las conexiones que el usuario administrador necesite para ejecutar y validar un servicio. Se debe tener en cuenta cuales son los datos básicos que se necesitan para generar una conexión a una base de datos de postgreSQL utilizando las librerías que ofrece Python. Para este caso la librería que soporta el API para las conexiones a bases de datos en python es psycopg2.	

Fuente: datos del proyecto

Tabla 32 ***Tarea de ingeniería 02.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 02	**Número de Historia:** 01
Nombre de tarea: Codificación diagrama para conexiones.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.3
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	

Descripción: Se necesita codificar las entidades diseñadas en modelos utilizando el ORM que ofrece Django.

Fuente: datos del proyecto

Tabla 33 ***Tarea de ingeniería 03.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 03	**Número de Historia:** 02
Nombre de tarea: Análisis y diseño de tablas para guardar datos de servicios.	
Tipo de tarea: Estructura.	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita una estructura de base de datos que permita almacenar los datos necesarios para construir y compartir un servicio SQL en el API.	

Fuente: datos del proyecto

Tabla 34 ***Tarea de ingeniería 04.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 04	**Número de Historia:** 02
Nombre de tarea: Codificación de diagrama para servicios.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.3
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita codificar las entidades diseñadas en modelos utilizando el ORM que ofrece Django.	

Fuente: datos del proyecto

Tabla 35 ***Tarea de ingeniería 05.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 05	**Número de Historia:** 03
Nombre de tarea: Análisis y diseño de estructura para agrupar servicios.	
Tipo de tarea: Estructura.	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita una estructura de base de datos para guardar información que brinde la información necesaria para construir tipos GraphQL que representan agrupaciones de servicios.	

Fuente: datos del proyecto

Tabla 36 ***Tarea de ingeniería 06.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 06	**Número de Historia:** 03
Nombre de tarea: Codificación de diagrama de agrupación de servicios.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0.3
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita codificar las entidades diseñadas en modelos utilizando el ORM que ofrece Django.	

Fuente: datos del proyecto

1.2.5.1.2 Pruebas de aceptación: Primera iteración

Tabla 37 ***Prueba de aceptación 01****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 01	**No de historia de usuario:** 1 Gestión de conexiones a bases de datos
Nombre: Prueba de conexión a base de datos de postgres.	
Condiciones de ejecución: • Se debe tener acceso a un servidor de base de datos en postgreSQL. • Se deben utilizar los datos de una conexión previamente creada desde el sitio de administración de Django. • Se debe tener instalada la librería psycopg2 y lista para ser usada dentro del proyecto.	

Entrada/Pasos de Ejecución:
• Obtener los datos para generar la conexión:
  • usuario
  • contraseña
  • IP servidor
  • puerto
  • nombre de base de datos (opcional)
• utilizar los métodos de la clase psycopg2 para abrir y cerrar una conexión dentro de una estructura para el manejo de excepciones try/catch.
• Muestra un mensaje de éxito si la operación termina sin errores, en caso contrario mostrar el mensaje de la excepción.
• Preparar la vista para que ejecute el anterior script al momento de lanzar el servicio de la aplicación.	

Resultado Esperado: Ver por consola el mensaje de éxito si los datos de la conexión son válidos.	
Evaluación de la Prueba: la prueba fue positiva, no hubo complicaciones al utilizar el API para ejecutar conexiones con los argumentos definidos en el modelo.	

Fuente: datos del proyecto

Tabla 38 ***Prueba de aceptación 02****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 02	**No de historia de usuario:** 02 Gestión de servicios SQL.
Nombre: Creación de un servicio SQL	
Condiciones de ejecución: • El modelo de servicios debe estar disponible para crear servicios a través del sitio de administración.	

Entrada/Pasos de Ejecución:
• Registrar los datos básicos para la creación un servicio de tipo SQL:
  • nombre del servicio
  • tipo de servicio (servicio SQL)
  • descripción del servicio
  • conexión
  • consulta SQL
• Consultar los datos del servicio creado utilizando el ORM de Django.
• Realizar la validación de la conexión.
• Ejecutar el SQL del servicio y mostrar por consola los datos de la consulta.
• Preparar la vista para que ejecute el anterior script al momento de lanzar el servicio de la aplicación.	

Resultado Esperado: visualizar por consola la información de la consulta.	
Evaluación de la Prueba: la prueba fue realizada con éxito. Se tiene estructurado el modelo para la ejecución de servicio SQL.	

Fuente: datos del proyecto

Tabla 39 ***Prueba de aceptación 03****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 03	**No de historia de usuario:** 03 Agrupación de servicios.

Nombre: Creación de grupos de servicios

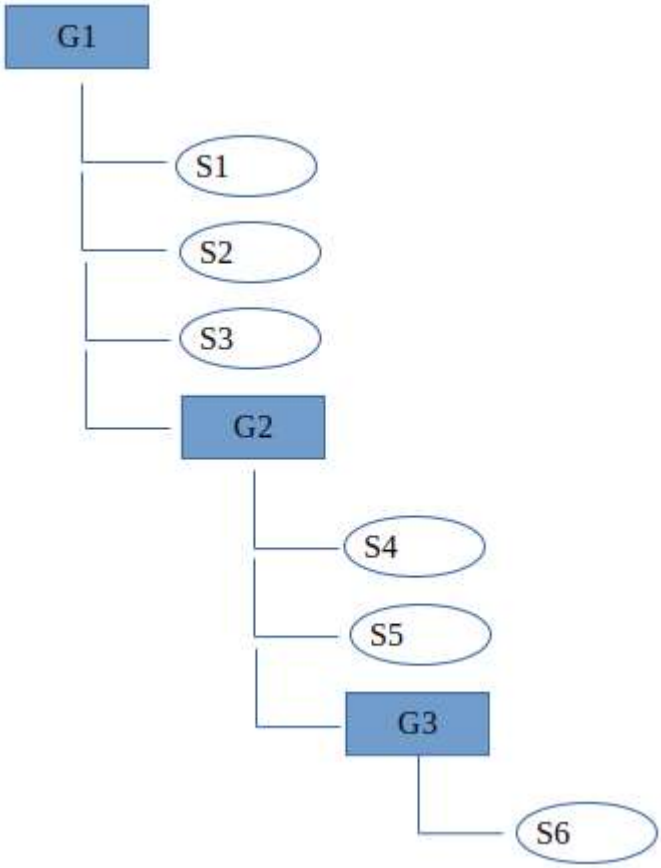
Condiciones de ejecución:

- El modelo de servicios debe estar disponible para crear y modificar servicios a través del sitio de administración.

Entrada/Pasos de Ejecución:

- Para la prueba se propone crear la estructura de servicios representada en la figura x1:

  Estructura de servicios para caso de prueba 03. Fuente: esta investigación.



- Los servicios llamados G1, G2, G3 representan servicios compuestos o grupos de servicios. Los servicios S1, S2, S3, S4, S5 y S6 representan servicios de tipo SQL.
- Para la creación de la estructura planteada se propone seguir los siguientes pasos:
  - Crear cada uno de los servicios de manera individual con los datos básicos para cada uno de ellos desde el administrador de Django:
    - nombre del servicio
    - tipo de servicio (servicio SQL, grupo de servicios)

○ descripción del servicio NOTA: Los servicios de tipo “grupo de servicios”, son los únicos que permiten la asociación. • Asociar los servicios S1, S2, S3 y G2 al servicio G1 • Asociar servicios S4, S5 y G3 al servicio G2 • Asociar servicios S6 al servicio G3 • Una vez asociados los servicios se debe realizar una consulta utilizando el ORM para verificar que la estructura es correcta partiendo del servicio G1 como servicio raíz.
Resultado Esperado: Obtener la misma estructura planteada en el diagrama después de consultar los datos del servicio G1 utilizando el ORM.
Evaluación de la Prueba: La prueba fue realizada con éxito. Se observó que se debe restringir la formación de ciclos al asociar servicios. Por ejemplo: el servicio G3 no debe permitir agrupar el servicio G1.

Fuente: datos del proyecto

1.2.5.2 Segunda iteración

En esta iteración se trabajará en la lógica del negocio, desarrollando el núcleo que se encargará de construir los servicios tomando como insumos la información básica que se podrá recolectar después de tener la estructura base resultado de la primera iteración.

Las historias de usuario para esta iteración son:

- Modelado de servicios.
- Resolución de servicios.

1.2.5.2.1 Tareas de ingeniería: Segunda iteración

Tabla 40 ***Tarea de ingeniería 07.*** *Elaboración propia.*

Tarea de Ingeniería

Número de tarea: 07	**Número de Historia:** 04 Modelado de servicios
Nombre de tarea: Instanciar tipos GraphQL para servicios SQL	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	

Descripción: Se debe generar tipos GraphQL de manera automática a partir de la información registrada para un servicio SQL, instanciando la clase que lo representa utilizando la metaclase “type”.

Se debe tener en cuenta cuales son los elementos que componen una clase en Python.

Ejemplo de clase python. Fuente: esta investigación.

```
class Persona(graphene.ObjectType):
    nombres = graphene.String()
    apellidos = graphene.String()
    identificacion = graphene.String()
    edad = graphene.Ing()
```

En la figura x se puede observar un ejemplo de una clase de la cual se puede distinguir los siguientes elementos:

- Nombre de la clase: Persona
- Herencia de clase: ObjectType
- Atributos de clase: nombre, apellidos, identificación, edad

De la información que se necesitan para construir una clase, el nombre de la clase es un elemento que se puede personalizar o asignar un valor automático teniendo en cuenta cuales son las condiciones que exige python para nombrar una clase, es decir, que debe empezar con una letra mayúscula, no debe contener caracteres especiales, etc.

La clase de la cual se hereda para construir un tipo objeto GraphQL ya está definida por la librería que se utilizará para la implementación de GraphQL en python la cual se llama Graphene.

En cuanto a los atributos de clase se los obtendrá de la consulta SQL que define la información del servicio, cada atributo corresponde al nombre de cada una de las columnas de la consulta SQL como se muestra en la figura x.

Ejemplo consulta SQL. Fuente: esta investigación.

```
SELEC
    nombres,
    apellidos,
    identificacion,
    age(fecha_nacimiento, now()) as edad
FROM
    personas
```

La metaclase “type” como se revisó en el marco teórico, permite crear una instancia de una clase. Para lograr esto “type” recibe 3 parámetros:

1. nombre de la clase
2. clases herencia
3. diccionario de atributos

Por lo tanto la instancia de la clase Persona se realizará como se muestra en la figura x.

Instancia de la clase Persona utilizando metaclases. Fuente: esta investigación.

```
Persona = type("Persona", (graphene.ObjectType,), {"nombres":graphene.String(),
                                                   "apellidos"graphene.String(),
                                                   "identificacion":graphene.String(),
                                                   "edad":graphene.Int()}
                )
```

Se debe tener en cuenta que el nombre de la clase no define el nombre del servicio con el cual será consultado a través de la API. Quien define el nombre del servicio es el nombre del campo del elemento padre que contiene dicho servicio. Se puede observar la figura x para entender cómo se representan los tipos definidos definidos mediante clases python para una consulta GraphQL.

Definición de un esquema GraphQL con su respectiva consulta.

NOTA: Todos los atributos o campos de un tipo serán representados como una cadena de texto, por lo tanto, se usará el tipo escalar String para cada uno de los campos.

Fuente: datos del proyecto

Tabla 41 ***Tarea de ingeniería 08****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 08	**Número de Historia:** 04 Modelado de servicios
Nombre de tarea: Instanciar tipos GraphQL para grupos de servicios	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe generar tipos GraphQL de manera automática a partir de la información registrada para grupos de servicios, instanciando la clase que lo representa utilizando la metaclase “type”. Al igual que los servicios SQL, un grupo de servicios se construye a partir de una clase, pero cada uno de los campos o atributos de clase hace referencia a otros servicios o tipos objeto GraphQL como se muestra en la figura x. Ejemplo de grupo. Fuente: esta investigación.	

```
class Grupo1 (graphene.ObjectType):
    persona = graphene.List(Persona)
    animal = graphene.List(Animal)
```

La instancia de la clase Grupo1 utilizando type se muestra en la figura x.

Instancia de la clase Grupo 1 utilizando metaclases. Fuente: esta investigación.

```
Grupo1 = type("Grupo1", (graphene.ObjectType,), {"persona": graphene.List(Persona),
                                                 "animal": graphene.List(Animal),
                                                })
```

Como se puede observar en la imagen x para poder crear una instancia de la clase Grupo1 se debe tener creada una instancia de cada uno de los tipos o servicios hijos. Para este caso debe existir una instancia de la clase Persona y Animal.

Por lo tanto la construcción de un grupo debe realizarse instanciando primero los servicios SQL ya que no dependen de otras clases y por último crear las clases de los grupo.

Tomando como referencia la figura x1 el orden de creación de los servicios para la estructura propuesta sería el siguiente:

1. S1, S2, S3, S4, S5, S6
2. G3
3. G2
4. G1

Se debe tener en cuenta que el nombre de la clase no define el nombre del servicio con el cual será consultado a través de la API. Quien define el nombre del servicio es el nombre del campo del elemento padre que contiene el servicio, en este caso puede ser el tipo raíz Query o un grupo de servicios. Para comprender de mejor manera

Fuente: datos del proyecto

Tabla 42 ***Tarea de ingeniería 09.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 09	**Número de Historia:** 05 Resolución de servicios
Nombre de tarea: Obtener información de un servicio SQL.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	

Descripción: Se necesita obtener la información de un servicio a partir de su respectiva consulta SQL. La ejecución de la consulta SQL utilizará la conexión establecida en la parametrización del servicio, y cada registro que se obtenga debe presentarse en forma de diccionarios o tuplas mostrando como llaves los nombres de cada uno de los campos de la consulta.

Como ejemplo de la estructura que se necesita obtener se puede observar la figura x que representa el resultado de la ejecución de la consulta SQL de la figura x2.

Consulta SQL. Fuente: Esta investigación.

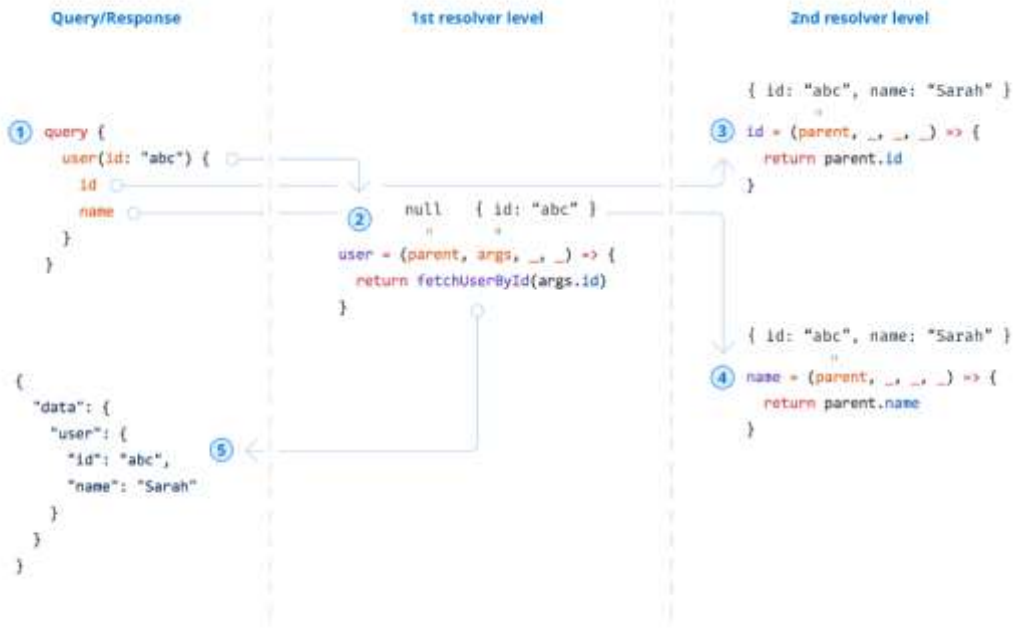
```
SELEC
    nombres,
    apellidos,
    identificacion,
    age(fecha_nacimiento, now()) as edad
FROM
    personas
```

Resultado de la ejecución de una consulta SQL de la figura x2. Fuente: esta investigación.

```
[{'nombres': 'Juan', 'apellidos': 'Perez', 'identificacion': '0001', 'edad': 25},
 {'nombres': 'John', 'apellidos': 'Getial', 'identificacion': '0002', 'edad': 25}
]
```

Fuente: datos del proyecto

Tabla 43 ***Tarea de ingeniería 10.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 10	**Número de Historia:** 05 Resolución de servicios
Nombre de tarea: Crear funciones de resolución.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita crear las funciones de resolución para cada servicio de manera automática. El proceso de resolución se debe hacer de manera jerárquica teniendo en cuenta el flujo de ejecución que un servidor GraphQL realiza para resolver la información de una consulta, como se muestra en la imagen x. Ejemplo de ejecución de una consulta GraphQL. Fuente:    Tomando como ejemplo la consulta de la figura x, para el servicio “user” en el cual se solicitan dos campos: id y name, la información que se necesita se resuelve ejecutando 3 funciones de resolución.

