Universidad de las Ciencias Informáticas
Facultad 3

Componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores:

Dayanis Pérez Pérez

Migyara María González Labori

Tutores:

MSc. Yordanis García Leiva

Ing. Luis Javier Rodríguez Castro

Ing. Rafael Mayor Alberto

La Habana, 2019

“Año 61 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los _____ días del mes de _____ del año _____.

_____ _____

Dayanis Pérez Pérez Migyara María González Laborí

_____ _____

MSc. Yordanis García Leiva Ing. Luis Javier Rodríguez Castro

_____

Ing. Rafael Mayor Alberto

AGRADECIMIENTOS

De Dayanis:

Le agradezco a todos los que hoy están conmigo y a los que han estado, a los que cuando le pedía ayuda lo daban todo para ayudarme, y a los que fueron capaces de apreciarme sin haberme perdido antes. A la familia, sobre todo a mi mamá y a mi papá por apoyarme en este camino tan duro y por no dejarme caer.

Le agradezco a las muchachitas, las conversaciones de madrugada, las tardes en buena compañía, los momentos eternos, los detalles.

Agradezco a todos mis amigos que han pasado a ser hermanos, los que derrochan buena vibra y consiguieron contagiarla. Le agradezco a todo, a lo bueno a lo malo que pasamos juntos, porque de todo se aprende, y por las experiencias.

Te agradezco a ti migyi, y a mis tutores por todos esos momentos que pasamos, y ahora pasan a ser recuerdos inolvidables, a esas fechas que nunca se olvidan y a las sonrisas que tienen nombre y apellidos GRACIAS.

De Migyara:

Primero que todo me encantaría agradecer a mis adorables tutores, esas tres personas magnificas que tuve el placer de conocer este año, que son los principales responsables que me esté convirtiendo en una ingeniera, gracias principalmente por la confianza entregada.

Agradezco a todos los profesores que han contribuido a mi formación como ingeniera, especialmente a la profesora Dariela, la madre de todos nosotros, gracias por sacarnos de todos los apuros.

A mis súper chicos (Yoe, Orel, Ali, Jose Carlos, Sergio, el Pablix y Marco), esos amigos que nos visitaban noches tras noches, que nos extrañaban cuando no nos veían, espero de todo corazón algún día volverlos a ver.

A las niñas (Linda, Yayi, Dayi y Gaby), esas compañeras de vida, más que de cuarto, gracias por las largas horas hablando siempre de nuestros 3 temas favoritos, esas madrugadas de estudio intenso socializando el conocimiento, gracias por todo.

A mis colegas, mis amigas, esas que jamás olvidare, simplemente porque llegaron para cambiarme y apoyarme en todo, gracias más que nada por ser amigas.

Gracias Yannia por ser esa amiga que te sigue en todas las locuras, que dice no importa vamos a hacerlo, por muy loca que sea la propuesta, te voy a extrañar mi gaga bella.

A mi hermana, no de sangre, pero sí de alma y espíritu, esa enana, mi loquita favorita, eres unas de las mejores cosas que me paso en la universidad, gracias Tahimi Gonzalez por existir.

Mi flaca, mi flacucha bella, mi hermana, te doy gracias por ser una de mis principales motores de inspiración, en estos momentos te exhorto para que luches por tus sueños y veas que nada es imposible.

Mi pipo lindo, ese hombre que amo con todas mis fuerzas, mi guía, mi ejemplo a seguir, gracias por todo tu apoyo y esfuerzo, por darme aliento siempre, gracias papa, te amo.

A mi madre hermosa, y perfecta, mi gran amiga, mi razón de vivir, gracias por darme la vida, por confiar cada instante en mí, y por no dejarme caer nunca, eres mi persona favorita y te amo más que a mi propia vida.

DEDICATORIA

De Dayanis:

Esta tesis va dedicada a mis padres, por todo su esfuerzo y cariño entregado.

De Migyara:

Dedico este trabajo de diploma a la persona más importante en mi vida, mi gran tesoro, mi ejemplo a seguir, mi madre, y a mi hermana preciosa que amo con todas mis fuerzas.

RESUMEN

En la actualidad el desarrollo de soluciones informáticas se caracteriza por el uso de arquitecturas que permiten el consumo de servicios web. Entre las tecnologías que brindan la posibilidad de desarrollar este tipo de soluciones se encuentran transferencia de estado representacional y GraphQL. Estas permiten consultar recursos desde clientes web a través de servicios. Sin embargo, GraphQL se define como un lenguaje de consulta que supera a la transferencia de estado representacional, cuando a través de esta última se requiere de varias peticiones para consultar más de un recurso, mediante el uso de GraphQL solo se necesita una única petición para obtener los mismos datos.