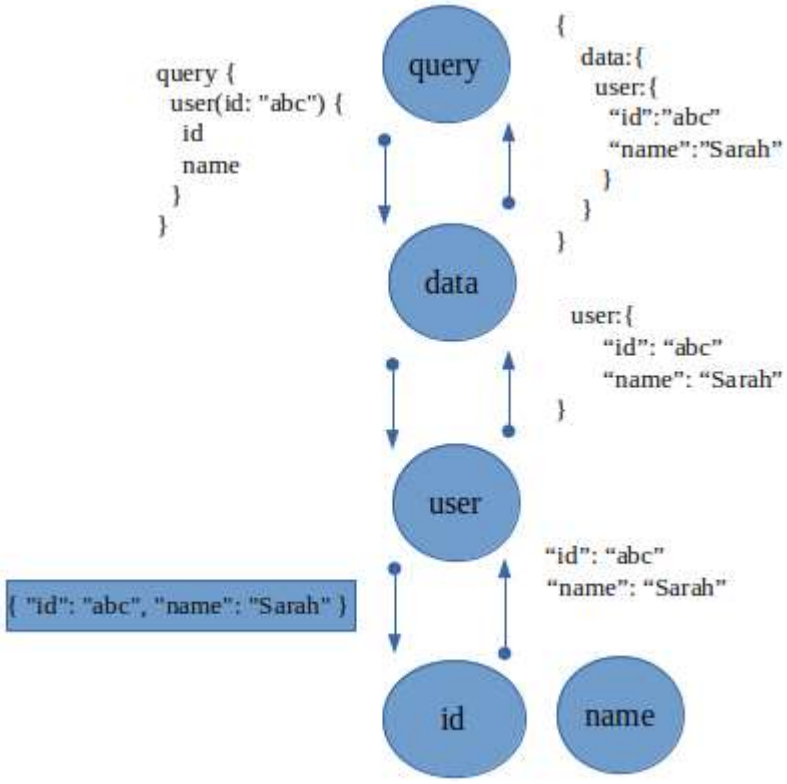
En el primer nivel de resolución se necesita obtener la información para el servicio consultado, en este caso es “user”. La función de resolución para “user” se encarga de obtener la información que representa un objeto “user” con id “abc”, por lo cual, la forma cómo se representan los datos de una consulta SQL se puede asimilar como un objeto cuyos campos corresponden a cada uno de sus atributos como se muestra en la figura x.

Ejemplo del objeto que se obtiene en la ejecución de la función de resolución para el campo raíz user. Fuente:

```
{ "id": "abc", "name": "Sarah" }
```

El segundo nivel de resolución recibe la información obtenida en el nivel anterior a través del parámetro parent. Las funciones de resolución que se ejecutan para este nivel obtienen el dato correspondiente para su respectivo campo(atributo), retornando parent.id y parent.name para cada uno de ellos.

Flujo de ejecución de una consulta GraphQL. Fuente: esta investigación



Fuente: datos del proyecto

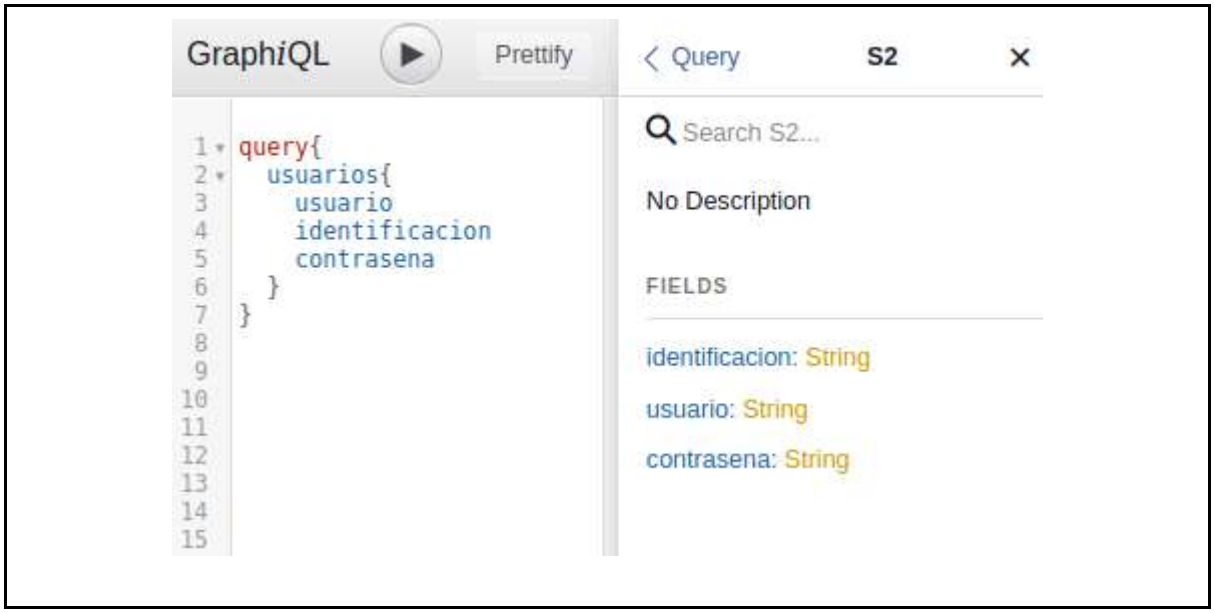
1.2.5.2.2 Pruebas de aceptación: Segunda iteración

Para las pruebas que se realizarán en esta segunda iteración se tendrá a disposición la URL de permitirá el acceso a la API GraphQL. Se necesitará realizar la configuración básica de la herramienta Graphiql para contar con la interfaz web que ofrece un entorno para la verificación de esquemas Graphql, ejecución de consultas y visualización de respuestas.

Tabla 44 ***Prueba de aceptación 04****. Elaboración propia.*

<table>
<tr><td colspan="2">Caso de prueba</td></tr>
<tr><td>No de caso de prueba: 04</td><td>No de historia de usuario: 04 Modelado de servicios.</td></tr>
<tr><td colspan="2">Nombre: Crear un servicio SQL.</td></tr>
<tr><td colspan="2">Condiciones de ejecución:
Tener disponible la interfaz Graphiql para la validación del esquema.
Contar con una conexión válida a una base de datos de prueba.</td></tr>
<tr><td colspan="2">Entrada/Pasos de Ejecución:
Crear un nuevo servicio teniendo en cuenta los siguientes parámetros:
Nombre del servicio: usuarios
Campos requeridos: identificación, usuario y contraseña
Diseñar consulta SQL.
Validar estado de la conexión.
Validar consulta SQL.
Verificar en el esquema GraphQL la existencia del nuevo servicio.</td></tr>
<tr><td colspan="2">Resultado Esperado: Visualizar en el esquema GraphQL el tipo creado para el nuevo servicio.</td></tr>
<tr><td colspan="2">Evaluación de la Prueba: La prueba se realizó de manera exitosa. Se debe tener en cuenta que una vez creado o modificado un servicio se necesita reiniciar la aplicación para que los nuevos servicios sean creados y actualizados en el esquema GraphQL.

Resultado obtenido de la verificación de la prueba de aceptación 04. Fuente: esta investigación.</td></tr>
</table>



Fuente: datos del proyecto

Tabla 45 ***Prueba de aceptación 05****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 05	**No de historia de usuario:** 04 Modelado de servicios.
Nombre: Creación de grupos.	
Condiciones de ejecución: • Tener disponible la interfaz Graphiql para la validación del esquema. • Contar con el servicio SQL usuarios creado en la prueba de aceptación 04.	
Entrada/Pasos de Ejecución: • Crear un nuevo grupo de servicios teniendo en cuenta los siguientes parámetros: ○ Nombre del grupo: administrativos • Asociar el servicio usuario al nuevo grupo. • Verificar en el esquema GraphQL la existencia del nuevo grupo.	
Resultado Esperado: Visualizar en el esquema GraphQL el tipo creado para el nuevo grupo.	
Evaluación de la Prueba: La prueba se realizó de manera exitosa. Se debe tener en cuenta que no pueden existir grupos vacíos, esto quiere decir que en algún	

punto del árbol jerárquico del esquema GraphQL se debe contener un servicio SQL válido. Resultado obtenido de la verificación de la prueba de aceptación 05. Fuente: esta investigación. 

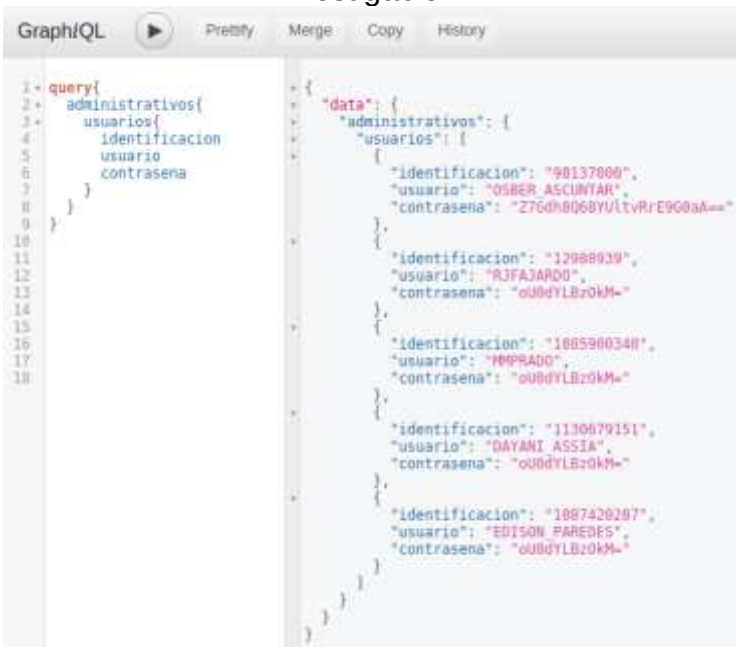
Fuente: datos del proyecto

Tabla 46 ***Prueba de aceptación 06****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 06	**No de historia de usuario:** 05 Resolución de servicios.
Nombre: Ejecución de consultas.	
Condiciones de ejecución: • Contar con la interfaz Graphiql para la ejecución y visualización de consultas. • Contar el grupo “administrativos” creado en la prueba de aceptación 05. • Contar con una conexión válida a una base de datos de prueba.	
Entrada/Pasos de Ejecución: • Diseñar la consulta GraphQL para obtener los datos del servicio usuarios • Ejecutar la consulta.	
Resultado Esperado: Obtener los datos del servicio usuarios.	
Evaluación de la Prueba: La prueba se realizó de manera exitosa. Se debe tener en cuenta que el servicio ‘usuarios’ está encapsulado dentro de un grupo de	

servicios, por lo cual la entrada a la información del servicio se hace a través del servicio que lo contiene en este caso es el servicio ‘administrativos’.

Resultado obtenido de la ejecución de la prueba de aceptación 06. Fuente: esta investigación.



Fuente: datos del proyecto

1.2.5.3 Tercera iteración

En esta iteración se integrarán las opciones de seguridad tanto para controlar el acceso a la información compartida (servicios) a través de la API, y la seguridad de la plataforma que se utilizará para la administración de los servicios.

Las historias de usuario planeadas para esta iteración son:

- Autenticación de clientes Graphql.
- Gestión de roles.
- Autorización a servicios.

1.2.5.3.1 Tareas de ingeniería: Tercera iteración

Tabla 47 ***Tarea de ingeniería 11.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 11	**Número de Historia:** 06 Autenticación de clientes GraphQL.
Nombre de tarea: Crear servicios de autenticación.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita exponer un tipo GraphQL que permita la autenticación de un usuario. El servicio debe recibir la información necesaria para realizar la autenticación del usuario y generar un token si los datos de autenticación son correctos. Para la generación y validación de un token se utilizará el estándar JWT, el cual puede ser integrado con GraphQL y Django usando la librería que ofrece python llamada django-GraphQL-jwt. La integración de nuevas formas de autenticación en Django se hace a través los backend de autenticación, los cuales son funciones que se registran en la configuración de la aplicación para validar la información de un usuario que quiere ingresar a la aplicación. Estos backend pueden personalizarse para obtener la información de un usuario de la fuente de información que se necesite, para este caso se utilizará una conexión a una base de datos o una conexión a un servidor LDAP.	

Fuente: datos del proyecto

Tabla 48 ***Tarea de ingeniería 12.*** *Elaboración propia.*

Tarea de Ingeniería

Número de tarea: 12	**Número de Historia:** 06 Autenticación de clientes GraphQL.
Nombre de tarea: Validar credenciales de usuario en base de datos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita contar con un backend de autenticación que permita verificar las credenciales de un usuario utilizando una base de datos. Se debe especificar una consulta SQL que reciba las credenciales de usuario que se proporcionarán a través del servicio GraphQL. También se debe especificar qué conexión se utilizará para ejecutar la consulta SQL para la autenticación. Los datos que se recibirán a través de la API son el usuario y la contraseña, los cuales deben ser adicionados a la consulta SQL usando los denominados preparedstatement para garantizar una ejecución segura. Debe integrarse esta forma de validación de un usuario con JWT para generar el token en caso de que la autenticación sea correcta.	

Fuente: datos del proyecto

Tabla 49 ***Tarea de ingeniería 13.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 13	**Número de Historia:** 06 Autenticación de clientes GraphQL.
Nombre de tarea: Enlazar un servidor LDAP.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**

Programador responsable: John Getial, David Ceron
Descripción: Se debe permitir parametrizar y enlazar conexiones a un servidor LDAP para la ejecución de búsquedas y validaciones de usuarios.

Fuente: datos del proyecto

Tabla 50 ***Tarea de ingeniería 14.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 14	**Número de Historia:** 06 Autenticación de clientes GraphQL.
Nombre de tarea: Validar credenciales de usuario en directorio activo (LDAP).	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita contar con un backend de autenticación que permita verificar las credenciales de un usuario utilizando un servidor LDAP. Se debe especificar una plantilla o dn que recibe las credenciales de usuario que se proporcionarán a través del servicio GraphQL para su autenticación. Los datos que se recibirán a través de la API son el usuario y la contraseña, los cuales deben ser adicionados a la plantilla dn para que sean validados en los datos del directorio activo. Debe integrarse esta forma de validación de un usuario con JWT para generar el token en caso de que la autenticación sea correcta.	

Fuente: datos del proyecto

Tabla 51 ***Tarea de ingeniería 15.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 15	**Número de Historia:** 07 Gestión de roles.

Nombre de tarea: Creación de roles para usuarios en bases de datos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita poder crear roles que permitan validar usuarios que han sido autenticados utilizando la información de una base datos. La validación de los usuarios debe realizarse a través de consultas SQL que reciben el nombre de usuario y comprueban las condiciones que el usuario debe cumplir para un rol.	

Fuente: datos del proyecto

Tabla 52 ***Tarea de ingeniería 16****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 16	**Número de Historia:** 07 Gestión de roles.
Nombre de tarea: Creación de roles para usuarios en directorio activo (LDAP).	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita poder crear roles que permitan validar usuarios que han sido autenticados utilizando la información de un directorio activo. La validación de los usuarios debe realizarse a través de plantillas dn que reciben el nombre de usuario y comprueban las condiciones que el usuario debe cumplir para un rol.	

Fuente: datos del proyecto

Tabla 53 ***Tarea de ingeniería 17.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 17	**Número de Historia:** 08 Autorización a servicios.
Nombre de tarea: Asociar roles a servicios.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita poder asociar roles a la información de un servicio. Los roles asociados a un servicio pueden ser roles para base datos o roles para un directorio activo.	

Fuente: datos del proyecto

Tabla 54 ***Tarea de ingeniería 18****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 18	**Número de Historia:** 08 Autorización a servicios.
Nombre de tarea: Verificar autorizaciones de usuarios.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se necesita poder validar si un usuario tiene acceso a la información de un servicio al comprobar si cumple con cada uno de los roles asociados al servicio. Se restringe el acceso a la información de un servicio si no se cumple con las condiciones para un rol.	

Fuente: datos del proyecto

1.2.5.3.2 Pruebas de aceptación: Tercera iteración

Tabla 55 ***Prueba de aceptación 07****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 07	**No de historia de usuario:** 06 Autenticación de clientes GraphQL.
Nombre: Autenticar usuarios en base de datos a través de un servicio GraphQL.	
Condiciones de ejecución: • Debe existir una conexión válida a una base de datos. • Definir la tabla y los campos de la base de datos que se utilizarán para validar el nombre de usuario y contraseña • Se debe tener parametrizada una plantilla SQL válida. • Debe estar configurado el backend de autenticación para ejecutar la validación de las credenciales de usuario de acuerdo a los parámetros establecidos. • Se debe contar con el servicio GraphQL que recibirá las credenciales de usuario.	
Entrada/Pasos de Ejecución: • Ingresar al entorno de pruebas GraphiQL. • Utilizar el servicio GraphQL para verificar las credenciales de un usuario.	
Resultado Esperado: Obtener un token si las credenciales de un usuario son correctas.	
Evaluación de la Prueba: La prueba se realizó de manera exitosa.	

Fuente: datos del proyecto

Tabla 56 ***Prueba de aceptación 08****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 08	**No de historia de usuario:** 06 Autenticación de clientes GraphQL.
Nombre: Autenticar usuarios en servidor LDAP a través de un servicio GraphQL.	

Condiciones de ejecución: • Debe existir un servidor LDAP activo • Definir el dn de búsqueda que se utilizará para validar el nombre de usuario y contraseña • Debe estar configurado el backend de autenticación para ejecutar la validación de las credenciales de usuario de acuerdo a los parámetros establecidos para este tipo de autenticación. • Se debe contar con el servicio GraphQL que se encargará de recibir las credenciales de usuario.
Entrada/Pasos de Ejecución: • Ingresar al servicio GraphQL para autenticar un usuario utilizando el entorno de pruebas GraphiQL.
Resultado Esperado: Obtener un token si las credenciales de un usuario son correctas.
Evaluación de la Prueba: La prueba se realizó de manera exitosa. Se debe asegurar que exista solamente una forma de autenticación activa, es decir, si se establece la configuración para base de datos se debe deshabilitar la autenticación LDAP y viceversa. Esto para evitar ambigüedades en la búsqueda de usuarios.

Fuente: datos del proyecto

Tabla 57 ***Prueba de aceptación 09****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 09	**No de historia de usuario:** 08 Autorización a servicios.
Nombre: Control de acceso a servicios.	
Descripción: Se restringirá el acceso a la información de un servicio SQL o grupo de servicios mediante la parametrización de roles SQL.	
Condiciones de ejecución: • Deben existir servicios GraphQL con información para consultar. • El cliente que intentará acceder a los servicios debe contar con un token válido obtenido previamente en la autenticación. • Deben haber creado roles y estar asociados a cada uno de los servicios activos.	

Entrada/Pasos de Ejecución: • Preparar la consulta GraphQL para los servicios expuestos. • Adicionar al encabezado de la petición HTML el tipo de autenticación y el token de validación. • Adicionar al cuerpo de la petición HTML el JSON con la consulta GraphQL. • Enviar la petición HTML al punto final preparado con la API GraphQL.
Resultado Esperado: Obtener la información del servicio si el cliente cumple con la validación del rol o roles asociados al servicio.
Evaluación de la Prueba: La prueba se realizó de manera exitosa. Se debe tener en cuenta que los clientes que no están autenticados son tratados como clientes o usuarios anónimos que tendrán acceso solamente a servicios públicos, es decir, servicios que no están restringidos o asociados a ningún rol.

Fuente: datos del proyecto

1.2.5.4 Cuarta iteración

En esta última iteración se entregará la plataforma para el usuario que se encargará de administrar todos los elementos para el funcionamiento de la API Graphql, como son las conexiones a las bases de datos, servicios SQL, grupos, configuración de la autenticación y autorizaciones a servicios.

Las historias de usuario planeadas para esta iteración son:

- Interfaz de usuario para conexiones.
- Interfaz de seguridad.
- Interfaz de servicios.

1.2.5.4.1 Tareas de ingeniería: Cuarta iteración

Tabla 58 ***Tarea de ingeniería 19.*** *Elaboración propia.*

Tarea de Ingeniería

Número de tarea: 19	**Número de Historia:** 09 Interfaz de usuario para conexiones.
Nombre de tarea: Interfaz para gestión de conexiones.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita crear, modificar y eliminar conexiones.	

Fuente: datos del proyecto

Tabla 59 ***Tarea de ingeniería 20****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 20	**Número de Historia:** 10 Interfaz de seguridad.
Nombre de tarea: Interfaz de autenticación SQL.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita parametrizar el tipo de autenticación por base de datos. Se debe dar la opción de establecer este tipo de autenticación como la predeterminada para autenticar clientes de los servicios de la API.	

Fuente: datos del proyecto

Tabla 60 ***Tarea de ingeniería 21****. Elaboración propia.*

Tarea de Ingeniería

Número de tarea: 21	**Número de Historia:** 10 Interfaz de seguridad.
Nombre de tarea: Interfaz de autenticación LDAP.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita parametrizar el tipo de autenticación en directorio activo (LDAP). Se debe dar la opción de establecer este tipo de autenticación como la predeterminada para autenticar clientes de los servicios de la API.	

Fuente: datos del proyecto

Tabla 61 ***Tarea de ingeniería 21****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 21	**Número de Historia:** 10 Interfaz de seguridad.
Nombre de tarea: Interfaz de roles SQL.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita crear, modificar y eliminar roles SQL para una base de datos.	

Fuente: datos del proyecto

Tabla 62 ***Tarea de ingeniería 22****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 22	**Número de Historia:** 10 Interfaz de seguridad.

Nombre de tarea: Interfaz de roles LDAP.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita crear, modificar y eliminar roles para un directorio activo.	

Fuente: datos del proyecto

Tabla 63 ***Tarea de ingeniería 23.*** *Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 23	**Número de Historia:** 10 Interfaz de seguridad.
Nombre de tarea: Interfaz de roles LDAP.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita crear, modificar y eliminar roles para un directorio activo.	

Fuente: datos del proyecto

Tabla 64 ***Tarea de ingeniería 24****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 24	**Número de Historia:** 11 Interfaz de servicios.
Nombre de tarea: Desarrollo de interfaz de información general de servicios.	

Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita crear y modificar la información básica de un servicio, como: • Nombre del servicio • Descripción del servicio. • Tipo de servicio (servicio SQL, grupo)	

Fuente: datos del proyecto

Tabla 65 ***Tarea de ingeniería 25****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 25	**Número de Historia:** 11 Interfaz de servicios.
Nombre de tarea: Desarrollo de interfaz de información avanzada de servicios SQL.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita parametrizar un servicio de tipo SQL. El usuario podrá configurar la siguiente información del servicio: • Conexión a la base de datos. • Consulta SQL • Filtros o condiciones de la consulta. • Nombre de la clase. • Nombre del tipo para la consulta GraphQL. Además, la interfaz debe permitir validar la consulta SQL y cada uno de los campos que van a ser modelados para un servicio GraphQL.	

Fuente: datos del proyecto

Tabla 66 ***Tarea de ingeniería 26****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 26	**Número de Historia:** 11 Interfaz de servicios.
Nombre de tarea: Desarrollo de interfaz de información avanzada para grupos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita asociar a un grupo diferentes servicios que ya han sido creados.	

Fuente: datos del proyecto

Tabla 67 ***Tarea de ingeniería 27****. Elaboración propia.*

Tarea de Ingeniería	
Número de tarea: 27	**Número de Historia:** 11 Interfaz de servicios.
Nombre de tarea: Desarrollo de interfaz de seguridad de servicios.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.7
Fecha inicio:	**Fecha fin:**
Programador responsable: John Getial, David Ceron	
Descripción: Se debe proporcionar una interfaz de usuario que permita asociar los roles necesarios a un servicio para restringir el acceso a su información.	

Fuente: datos del proyecto

1.2.5.4.2 Pruebas de aceptación: Cuarta iteración

Tabla 68 ***Prueba de aceptación 10.*** *Elaboración propia.*

Caso de prueba	
No de caso de prueba: 10	**No de historia de usuario:** 09 Interfaz de usuario para conexiones.
Nombre: Registro de conexiones.	
Descripción: Se utilizará la interfaz de usuario para el registro de una conexión a una base de datos SQL.	
Condiciones de ejecución: • Tener acceso a la interfaz de usuario.	
Entrada/Pasos de Ejecución: • Ingresar a la ventana de gestión de conexiones. • Ingresar la información necesaria para la creación de una conexión. • Guardar la información de la nueva conexión.	
Resultado Esperado: Visualizar en un listado la nueva conexión creada.	
Evaluación de la Prueba: La prueba se realizó de manera exitosa. Se debe tener en cuenta que las credenciales del usuario que se conectará al gestor de base de datos deben ser correctas para que se pueda finalizar el registro de una nueva conexión.	

Fuente: datos del proyecto

Tabla 69 ***Prueba de aceptación 11****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 11	**No de historia de usuario:** 09 Interfaz de usuario para conexiones.
Nombre: Modificación y eliminación de conexiones.	
Descripción: Se utilizará la interfaz de usuario para modificar y eliminar una conexión.	

Condiciones de ejecución: • Tener acceso a la interfaz de usuario. • Contar con conexiones registradas.
Entrada/Pasos de Ejecución: • Ingresar a la ventana que muestra la lista de conexiones. • Seleccionar la conexión a modificar y/o eliminar. • Modificar los datos de una conexión y guardar los cambios. • Eliminar una conexión seleccionada
Resultado Esperado: Visualizar la información de la conexión modificada y visualizar la nueva lista de las conexiones registradas.
Evaluación de la Prueba: La prueba se realizó de manera exitosa. Se debe tener en cuenta que las conexiones que están siendo utilizadas por algún servicio no se pueden eliminar.

Fuente: datos del proyecto

Tabla 70 ***Prueba de aceptación 12****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 12	**No de historia de usuario:** 10 Interfaz de seguridad.
Nombre: Parametrización de seguridad con base de datos.	
Descripción: Se utilizará la interfaz de usuario para ingresar la información necesaria para parametrizar la autenticación de clientes GraphQL en bases de datos.	
Condiciones de ejecución: • Tener acceso a la interfaz de usuario. • Contar con conexiones registradas.	
Entrada/Pasos de Ejecución: • Ingresar a la ventana de parametrización de seguridad • Parametrizar el tipo de autenticación SQL. • Guardar la información registrada. • Establecer la autenticación por base de datos como configuración predeterminada.	

Resultado Esperado: Visualizar los parámetros establecidos en los archivos de configuración de Django.
Evaluación de la Prueba: La prueba se realizó de manera exitosa.

Fuente: datos del proyecto

Tabla 71 ***Prueba de aceptación 13.*** *Elaboración propia.*

Caso de prueba	
No de caso de prueba: 12	**No de historia de usuario:** 10 Interfaz de seguridad.
Nombre: Parametrización de seguridad LDAP.	
Descripción: Se utilizará la interfaz de usuario para ingresar la información necesaria para parametrizar la autenticación de clientes GraphQL a través de un servidor LDAP	
Condiciones de ejecución: • Tener acceso a la interfaz de usuario. • Se debe tener acceso a un servidor LDAP.	
Entrada/Pasos de Ejecución: • Ingresar a la ventana de parametrización de seguridad • Parametrizar el tipo de autenticación LDAP. • Guardar la información registrada. • Establecer la autenticación por medio de un servidor LDAP como configuración predeterminada.	
Resultado Esperado: Visualizar los parámetros establecidos en los archivos de configuración de Django.	
Evaluación de la Prueba: La prueba se realizó de manera exitosa.	

Fuente: datos del proyecto

Tabla 72 ***Prueba de aceptación 13****. Elaboración propia.*

Caso de prueba

No de caso de prueba: 13	**No de historia de usuario:** 10 Interfaz de seguridad.
Nombre: Gestión de roles SQL.	
Descripción: Se utilizará la interfaz de usuario para la creación de roles para validar usuarios en bases de datos.	
Condiciones de ejecución: • Tener acceso a la interfaz de usuario. • Contar con conexiones registradas.	
Entrada/Pasos de Ejecución: • Ingresar a la ventana de gestión de roles. • Crear un rol SQL. • Guardar la información registrada.	
Resultado Esperado: Visualizar en la lista de roles la información del nuevo rol registrado.	
Evaluación de la Prueba: La prueba se realizó de manera exitosa.	

Fuente: datos del proyecto

Tabla 73 ***Prueba de aceptación 14****. Elaboración propia.*

Caso de prueba	
No de caso de prueba: 14	**No de historia de usuario:** 10 Interfaz de seguridad.
Nombre: Gestión de roles LDAP.	
Descripción: Se utilizará la interfaz de usuario para la creación de roles para validar usuarios en un directorio activo (LDAP).	
Condiciones de ejecución: • Tener acceso a la interfaz de usuario. • Se debe tener acceso a un servidor LDAP.	
Entrada/Pasos de Ejecución: • Ingresar a la ventana de gestión de roles. • Crear un rol LDAP.	

• Guardar la información registrada.
Resultado Esperado: Visualizar en la lista de roles la información del nuevo rol registrado.
Evaluación de la Prueba: La prueba se realizó de manera exitosa.

Prueba de aceptación 15. Fuente: esta investigación

Caso de prueba	
No de caso de prueba: 15	**No de historia de usuario:** 11 Interfaz de servicios.
Nombre: Crear un servicio SQL.	
Descripción: Se utilizará la interfaz de usuario para la creación y parametrización de un servicio SQL.	
Condiciones de ejecución: • Tener acceso a la interfaz de usuario. • Se debe tener creada una conexión a una base de datos.	
Entrada/Pasos de Ejecución: • Ingresar a la ventana de creación de servicios. • Ingresar la información básica para un servicio SQL. • Parametrizar la consulta SQL para el nuevo servicio. • Reiniciar el servicio de la aplicación. • Ingresar al entorno de pruebas GraphiQL para consultar el nuevo servicio creado.	
Resultado Esperado: Obtener la información del nuevo servicio a través de la consulta GraphQL.	
Evaluación de la Prueba: La prueba se realizó de manera exitosa. Se debe proporcionar una opción para reiniciar el servidor de aplicaciones de Django desde la interfaz del usuario.	