La presente investigación tiene como objetivo desarrollar un componente, organizado en una interfaz de programación de aplicaciones y un cliente GraphQL para la administración de usuarios, roles, permisos y trazas en una aplicación web, que reduzca las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor. El proceso de desarrollo está guiado por el uso de la metodología Proceso Unificado Ágil variación UCI. En la implementación del componente se utilizan tecnologías de código abierto. El resultado de la investigación fue validado utilizando una estrategia de pruebas en los niveles de unidad y aceptación. Además, se establecen un conjunto de criterios que permiten evaluar a través de un antes y un después la relación causa-efecto de las variables de la investigación.

PALABRAS CLAVES: componente**,** GraphQL, transferencia de estado representacional

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad son muchas las esferas de la sociedad a las cuales van dirigidos los desarrollos de sistemas informáticos. En cada desarrollo el uso de las arquitecturas puede ser variado. A pesar de la variabilidad tecnológica, la mayoría de los software presentan requerimientos comunes, tales como la necesidad de administrar usuarios, roles, permisos y trazas con el propósito de contribuir a su seguridad.

En Cuba una de las entidades que fomenta el avance de la industria del software lo constituye la Universidad de las Ciencias Informáticas, en la cual radica el Centro de Gobierno Electrónico (CEGEL), que tiene como misión satisfacer necesidades de clientes gubernamentales mediante el desarrollo de productos, servicios y soluciones integrales de alta confiabilidad, calidad, competitividad, fidelidad y eficiencia, a partir de un personal altamente calificado. Las soluciones desarrolladas en este centro forman parte de un sistema que responde a las necesidades y la gobernanza de las instituciones jurídicas y de gobierno para las cuales están destinadas.

CEGEL ha desarrollado soluciones informáticas para los distintos órganos de gobiernos nacionales e internacionales, tales como el Tribunal Supremo Popular, la Fiscalía General de la República de Cuba y la Asamblea Nacional del Poder Popular (ANPP), así como la División de Antecedentes Penales y el Servicio Autónomo de Registros y Notarías de la República Bolivariana de Venezuela. En el caso de la ANPP se desarrolló el Sistema de Gestión para la Atención a la Población (SIGAP), utilizando una arquitectura micro-servicios basada en (*REST Representational State Transfer*)[1], quedando identificado la gestión de usuarios, roles, permisos y trazas como un servicio.

Una arquitectura de micro-servicios es una aproximación para el desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros. Cada servicio se encarga de implementar una funcionalidad completa del negocio y es desplegado de forma independiente. Además puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos. (Victor S, 2018)

Las arquitecturas basadas en REST constituyen un nuevo enfoque de desarrollo de proyectos y servicios web. Sus principales características son: (Victor S, 2018)

[1] REST: transferencia de representación de estado, es un estilo arquitectónico para proporcionar estándares entre los sistemas informáticos en la web, lo que facilita la comunicación entre los sistemas. (Briging the semantic web and web2.0 winth representational state transfer (REST), 2008)

- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación Protocolo de Transferencia de Hipertexto (del inglés Hypertext Transfer Protocol) HTTP son cuatro: *POST* (crear), *GET* (leer y consultar), *PUT* (editar) y *DELETE* (eliminar).

- Los objetos en REST siempre se manipulan a partir del Identificador de Recursos Uniformes (*URI uniform resource identifie*). Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI facilita acceder a la información para su modificación o borrado.

- Sistema de capas: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.

El uso de arquitecturas REST tiene ventajas como (Victor S, 2018):

- Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos.

- La API REST es independiente del tipo de plataformas o lenguajes: la API REST se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado.

A pesar de las características y ventajas que brinda una arquitectura micro-servicio basada en REST, estas aun no resuelven los siguientes problemas:

- Obtener grandes conjuntos de datos que necesitan de recursos relacionados a través de API REST, requiere de múltiples llamadas (paralelas o secuenciales para obtener toda la información que se necesita). Esto implica que el servidor requiera de mayor cantidad de recursos para atender las peticiones y aumenten los tiempos de respuesta.

- El uso de REST puede brindar datos innecesarios durante las repuestas del servidor a las peticiones del cliente. Esto tiene como consecuencia que se consuma mayor cantidad de ancho de banda y una carga más lenta de los datos.