Fuente: datos del proyecto

Tabla 74 ***Prueba de aceptación 16.*** *Elaboración propia.*

Caso de prueba

No de caso de prueba: 16	**No de historia de usuario:** 11 Interfaz de servicios.
Nombre: Crear un grupo de servicios.	
Descripción: Se utilizará la interfaz de usuario para la creación y parametrización de un grupo.	
Condiciones de ejecución: • Tener acceso a la interfaz de usuario. • Se necesita contar con servicios ya creados en las pruebas anteriores.	
Entrada/Pasos de Ejecución: • Ingresar a la ventana de creación de servicios. • Ingresar la información básica para un grupo. • Guardar la información básica del nuevo servicio. • Asociar los servicios disponibles al grupo creado. • Reiniciar el servicio de la aplicación. • Ingresar al entorno de pruebas GraphiQL para consultar el nuevo grupo creado.	
Resultado Esperado: Obtener la información de los servicios asociados al nuevo grupo a través de la consulta GraphQL.	
Evaluación de la Prueba: La prueba se realizó de manera exitosa.	

Fuente: datos del proyecto

Tabla 75 ***Prueba de aceptación 17.*** *Elaboración propia.*

Caso de prueba	
No de caso de prueba: 17	**No de historia de usuario:** 11 Interfaz de servicios.
Nombre: Seguridad de servicios.	
Descripción: Se utilizará la interfaz de usuario para asociar roles a los servicios creados para restringir el acceso.	
Condiciones de ejecución: • Tener acceso a la interfaz de usuario. • Se necesita contar con servicios ya creados en las pruebas anteriores.	

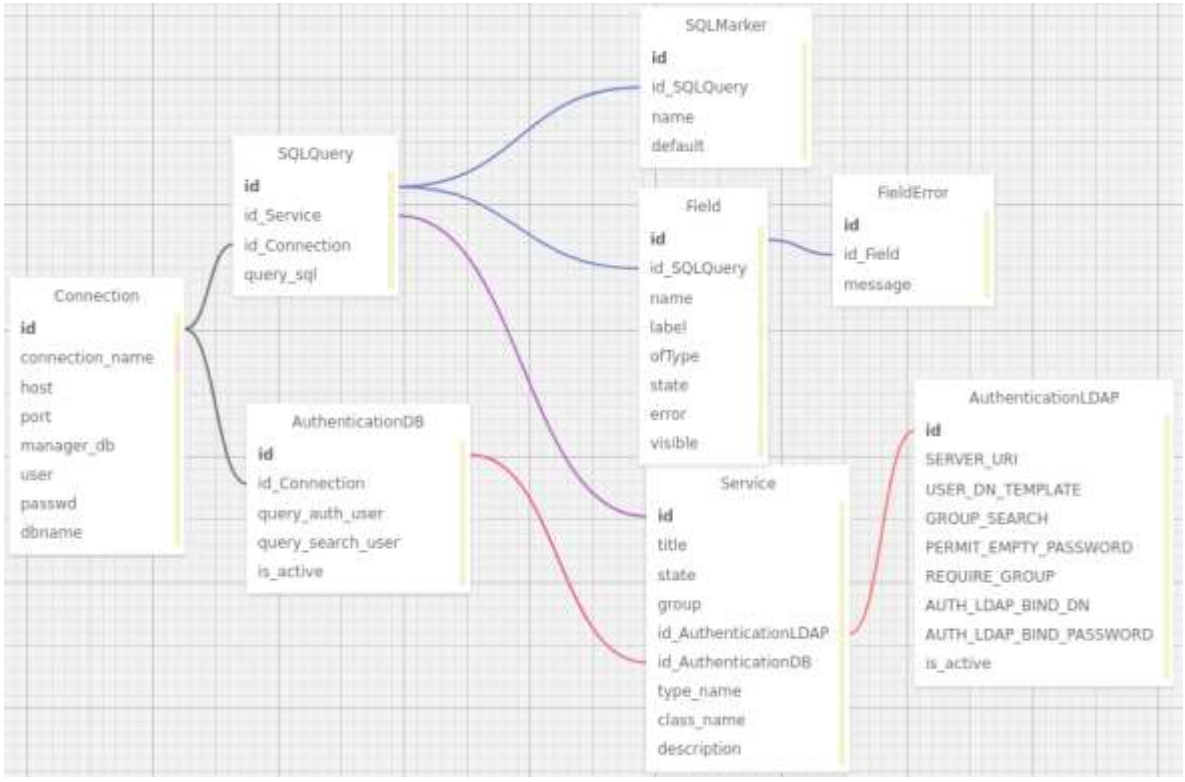
• Se necesita tener parametrizada y activas una de las configuraciones para la autenticación de usuarios.
Entrada/Pasos de Ejecución: • Ingresar a la ventana de seguridad de servicios. • Asociar los roles necesarios a la configuración de un servicio.
Resultado Esperado: Obtener la lista de roles asociados al servicio.
Evaluación de la Prueba: La prueba se realizó de manera exitosa.

Fuente: datos del proyecto

1.2.6 Presentación del aplicativo

Como resultado del proceso de desarrollo, se obtiene una aplicación web desarrollada con el framework Django, los modelos creados pueden verse en la gráfica 43.

Gráfica 47 ***Estructura final de base de datos****. Elaboración propia.*



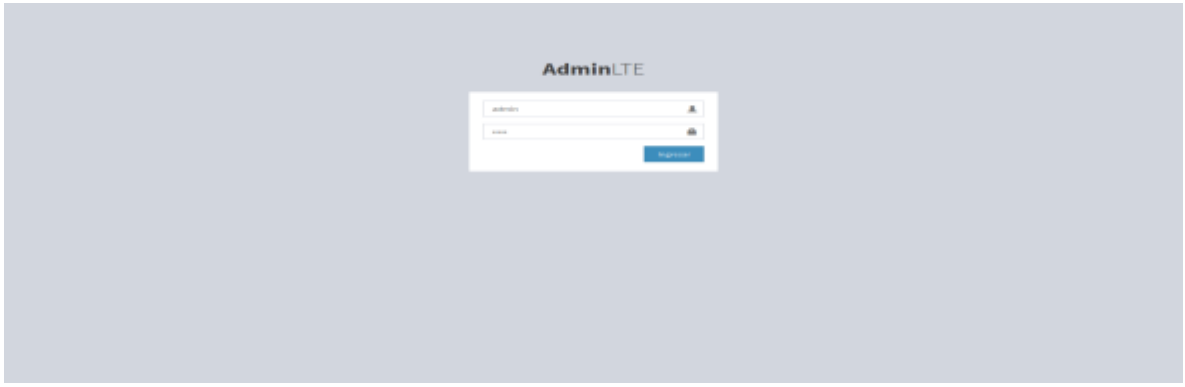
Fuente: datos del proyecto

En el anexo 12 se encuentra una copia del código fuente de la aplicación, con un manual de instalación en el anexo 11 y un manual de uso detallado en el anexo 13. Además, la aplicación fue instalada en un servidor de pruebas de la Universidad de Nariño, y se encuentra disponible a través de este link. Se puede acceder usando las siguientes credenciales:

usuario: admin

contraseña: admin

Gráfica 48 ***Ventana de inicio de sesión****. Elaboración propia.*



Fuente: datos del proyecto

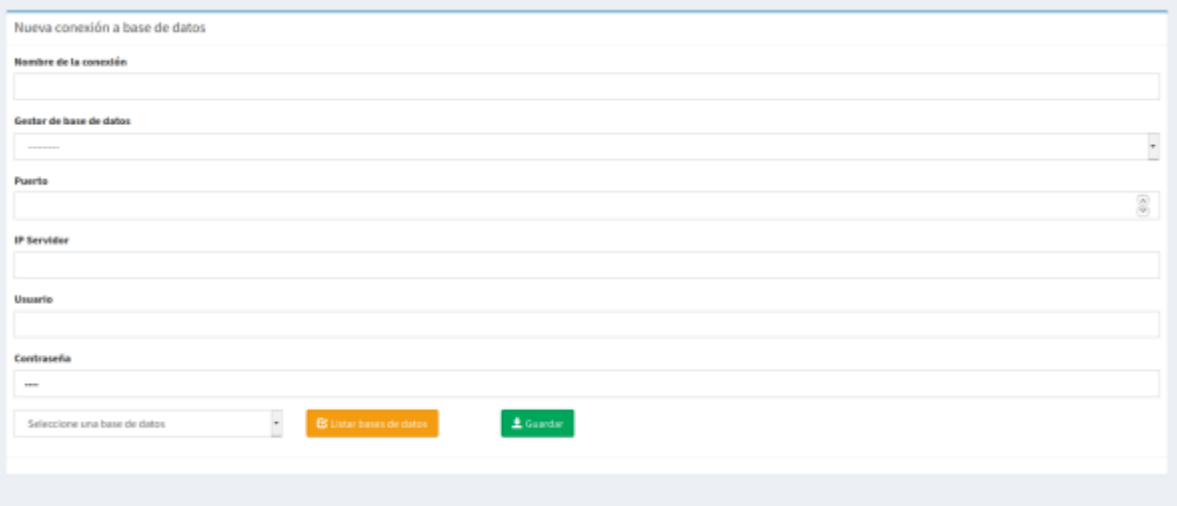
Gráfica 49 ***Ventana del menú principal.*** *Elaboración propia.*

Fuente: datos del proyecto

1.2.6.1 Conexiones a bases de datos

La aplicación permite gestionar diferentes conexiones a bases de datos, actualmente se ofrece soporte para PostgreSQL, MySQL y Oracle. Estas conexiones se configuran una vez y quedan disponibles para obtener datos a partir de ellas en la construcción de servicios para la API.

Gráfica 50 ***Ventana de conexiones****. Elaboración propia.*



Fuente: datos del proyecto

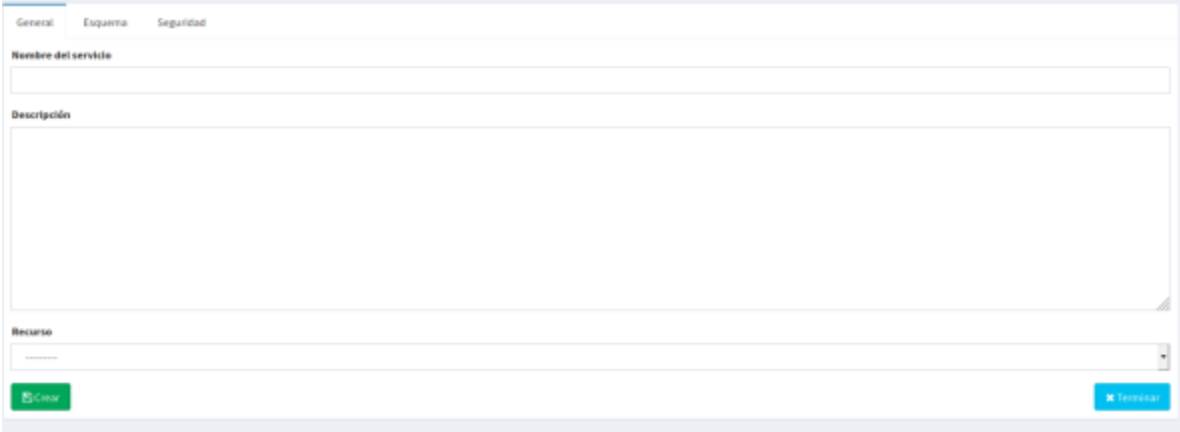
Gráfica 51 ***Ventana de lista de conexiones.*** *Elaboración propia.*

Fuente: datos del proyecto

1.2.6.2 Creación de servicios

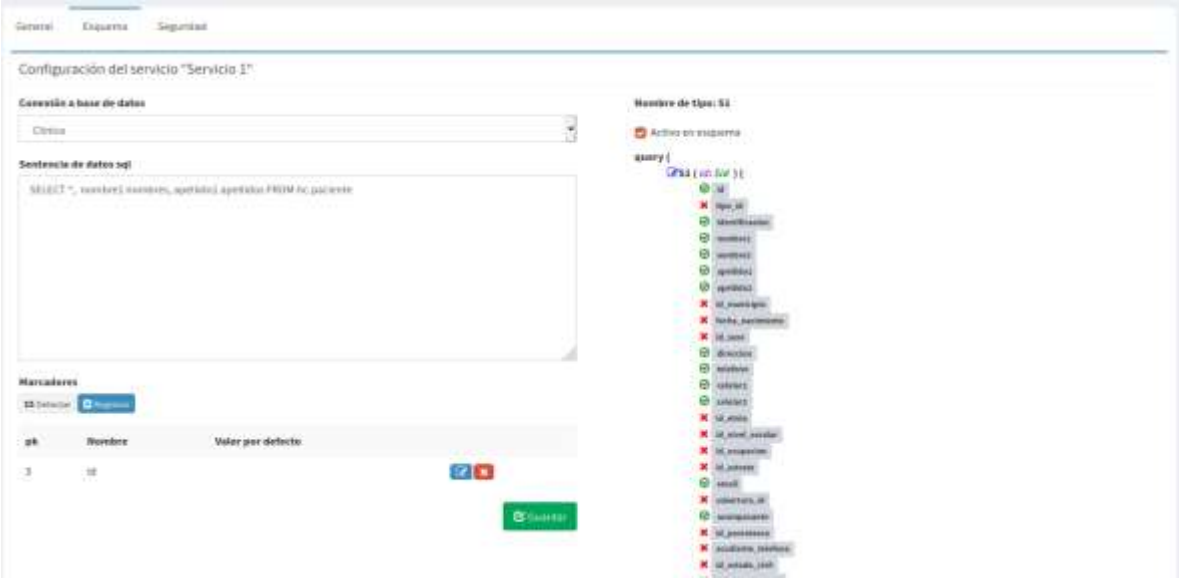
La aplicación permite gestionar servicios mediante consultas SQL que son esquematizados en un tipo GraphQL para que puedan ser consultado a través de la API. Estas consultas SQL usan las conexiones a bases de datos configuradas, y son capaces de acceder a toda la información disponible en ellas.

Gráfica 52 ***Ventana de creación de servicios.*** *Elaboración propia.*



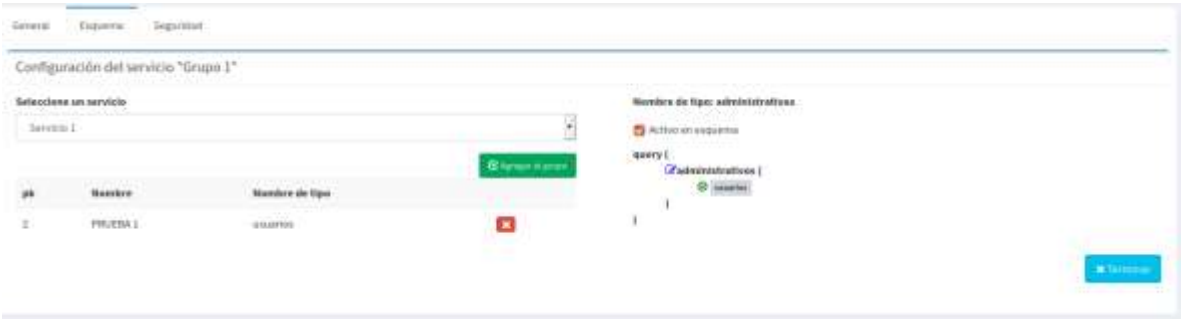
Fuente: datos del proyecto

Gráfica 53 ***Ventana de parametrización de servicios SQL.*** *Elaboración propia.*



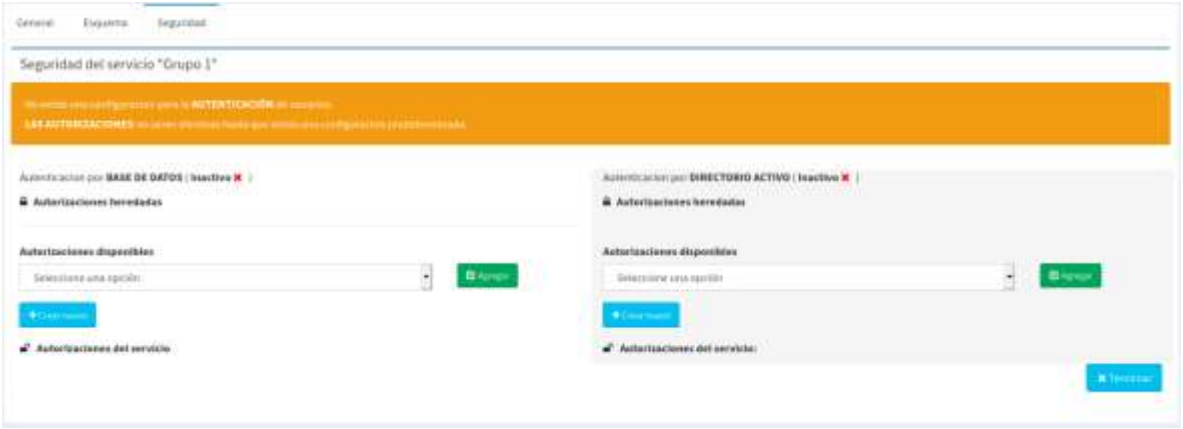
Fuente: datos del proyecto

Gráfica 54 ***Ventana de parametrización de grupos.*** *Elaboración propia.*



Fuente: datos del proyecto

Gráfica 55 ***Ventana de seguridad de servicios.*** *Elaboración propia.*

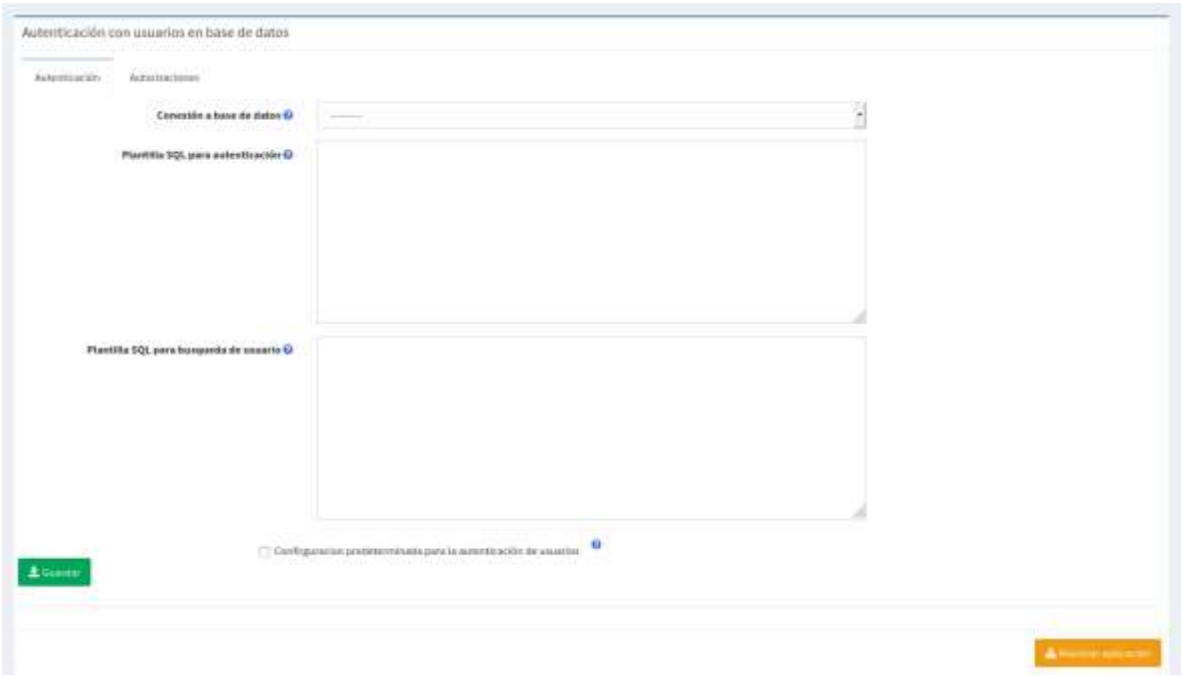


Fuente: datos del proyecto

1.2.6.3 Sistema de autenticación.

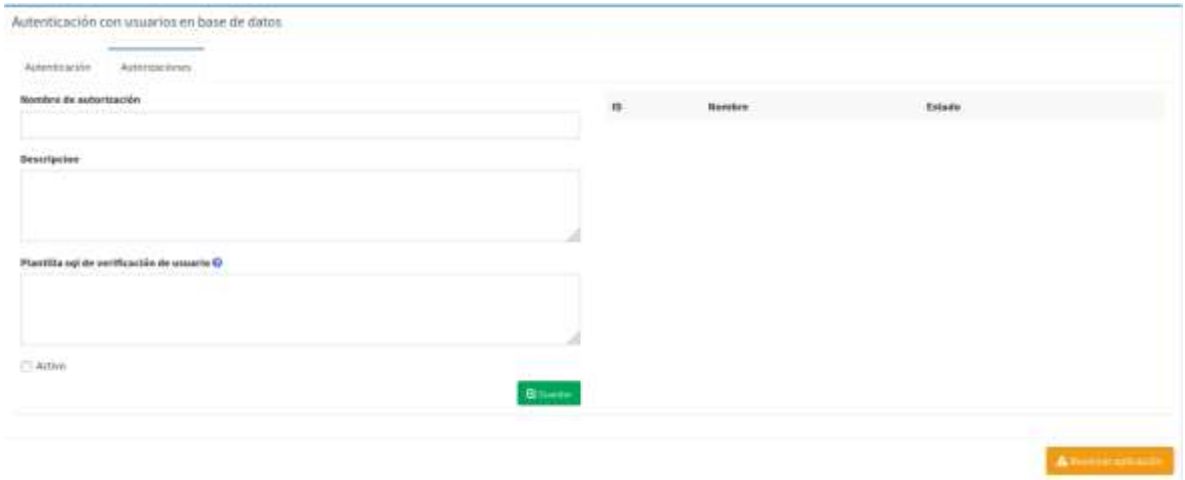
La aplicación ofrece la posibilidad de restringir el acceso a los servicios configurados, gestionando grupos de usuarios que tendrán acceso a estos servicios. Estos grupos pueden ser configurados a partir de las conexiones a bases de datos disponibles, escogiendo el modelo y los campos que serán usados para obtener y validar los campos ‘usuario’ y ‘contraseña’.

Gráfica 56 ***Ventana de parametrización de autenticación para usuarios en bases de datos*** *Elaboración propia.*



Fuente: datos del proyecto

Gráfica 57 ***Ventana de gestión de roles para usuario en bases de datos.*** *Elaboración propia.*

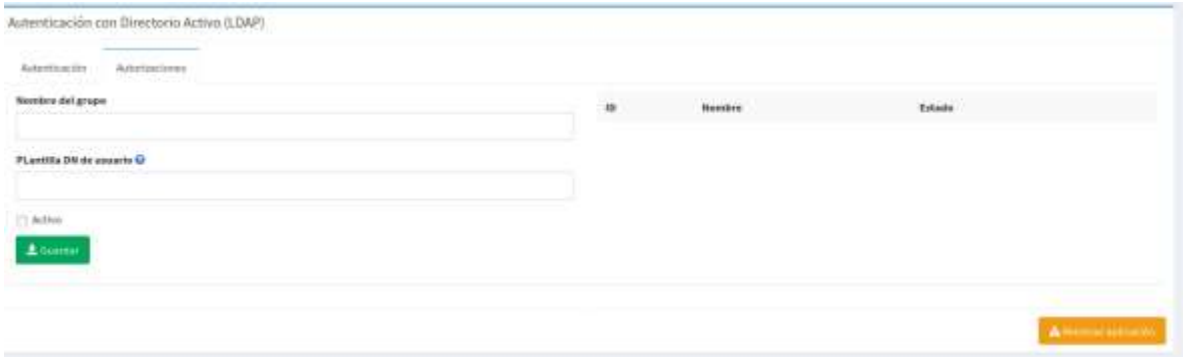


Fuente: datos del proyecto

Gráfica 58 ***Ventana de parametrización de autenticación para usuarios en directorio activo.*** *Elaboración propia.*

Fuente: datos del proyecto

Gráfica 59 ***Ventana de gestión de roles para usuario en directorio activo.*** *Elaboración propia.*

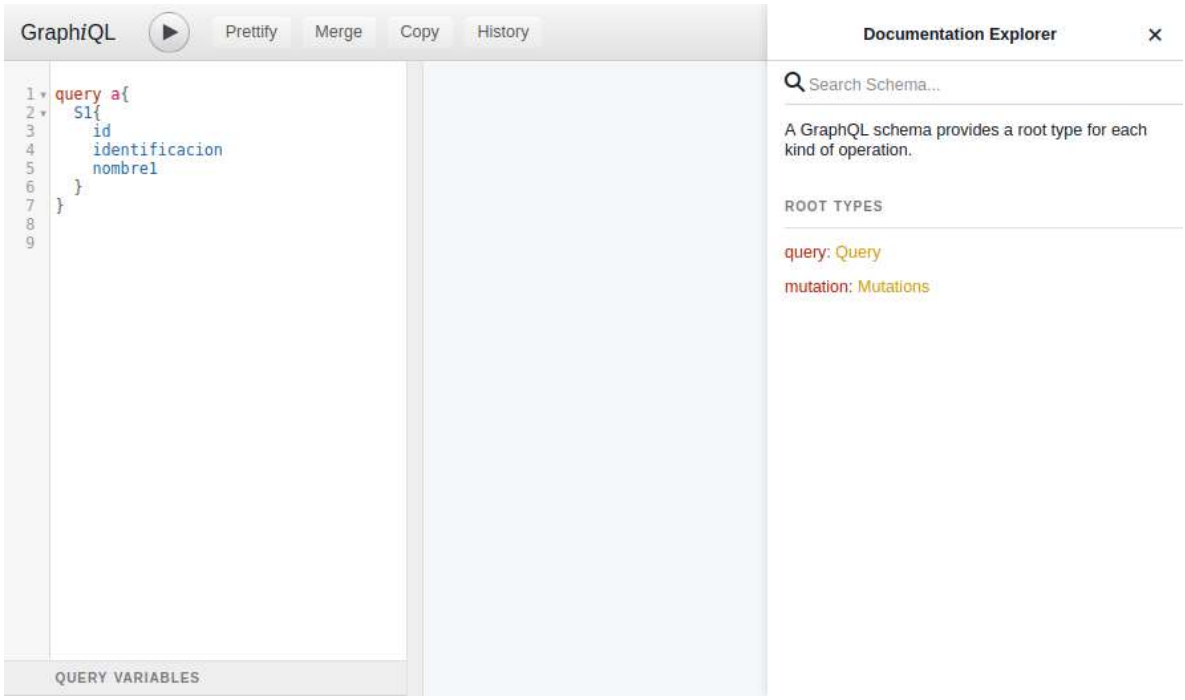


Fuente: datos del proyecto

1.2.6.4 Entorno de pruebas

Se ofrece un entorno de pruebas para consultas GraphQL llamado Graphil. Este entorno ofrece una documentación de la API, y es capaz de autocompletar y ofrecer sugerencias durante la construcción de queries para consultar la información disponible en los diferentes servicios creados por el usuario.

Gráfica 60 ***Entorno de pruebas GraphiQL.*** *Elaboración propia.*



Fuente: datos del proyecto

1.2.7 Síntesis

En este capítulo se hace un análisis documental comparando las arquitecturas de desarrollo de API: REST y GraphQL, a partir de este análisis, se determina la creación de una API GraphQL. Se hace mediante el desarrollo de un aplicativo Django, usando la metodología Extreme Programming. Este aplicativo permite esquematizar recursos de datos y posteriormente compartirlos para su consulta.

Estos datos pueden provenir de una o varias bases de datos que el usuario podrá administrar desde una interfaz. Una vez se gestionan las conexiones a la base de datos, el usuario tendrá la posibilidad de diseñar los servicios de la API partiendo de una consulta SQL, que será tomada como base para modelar y definir los tipos del esquema GraphQL de manera dinámica y de la misma forma obtener los datos para la resolución. En cuanto a seguridad el usuario podrá definir credenciales de acceso para cada uno de los servicios, por medio de un sistema de autenticación de usuarios que también será compartido como un servicio a través de GraphQL.

1.3 VALIDACIÓN DEL APLICATIVO

1.3.1 Elaboración de instrumentos de recolección de datos

Se realizó una encuesta dividida en dos secciones, la primera pretende recoger datos sobre los desarrolladores encuestados, de modo que se pueda caracterizar su nivel de conocimientos con respecto al área de la investigación. La segunda sección es una serie de preguntas que permitan establecer la opinión y el nivel de aceptación de la aplicación propuesta como aporte para el sistema de información de la Universidad de Nariño.

Para la primera sección se formularon dos preguntas de selección múltiple. La primera de ellas fue: ‘¿Como calificaría su conocimiento acerca de los siguientes conceptos?’. Para la cual se establecieron 5 posibles respuestas, nuevamente basadas en la escala Likert, estas opciones fueron: Muy bueno, Bueno, Regular, Malo y Muy malo. La segunda pregunta fue: ‘¿Con que frecuencia hace uso de las siguientes herramientas en su trabajo?’. Para esta pregunta también se estableció una escala Likert de 5 niveles, cuyas opciones fueron: Siempre, Casi siempre, Regularmente, Casi nunca y Nunca. Los ítems evaluados para ambas preguntas en esta sección fueron: API, REST y GraphQL.

En la segunda sección se formularon tres preguntas de selección múltiple con respuestas basadas en escala Likert. La primera pregunta de selección múltiple fue '¿Como calificaría el nivel de utilidad de la aplicación?', sus opciones de respuesta: Muy útil, Útil, Indiferente, Poco útil, Nada útil. La segunda pregunta de selección múltiple fue 'Cree que, con el uso de esta aplicación, el tiempo de desarrollo:', sus opciones de respuesta: Disminuiría mucho, Disminuiría un poco, No cambiaria, Aumentaría un poco, Aumentaría mucho. La última pregunta de selección múltiple fue 'Cree que, con el uso de esta aplicación, el proceso de intercambio de información:', sus opciones de respuesta: Mejoraría mucho, Mejoraría un poco, No cambiaria, Empeoraría un poco, Empeoraría mucho.

Finalmente, se estableció una pregunta de Si, No y Por qué fue: '¿Recomendaría utilizar esta aplicación en su trabajo?'. Siendo esta una pregunta más abierta a descripción y opinión por parte de los desarrolladores, a nivel cualitativo.