A partir de la problemática antes descrita se genera la necesidad de resolver el siguiente **problema de investigación**:

¿Cómo administrar usuarios, roles, permisos y trazas, reduciendo las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor?

Tomando en cuenta el problema antes propuesto se define como **objeto de estudio:** la administración de usuarios, roles, permisos y trazas.

Para ello se identifica como **campo de acción:** componente para la administración de usuarios, roles, permisos y trazas basado en GraphQL.

Determinándose como **objetivo general:** desarrollar un componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas que reduzca las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor.

Se desglosan del objetivo general los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación mediante un estudio de los referentes teóricos relacionados con las tecnologías REST y GraphQL.
- Desarrollar las disciplinas de identificación de requisitos, análisis, diseño e implementación del componente para la administración de usuarios, roles, permisos y trazas basado en GraphQL.
- Validar el diseño y el funcionamiento del componente propuesto aplicando métricas y pruebas de software respectivamente.
- Verificar el cumplimiento de la relación causa efecto de la variable independiente sobre las variables dependientes de la investigación.

Definiéndose como **idea a defender:** el desarrollo de un componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas reducirá las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor.

Métodos teóricos:

Histórico-Lógico: se empleó en el análisis de las características y evolución del lenguaje de manipulación GraphQL con el propósito de ser aplicado en el desarrollo del componente propuesto.

Analítico-Sintético: posibilitó la realización del estudio teórico de la investigación haciendo más sencillo el proceso de análisis de los documentos y la extracción de los elementos fundamentales a tener en cuenta para desarrollar el componente.

Modelación: se empleó para la confección de los prototipos funcionales, la representación de los requisitos del componente y el diseño de la base de datos.

Métodos empíricos:

Entrevista: se aplicó en el proceso de capturar de los requisitos correspondientes al componente propuesto.

El contenido de este trabajo consta de tres capítulos, definidos de la siguiente forma:

Capítulo 1: Fundamentación teórica. En este capítulo se describen una serie de conceptos relacionados con el objeto de estudio de la presente investigación, así como las características fundamentales de las tecnologías REST y GraphQL con el propósito de demostrar porque se decide emplear esta última en el desarrollo de la tesis. Por otra parte, se describe la metodología utilizada para guiar el proceso de desarrollo de la investigación, junto a los lenguajes y herramientas que se emplean. El capítulo concluye con una descripción de los elementos teóricos que caracterizan los patrones de diseño y arquitectónicos aplicados en la solución propuesta, así como las pruebas de software ejecutadas.

Capítulo 2: Análisis, diseño e implementación de la solución propuesta. En este capítulo se realiza una descripción del proceso de desarrollo del componente propuesto, partiendo del estudio del estado del arte realizado en el capítulo anterior y organizado a partir de las disciplinas definidas en la metodología utilizada. En los epígrafes iniciales se describen los requisitos funcionales (RF) y no funcionales (RnF), así como la encapsulación de los RF en historias de usuario. Seguidamente, se modela el diseño y la arquitectura de la solución, quedando descritos los patrones de diseño y arquitectónicos utilizados. El capítulo concluye con la descripción de los estándares de codificación empleados y una descripción de la solución obtenida.

Capítulo 3: Validación de la solución propuesta. En este capítulo se muestran los resultados de aplicar cada técnica, métrica y pruebas de software utilizada para validar la solución propuesta. Describiendo como fueron aplicadas cada una de estas validaciones. Además, se realiza la validación de las variables de la investigación con el propósito de comprobar en qué grado se reducen las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se describen una serie de conceptos relacionados con el objeto de estudio de la presente investigación, además se realiza una caracterización de la tecnología GraphQL. Por otra parte, se describe la metodología utilizada para guiar el proceso de desarrollo de la investigación, junto a los lenguajes y herramientas que se emplean. Es importante destacar que en el capítulo solo se describen las tecnologías utilizadas, sin hacer la valoración de otras, teniendo en cuenta que la solución propuesta debe ser compatible con las aplicaciones desarrolladas en el departamento Desarrollo de Componentes CEGEL, por tanto, se deben utilizar las mismas tecnologías usadas en esta área. El capítulo concluye con una descripción de los elementos teóricos que caracterizan los patrones de diseño y arquitectónicos aplicados en la solución propuesta, así como de las pruebas de software ejecutadas.