1.3.2 Recolección de los datos

La recolección de datos se aplicó de forma remota, mediante una presentación en línea, esto debido a las condiciones impuestas por la emergencia sanitaria generada por la pandemia del Covid-19. La presentación estuvo dirigida a los desarrolladores del centro de informática de la Universidad de Nariño, entre ellos el director del centro de informática, quien desde que tuvo conocimiento del proyecto mostro interés y apoyo hacia él. Las encuestas se aplicaron en dos etapas, la primera sección, orientada a la caracterización del nivel de conocimientos de los desarrolladores respecto a la temática de la investigación, se aplicó al inicio del encuentro, posteriormente se hizo una presentación del proyecto, incluyendo la aplicación resultado del proceso de desarrollo, con una explicación de su funcionalidad y posible utilidad para la Universidad. Finalmente, se aplicó la última sección de la encuesta, haciendo las preguntas preparadas que permitirían evaluar

el nivel de aceptación de la propuesta planteada, y su nivel de utilidad, en opinión de los propios desarrolladores.

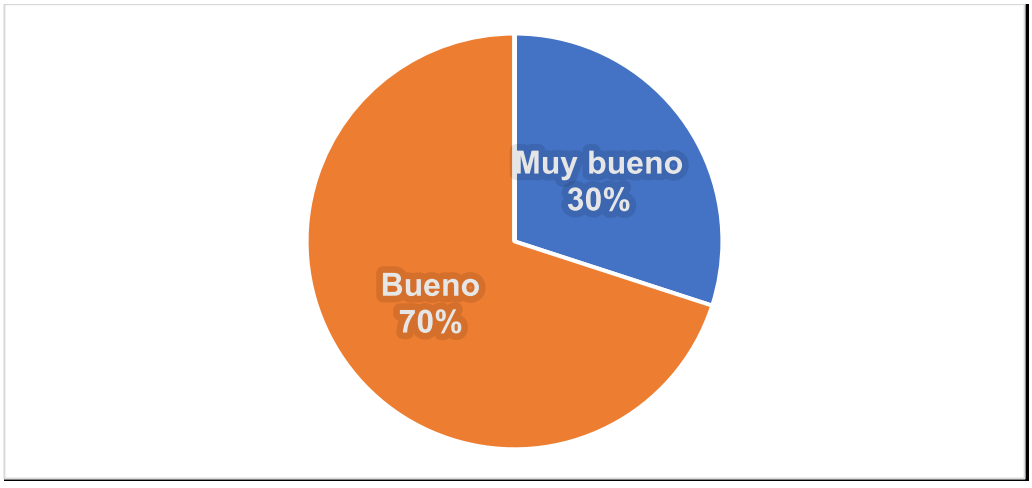
En total hubo 27 personas presentes, a quienes se les solicito responder las encuestas descritas anteriormente, una antes de la presentación y otra al final, sin embargo, de las 27 personas, solo 10 respondieron la primera parte de la encuesta, y 9 respondieron la segunda. La intención del proyecto era hacer una encuesta que arrojara datos en una escala mayor, sin embargo, teniendo en cuenta la limitada cantidad de encuestas respondidas, habría sido una mejor opción la aplicación de un instrumento de recolección de información diferente, como una entrevista.

1.3.3 Preparación de los datos para el análisis

Los datos se reportaron haciendo uso de gráficos circulares que permitan ver claramente el porcentaje de los desarrolladores que escogió cada opción.

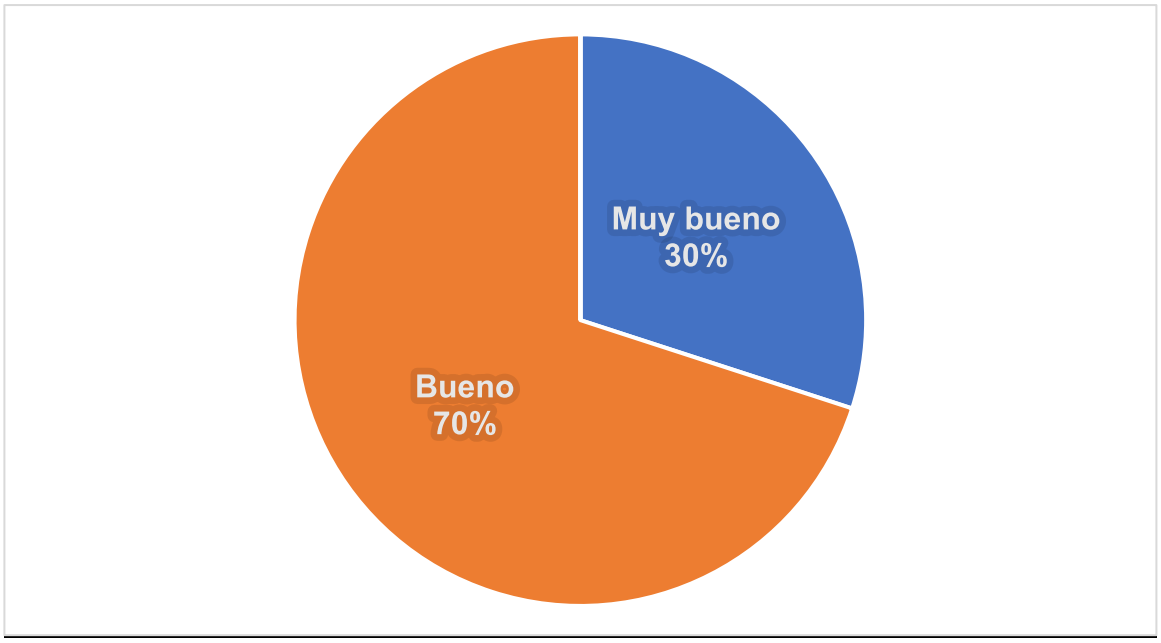
¿Como calificaría su conocimiento acerca de los siguientes conceptos?

Gráfica 61 ***Nivel de conocimiento de desarrolladores respecto al concepto de API.*** *Elaboración propia.*



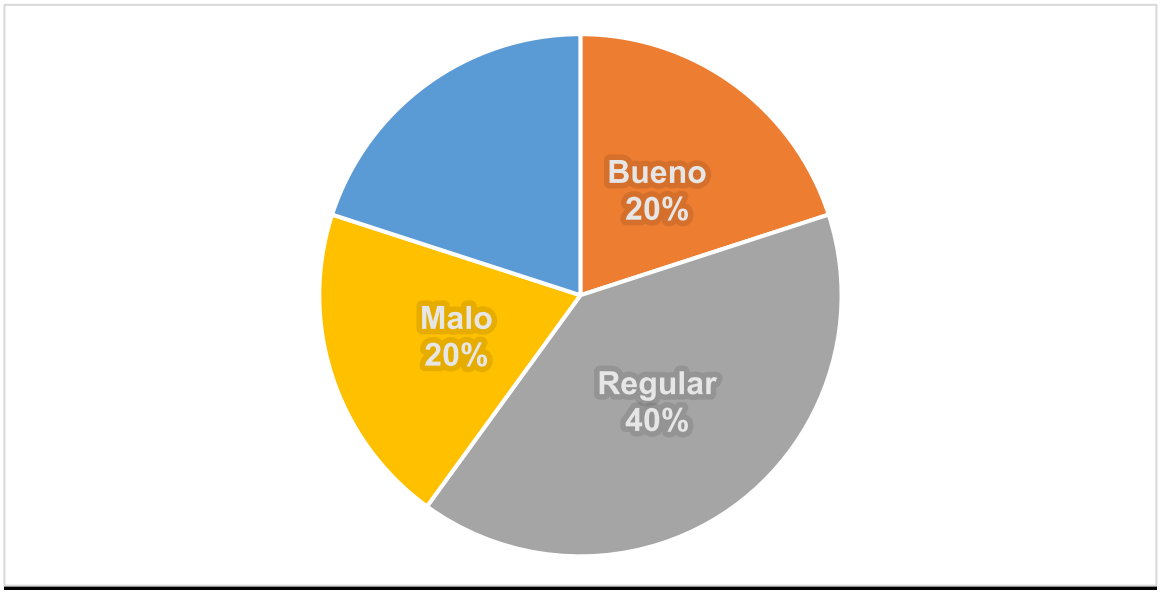
Fuente: datos del proyecto

Gráfica 62 ***Nivel de conocimiento de desarrolladores respecto al concepto de REST.*** *Elaboración propia.*



Fuente: datos del proyecto

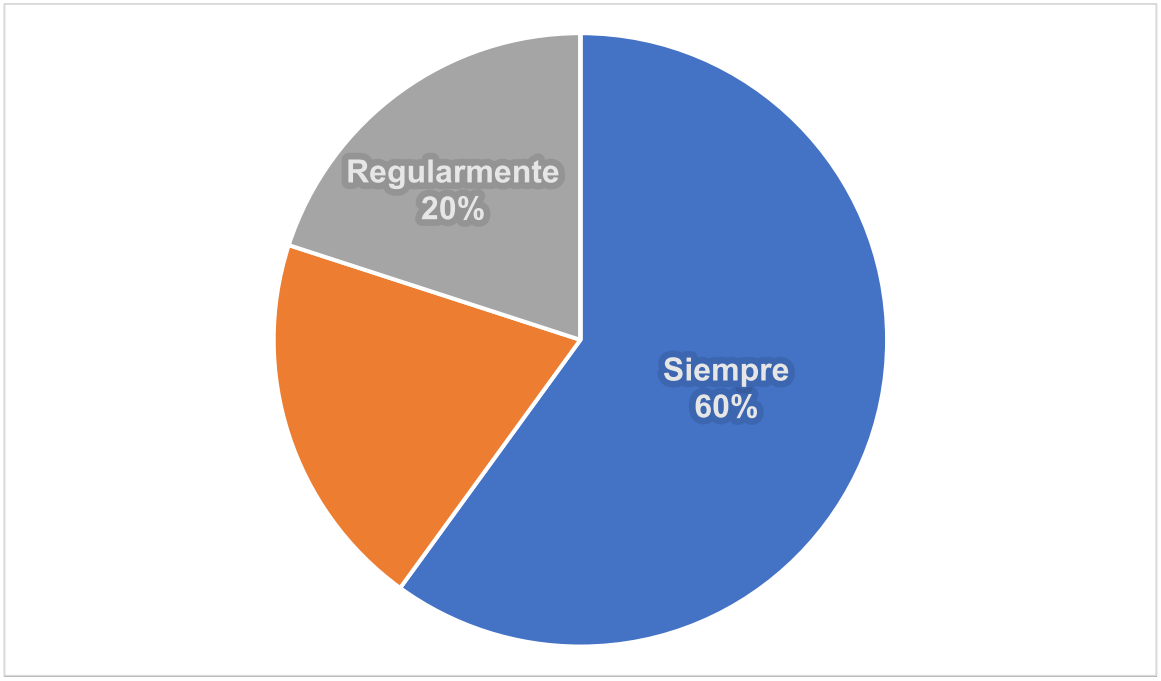
Gráfica 63 ***Nivel de conocimiento de desarrolladores respecto al concepto de GraphQL.*** *Elaboración propia.*



Fuente: datos del proyecto

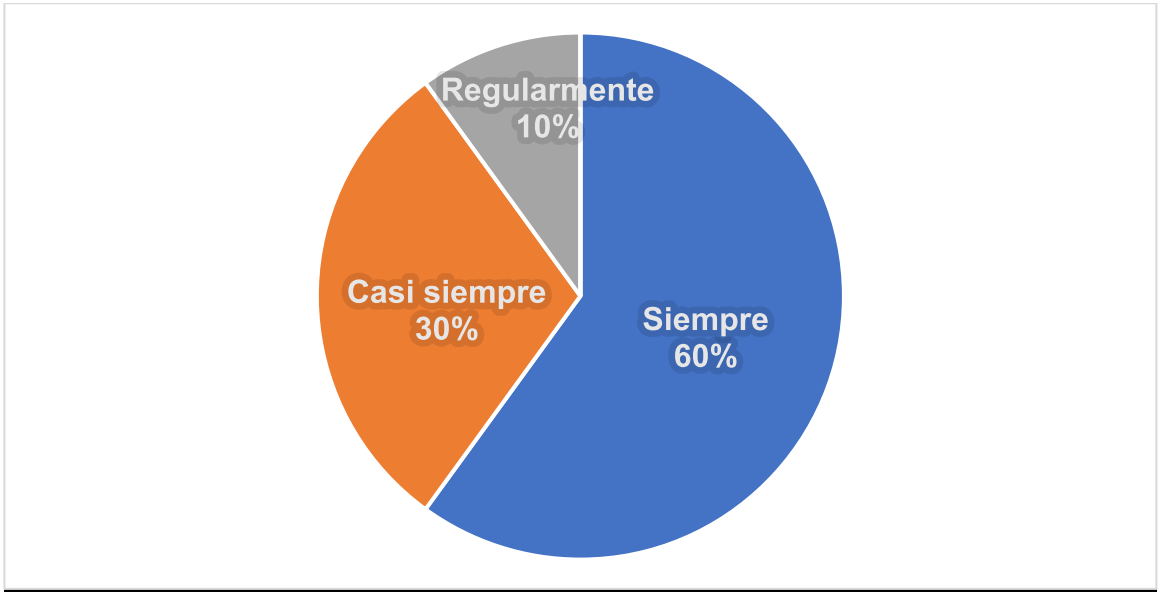
¿Con que frecuencia hace uso de las siguientes herramientas en su trabajo?

Gráfica 64 ***Frecuencia de uso por parte de desarrolladores de API.*** *Elaboración propia.*



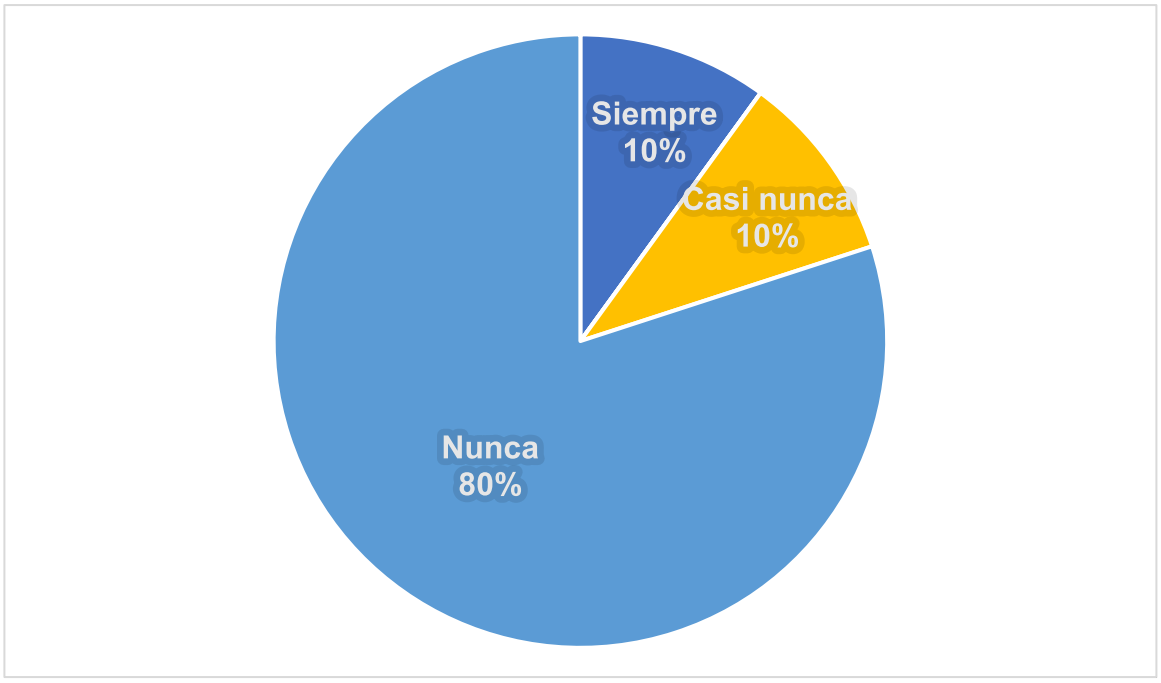
Fuente: datos del proyecto

Gráfica 65 ***Frecuencia de uso por parte de desarrolladores de REST.*** *Elaboración propia.*



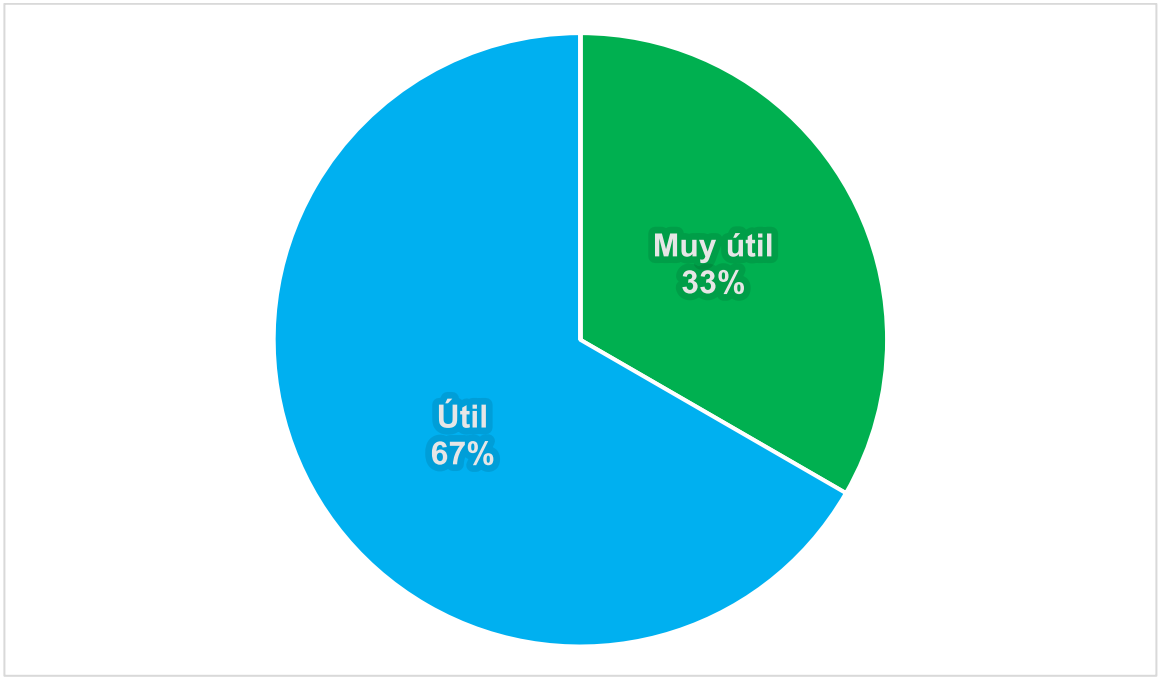
Fuente: datos del proyecto

Gráfica 66 ***Frecuencia de uso por parte de desarrolladores de GraphQL.*** *Elaboración propia.*



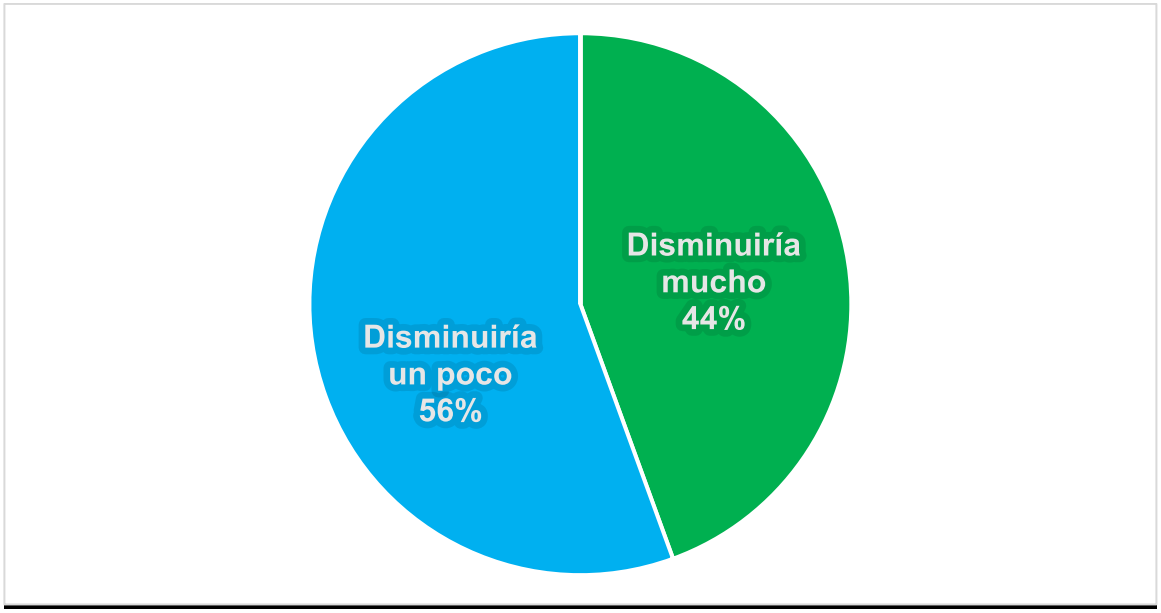
Fuente: datos del proyecto

Gráfica 67 ***Nivel de utilidad del aplicativo propuesto según desarrolladores.*** *Elaboración propia.*



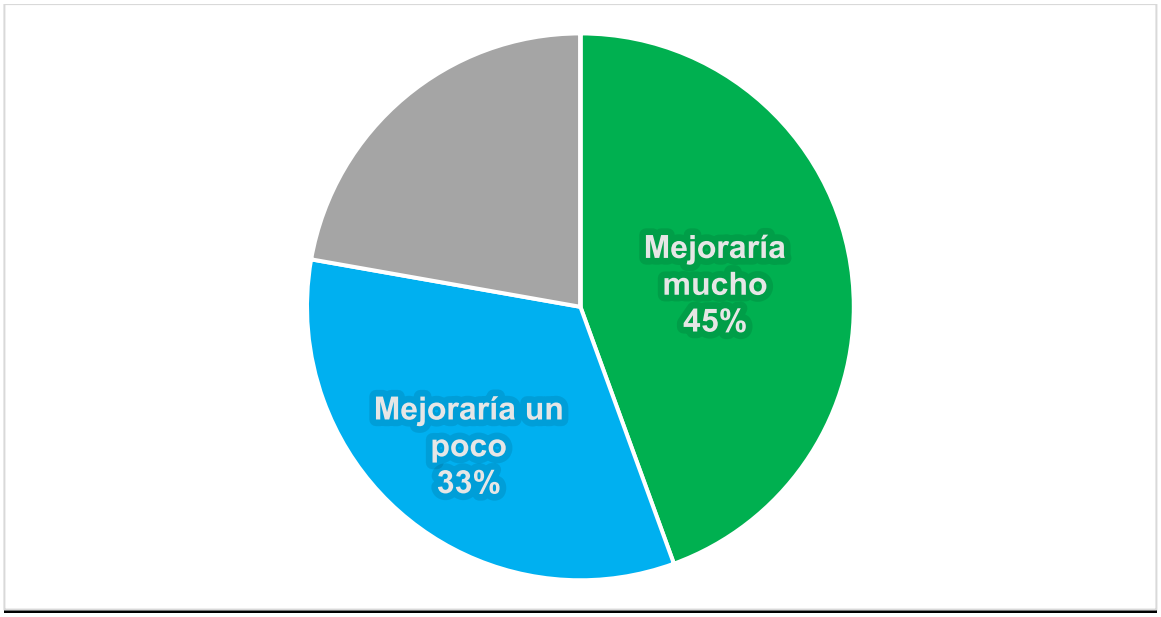
Fuente: datos del proyecto

Gráfica 68 ***Impacto en el tiempo de desarrollo del aplicativo propuesto según desarrolladores.*** *Elaboración propia*



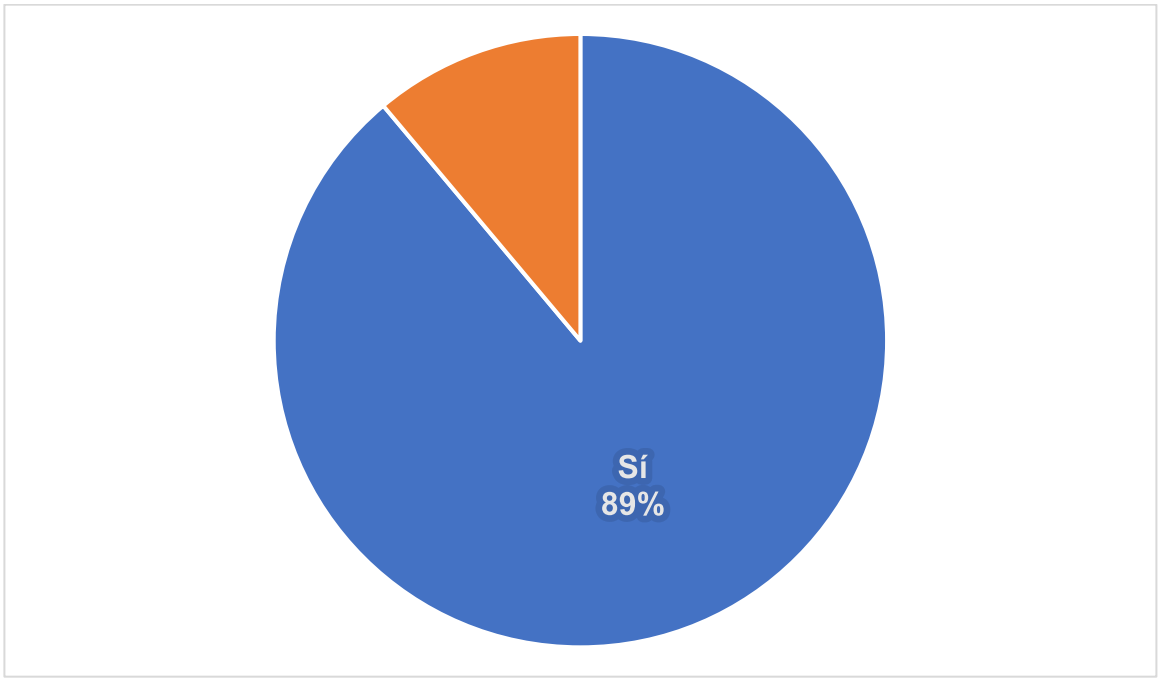
Fuente: datos del proyecto

Gráfica 69 ***Impacto en el proceso de intercambio de información del aplicativo propuesto según desarrolladores.*** *Elaboración propia.*



Fuente: datos del proyecto

Gráfica 70 ***Porcentaje de desarrolladores que recomiendan el uso del aplicativo propuesto.*** *Elaboración propia.*



Fuente: datos del proyecto

Razones:

- ahorro de tiempos en casos de consulta.
- Mejoraría el nivel de abstracción, mantenimiento y administración de accesos a la información
- LOS BENEFICIOS ANTERIORMENTE MENCIONADOS
- Agiliza el proceso de desarrollo y facilita la integración del backend estandarizando la respuesta del mismo
- Se puede ofrecer a los clientes una forma más directa, sencilla y eficiente para obtener exactamente los datos que requieren.
- Para hacer un acercamiento inicial de las bondades Básicas de usar GraphQL
- Sería bueno conocer e implementar nuevas tecnologías para que la forma de desarrollar software no se estanque, y si en un futuro se ve la necesidad de actualizar algunas cosas, se realice sin generar muchos traumas.
- Facilita el desarrollo de las aplicaciones
- Creo que es una herramienta que tendría mucha utilidad conforme su integración sea fácil para entornos de trabajo existentes. De lo contrario solo podría ser utilizada en entornos que se adapten a la metodología y estructura que genera el programa o en nuevos proyectos de desarrollo.

1.3.4 Análisis de los datos

Los resultados de la etapa de caracterización del nivel de conocimiento de los desarrolladores sobre la temática de la investigación arrojaron como resultado algo que ya era esperado, y es que GraphQL siendo una tecnología relativamente reciente aun no es tan ampliamente conocida en el grupo de desarrolladores de la Universidad de Nariño. Sin embargo, como se mencionó antes, durante el proceso de presentación de la propuesta, se entablo una conversación que dejó entrever un alto nivel de interés por parte de los desarrolladores por esta nueva herramienta y su potencial a futuro, además de las ventajas que podría otorgar de ser implementada en más procesos dentro del sistema de información de la Universidad.

En cuanto a la opinión sobre la investigación y la aplicación propuesta, esta fue en general muy positiva, tanto en los resultados de las encuestas, como en la interacción con algunos de los desarrolladores se notó un nivel de interés y aceptación muy alto. Entre las posibles ventajas o aportes de la aplicación para la Universidad se mencionaron la reducción del tiempo de desarrollo, el facilitar este proceso para los desarrolladores, y no solo en la creación de nuevas funcionalidades y servicios sino en el mantenimiento de las que ya existen.

Durante la presentación hubo mucha actividad a nivel de preguntas y propuestas, en la mayoría muy acertadas e interesantes. Se podría destacar una como un apunte curioso, pues se relacionaba con cual sería una forma de hacer que la API GraphQL propuesta funcionase, no como un solo endpoint al cual enviarle consultas personalizadas, sino como diferentes endpoints que devolvieran unos sets de data predefinidos de modo que se ahorrara el proceso de escritura de la consulta GraphQL. Este concepto encaja perfectamente con lo que una API RESTful ofrece, y no hay nada negativo al respecto, pero resulta interesante ver como a veces la

primera reacción al conocer herramientas nuevas, puede ser ver la forma en la que se puede hacerlas funcionar del mismo modo que aquello que ya se conoce.

1.3.5 Síntesis

En este capítulo se describe el proceso de validación del aplicativo propuesto por esta investigación. El proceso se realiza mediante la aplicación de encuestas a los desarrolladores del centro de informática de la Universidad de Nariño, después de hacer una presentación del aplicativo y sus funcionalidades. Los resultados del proceso de validación fueron muy positivos. De los desarrolladores del centro de informática que se pudo encuestar, el 89% tuvo una impresión positiva de la aplicación y considero que ofrece mucha utilidad en múltiples posibles escenarios. Se considera que puede ahorrar tiempo en el desarrollo de nuevas funcionalidades y servicios, además de facilitar el mantenimiento de los ya existentes, en especial los relacionados con entrega de información.

2 CONCLUSIONES

La frecuencia de uso de los servicios web ofrecidos por la Universidad de Nariño, además de la demanda por nuevos servicios que permitieron acceso a más información valiosa para la comunidad universitaria es alta. Muchos de estos servicios son indispensables para el desarrollo mismo de los objetivos misionales de la Universidad.

En el mundo interconectado de la actualidad, compartir y consumir servicios a través de internet fue la forma estándar de generar valor y soportar modelos de negocio, y proporcionar acceso a información y servicios a través de una API permite a las organizaciones ser competitivas. Por ello, la Universidad de Nariño debe permanecer a la vanguardia, fortaleciendo sus herramientas para mantener el flujo de información necesario para el adecuado funcionamiento y cumplimiento de sus objetivos misionales.

La caracterización de las necesidades de acceso a la información de los diferentes grupos que conforman la comunidad universitaria de la Universidad de Nariño, determinó que, aunque comparten interés en ciertos servicios comunes, fundamentalmente tienen requerimientos diferentes, asociados a su papel y funciones dentro de la Universidad.

Para organizaciones grandes, con multiplicidad de actores involucrados y con gran flujo de información entre ellos, como la Universidad de Nariño, fue fundamental tener un sistema de información flexible. Esto se debió, entre otras razones, a que tratar de dar solución a cada requisito como algo aislado, fue muy costoso, tanto en creación, pero sobre todo en mantenimiento. La implementación y el uso de las herramientas apropiadas pudieron ayudar a evadir o mitigar este costo, por ejemplo, aprovechando los beneficios de la arquitectura GraphQL o los conceptos de arquitectura de software.

El equipo de desarrolladores del centro de informática de la Universidad de Nariño, están familiarizados con el uso de API y en particular con la arquitectura REST, esto

debido a que el sistema de información de la Universidad está basado en ella. Las API basadas en la arquitectura REST fueron la base de la gran mayoría de los servicios web que se consumieron durante las últimas décadas, no solo en la Universidad, sino en el mundo, y aun hace parte de muchos de ellos.

El análisis documental determino que algunas de las características de REST no se adecuan a los requerimientos de aplicaciones grandes y complejas, con alto flujo de información, que son, cada vez más, la regla en el mundo moderno. Esto se debe, por ejemplo, a la necesidad de creación de múltiples endpoints y a los problemas de subfetching y overfetching que surgen a raíz de ello.

El equipo de desarrolladores del centro de informática de la Universidad de Nariño manifestaron interés entorno a la arquitectura de API GraphQL, que, al surgir como respuesta a los problemas encontrados con REST en el desarrollo de aplicaciones con alto nivel de complejidad como Facebook, provee ventajas claras en cuanto a flexibilidad y escalabilidad, y estos fueron requisitos indispensables para el desarrollo de este tipo de aplicaciones.

GraphQL es una herramienta favorable para la implementación de sistemas de información en organizaciones grandes y complejas como la Universidad de Nariño, esto gracias a sus características, entre las que destacan la existencia de un endpoint único, la posibilidad de obtener los datos que se requirieron en una sola petición, además de poder añadir nuevos objetos y campos fácilmente, sin interferir con el funcionamiento de la API y generando automáticamente documentación de uso.

3 RECOMENDACIONES Y TRABAJOS FUTUROS

Para emular la etapa de caracterización de necesidades de información en otras organizaciones, se sugiere modificar las preguntas planteadas en esta investigación, removiendo el componente ‘móvil’ de ellas. Se debería centrar la investigación inicialmente en las necesidades de acceso a la información de forma general, y una vez cubierto este aspecto, ya se podría analizar el nivel de interés de una aplicación móvil y que servicios concretos pueden ser preferibles en este tipo de aplicación.

Además, una pregunta complementaria a la información obtenida podría ser: ‘¿Como calificaría el nivel de utilidad de los siguientes servicios web institucionales?’ Con las opciones de respuesta: Muy importante, Importante, Relativamente importante, Poco importante, Nada importante. Esta pregunta permitiría la comparación entre la frecuencia de uso y el nivel de importancia, de modo que se pueda establecer causalidades, o establecer incongruencias que inviten a encontrar las razones por las cuales un servicio catalogado con alto nivel de importancia tiene baja frecuencia de uso.

El aplicativo propuesto exploro y aprovecho la funcionalidad del tipo ‘query’ de la arquitectura GraphQL, que permite la consulta de información en base a un esquema. Sin embargo, esta arquitectura también ofrece otras vías para interactuar con los datos a partir de ese esquema, y una de ellas es la proporcionada por el tipo ‘mutation’, que permite crear y modificar datos modelados a partir del esquema. Explorar esta área le abriría al aplicativo propuesto un sinfín de nuevas posibilidades en cuanto a los servicios que puede ofrecer, pues no se limitaría a ofrecer información, sino a interactuar activamente con esta.

Por otra parte, dado que el aplicativo desarrollado en esta investigación es, en pocas palabras, un gestor de información que permite organizarla en forma de servicios y presentarlos en una API GraphQL para ser consumidos, y pretende ser un punto de partida para el desarrollo de aplicaciones que requieran consumir información, seria de enorme utilidad la creación de estas aplicaciones haciendo uso del gestor de información provisto, y analizar su utilidad en escenarios de uso reales. Cabe señalar que la información recolectada por esta investigación también proporciona una idea de que servicios son requeridos por la comunidad universitaria de la Universidad de Nariño.

Finalmente, si se hace uso de la aplicación propuesta para la construcción de otras aplicaciones que consuman la API GraphQL provista, hacer una comparativa en este caso de uso específico entre REST y GraphQL sería un buen aporte a la no tan abundante literatura académica entorno a las diferencias en rendimiento y otros aspectos que hay al comparar las dos arquitecturas. Esta comparativa sería posible dado que la universidad ya cuenta con servicios REST, que pueden ser comparados con un equivalente GraphQL configurado en la aplicación de la presente investigación, que ya se encuentra instalada en los servidores de la Universidad de Nariño.

BIBLIOGRAFÍA

AEPI. (s.f.). Mini tutorial sobre SOAP. Simple Object Access Protocol: AEPI. Obtenido de AEPI: https://asociacionaepi.es/mini-tutorial-sobre-soap-simple-object-access-protocol/

Agile Alliance. (s.f.). Agile Alliance - Glossary. Obtenido de https://www.agilealliance.org/glossary/xp/

Álvarez, C. (2 de enero de 2019). Arquitecturas RESTFul y agregados: Arquitectura Java. Obtenido de Arquitectura Java: https://www.arquitecturajava.com/arquitecturas-restful-y-agregados/

Aplyca. (s.f.). GraphQL - Evolución del diseño de API's: Aplyca. Obtenido de Aplyca: https://www.aplyca.com/es/blog/graphql

Bajarin, T. (26 de junio de 2017). How Apple's iPhone Changed These 5 Major Industries: TIME. Obtenido de TIME: https://time.com/4832599/iphone-anniversary-industry-change/

BBC Mundo. (14 de diciembre de 2017). Las 10 empresas más valiosas del mundo: BBC. Obtenido de BBC: https://www.bbc.com/mundo/noticias-42327754

BBVA. (s.f.). Introducción al mundo de las APIs: BBVA. Obtenido de BBVA Open for you: https://bbvaopen4u.com/sites/default/files/ebook/bbva-open4u-ebook-101-apis-espok.pdf

BBVA. (23 de marzo de 2016). API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos: BBVA. Obtenido de BBVA: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

Beyang. (27 de octubre de 2017). GraphQL at Twitter: Sourcegraph. Obtenido de Sourcegraph: https://about.sourcegraph.com/graphql/graphql-at-twitter

Burk, N. (14 de noviembre de 2017). GraphQL Server Basics: GraphQL Schemas, TypeDefs & Resolvers Explained: Prisma. Obtenido de Prisma: https://www.prisma.io/blog/graphql-server-basics-the-schema-ac5e2950214e

Burk, N. (6 de febrero de 2018). GraphQL Server Basics: Demystifying the info Argument in GraphQL Resolvers: Prisma. Obtenido de Prisma: https://www.prisma.io/blog/graphql-server-basics-demystifying-the-info-argument-in-graphql-resolvers-6f26249f613a

Burk, N. (20 de marzo de 2018). Top 5 Reasons to Use GraphQL: Prisma. Obtenido de Prisma: https://www.prisma.io/blog/top-5-reasons-to-use-graphql-b60cfa683511

Cascio, W. F., & Montealegre, R. (2016). How Technology Is Changing. Annu. Rev. Organ. Psychol. Organ. Behav., 3:349–75.

Chai, R. (9 de abril de 2019). Why The World Needs The Internet Of Trusted Things: International Bussines Times. Obtenido de International Bussines Times: https://www.ibtimes.com/why-world-needs-internet-trusted-things-2821883

Chakray. (s.f.). ¿QUÉ DIFERENCIAS HAY ENTRE REST Y SOAP?: Chakray. Obtenido de Chakray: https://www.chakray.com/es/que-diferencias-hay-entre-rest-y-soap/

Cómo funcionan las metaclases en Python. (s.f.). Obtenido de Programacion.net: https://programacion.net/articulo/como_funcionan_las_metaclases_en_python_1517

Desjardins, J. (29 de marzo de 2019). How the Tech Giants Make Their Billions: Visual Capitalist. Obtenido de Visual Capitalist: https://www.visualcapitalist.com/how-tech-giants-make-billions/

Django. (s.f.). Django: The web framework for perfectionists with deadlines. Obtenido de Django: https://www.djangoproject.com/

EcuRed. (s.f.). Protocolo simple de acceso a objetos (SOAP): EcuRed. Obtenido de EcuRed: https://www.ecured.cu/Protocolo_simple_de_acceso_a_objetos_(SOAP)

Finley, K. (2 de abril de 2019). How Facebook Has Changed Computing: WIRED. Obtenido de WIRED: https://www.wired.com/story/how-facebook-has-changed-computing/

García Peñalvo, F. J. (2011). La Universidad de la próxima década: La Universidad Digital.

Garcia Rozas, J. M. (31 de Julio de 2017). GraphQL: ¡todos para uno y uno para todos!: Paradigma. Obtenido de Paradigma: https://www.paradigmadigital.com/dev/graphql-todos-uno-uno-todos/

Graphene Python. (s.f.). Graphene Python. Obtenido de Graphene Python: https://graphene-python.org/

GraphQL Specification Versions. (2 de Julio de 2015). Obtenido de https://spec.graphql.org/

Guillen-Drija, C., Quintero, R., & Kleiman, A. (2018). GraphQL vs REST: una comparación desde la perspectiva de eficiencia de desempeño.

Helfer, J. (23 de mayo de 2016). GraphQL explained: Apollo Blog. Obtenido de Apollo Blog: https://www.apollographql.com/blog/graphql-explained-5844742f195e/

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, M. (2010). Metodología De La Investigación (Quinta Edición ed.). (J. M. Chacón, Ed., & S. Interamericana Editores, Trad.) Mexico D.F, Mexico: Mc Graw Hill. Recuperado el 20 de enero de 2017

Hernandez, U. (1 de mayo de 2018). Qué Es La Metaprogramacion: Codigo facilito. Obtenido de Codigo facilito: https://codigofacilito.com/articulos/que-es-metaprogramacion

Herrera, C., Pacheco, P., & Suazo, J. (s.f.). PARADIGMAS CUANTITATIVO Y CUALITATIVO Y METODOLOGÍA DE LA INVESTIGACIÓN.

Hoff, J. (30 de marzo de 2015). Engaging Students with a Mobile App: EducauseReview. Obtenido de EducauseReview: https://er.educause.edu/articles/2015/3/engaging-students-with-a-mobile-app

IBM. (s.f.). ¿Qué es WSDL?: IBM. Obtenido de IBM: https://www.ibm.com/support/knowledgecenter/es/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ac34640_.htm

IBM. (s.f.). The structure of a SOAP message: IBM. Obtenido de IBM: https://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/ac55780_.htm

Investigación aplicada: qué es, características y ejemplos. (s.f.). Obtenido de Tipos de Investigacion: https://tiposdeinvestigacion.org/aplicada/

Ionos. (15 de abril de 2020). SOAP: explicación del protocolo de red: Ionos. Obtenido de Ionos: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/soap-simple-object-access-protocol/

Ionos. (15 de abril de 2020). Tutorial de WSDL, el lenguaje de descripción de servicios web: Ionos. Obtenido de Ionos: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/wsdl/

Jensen, C. T. (2015). API for Dummies. John Wiley & Sons, Inc.

Laurence, D., & Mancl, D. (19 de diciembre de 2002). Princeton ACM. Obtenido de http://princetonacm.acm.org/downloads/extreme_agile.html

Mano. (20 de febrero de 2019). Graphql, Entenderlo y Usarlo En Producción: Mano. Obtenido de Mano: https://www.ma-no.org/es/redes/bases-de-datos/graphql-entenderlo-y-usarlo-en-produccion

Marqués, A. (11 de abril de 2013). Conceptos sobre APIs REST: Asier Marques. Obtenido de Asier Marques: https://asiermarques.com/2013/conceptos-sobre-apis-rest/

Martin, R. C. (2016). Clean Architecture: A Craftsman´s Guide To Software And Design. Prentice Hall.

Maslow, A. H. (1970). The Psychology of Science: A Reconnaissance. Chicago Gateway.

Matas, A. (2018). Diseño del formato de escalas tipo Likert: un estado de la cuestión. Revista Electrónica de Investigación Educativa, Colección Trópicos, 20(1), 38-47. doi: https://doi.org/10.24320/redie.2018.20.1.1347

McGarvie, J., & Tracey, E. (Dirección). (2020). Graphql: The Documentary [Película]. Obtenido de https://cult.honeypot.io/originals/graphql-the-documentary

Meeker, M. (junio de 2019). Internet Trends 2019: Bond. Obtenido de Bond: https://www.bondcap.com/report/itr19/#view/1

Merino, M. (12 de Julio de 2014). ¿Qué es una API y para qué sirve?: TIC Beat. Obtenido de TIC Beat: https://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/

Miceli, S. (2019). La universidad como organizacion. En S. Miceli, El sistema de aseguramiento de la calidad (pág. 4). Teseo.

Muñoz, A., & Cabrera, J. (2017). ARQUITECTURA ORIENTADA A SERVICIOS PARA CLIENTES MÓVILES. San Juan de Pasto.

Murray, A. (2015). The new industrial revolution. Fortune, 6.

Myers, B. A., & Stylos, J. (2016). Improving API Usability. Communications of the ACM, 62-69.

Ordoñez, J. (s.f.). ¿Qué es una API REST?: Idento. Obtenido de Idento: https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/

Peña Gómez, C. F., & Ramírez Salamanca, D. F. (2016). NUBI: Navega, Ubícate, Busca, Interactúa. Bogotá D.C.: Pontificia Universidad Javeriana.

Prisma. (s.f.). The Fullstack Tutorial for GraphQL: How To Graphql. Obtenido de How To Graphql: https://www.howtographql.com/

Python. (s.f.). Python. Obtenido de Python: https://www.python.org/

Red Hat. (s.f.). Qué son las API y para qué sirven: Red Hat. Obtenido de Red Hat: https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

Reyes Rodriguez, E. M. (2015). ANÁLISIS Y DISEÑO DE UNA APLICACIÓN MÓVIL PARA EL ACCESO DE LA INFORMACIÓN ACADÉMICA EN TIEMPO REAL, DEL I.S.T. IDAT. Lima: Universidad Tecnológica del Perú.

Robledano, Á. (23 de septiembre de 2019). Qué es Python: Características, evolución y futuro: Open Webinars. Obtenido de Open Webinars: https://openwebinars.net/blog/que-es-python/

Sandy. (5 de febrero de 2014). Conceptos básicos de Servicios Web SOAP, WSDL y XSD: Desarrollo con SOA. Obtenido de Desarrollo con SOA: http://desarrolloconsoa.blogspot.com/2014/02/conceptos-basicos-de-servicios-web-soap.html

Sturtz, J. (s.f.). Python Metaclasses: Real Python. Obtenido de Real Python: https://realpython.com/python-metaclasses/

Thotapalli, K. V. (18 de junio de 2019). Understanding WSDL: DZone. Obtenido de DZone: https://dzone.com/articles/understanding-wsdl

Universidad de Nariño. (2018). Udenar en cifras: Anuario 2018. Universidad de Nariño.

Vazquez Ingelmo, A., Cruz Benito, J., & Garcia Peñalvo, F. (2017). Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL. Cádiz.

Vogel, M., Weber, S., & Zirpins, C. (2018). Experiences on Migrating RESTful Web Services to GraphQL.

Williams, M. (5 de junio de 2019). REST VS GraphQL - Which Should be Used to Develop a Web API and Why?: PromptBytes. Obtenido de PromptBytes: https://www.promptbytes.com/blog/rest-vs-graphql-web-api-development