1.2. Conceptos fundamentales asociados al dominio del problema

Para una mejor comprensión del tema de investigación, es necesario dominar los siguientes conceptos:

Transferencia de Estado Representacional (*REST del inglés Representational State Transfer*): define un grupo de principios arquitectónicos por los cuales se diseñan servicios web, haciendo énfasis en los recursos del sistema, incluyendo como se accede al estado de estos recursos y como se transfieren por HTTP hacia clientes para diseño de servicios (Fielding, 2000).

GraphQL: es un lenguaje de consulta para API que brinda a los clientes la posibilidad de solicitar exactamente lo que necesitan. GraphQL hace que sea más fácil evolucionar las API a lo largo del tiempo, además de independizar el frontend[2] del backend[3], permitiendo hacer cambios en uno sin afectar al otro (Vargas, 2017).

Servicio web: es un método de comunicación entre dos dispositivos electrónicos en una red. Constituye una colección de protocolos abiertos y estándares usados para intercambiar datos entre aplicaciones o sistemas. (Diego Lazaro , 2018).

HTTP (*Protocolo de Transferencia de Hiper Textos*): es el protocolo de transmisión de información de la *World Wide Web*, es decir , el código que se establece para que el computador solicitante y quien contiene

[2] Frontend: es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios.

[3] Backend: es la capa de acceso a datos de un software o cualquier dispositivo, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos.

la información solicitada puedan comunicarse en el mismo idioma a la hora de transmitir información por la red (Conceptos, 2018).

Las autoras de la presenta investigación asumen las definiciones anteriores teniendo en cuenta que, del estudio bibliográfico realizado, estas se ajustan al objeto de estudio de la presente investigación.

1.3. Caracterización de GrahpQL

La necesidad de avanzar en productos complejos, hace que las API estén en constante innovación y evolución. Esto provoca que las tecnologías sean reemplazadas por otras más innovadoras. A partir de esto, surge GrahpQL, como un fuerte candidato a sustituir a REST, quien ha modelado y estandarizado la comunicación cliente-servidor a través del protocolo HTTP (Rodríguez, 2017).

GraphQL es una de las alternativas que han surgido para solucionar la mayor parte de los problemas de REST. Este es un protocolo agnóstico[4] que no depende de HTTP. Una de sus principales características es que el lenguaje y sintaxis usado en las peticiones es el mismo que el empleado en las respuestas. GraphQL dispone de un sistema de tipado[5] fuerte, por lo que cualquier mal uso puede ser rápidamente detectado en tiempo de desarrollo. Por otra parte, a través de este lenguaje de consulta se pueden analizar datos entrantes en interfaces fuertemente tipadas sin tener que analizar y transformar manualmente los JSON en objetos (Rodríguez, 2017).

Con el uso de REST, no se pueden elegir los datos que se reciben en el JSON[6] de respuestas, en cambio en GraphQL se puede elegir exactamente lo que se necesita. Donde REST requiere muchas peticiones, a GraphQL tan solo le hace falta una única petición para obtener los mismos datos. API REST tiene poca documentación, mientras GraphQL al ser fuertemente tipado es autodocumentado (Rodríguez, 2017).

Entre las principales características de GraphQL se encuentra (Rodríguez, 2017):

- Es fuertemente tipado.
- Las consultas en GraphQL devuelven exactamente lo que el cliente solicita.
- Requiere de una sola petición.
- Se tiene solo un punto de salida en el que se accede a los datos del servidor.

Los elementos antes descritos justifican la decisión de utilizar GraphQL en el desarrollo del componente propuesto.

[4] Agnóstico: independiente.
[5] Tipado: GraphQL schema.
[6] JSON: JavaScript Object Notation.

1.4. Metodología de desarrollo de software

Para guiar el desarrollo del componente propuesto utilizando GraphQL, se decide emplear una metodología de desarrollo de software. Estas consisten en múltiples herramientas, modelos y métodos para asistir en el proceso de desarrollo de software. En ella se define con precisión los artefactos, roles y actividades involucradas, junto con prácticas y técnicas recomendadas (Enríquez Ruiz, y otros, 2017). Como metodología de desarrollo de software a utilizar en la presente investigación, se decide seleccionar la definida por la UCI. Esta es una variación del Proceso Unificado Ágil (AUP), ya que se adapta al ciclo de vida productivo de la universidad.

1.4.1. Metodología AUP-UCI

Dentro de las metodologías ágiles se encuentra el Proceso Unificado Ágil (AUP, por sus siglas en inglés), la cual consiste en una versión simplificada de la metodología de desarrollo tradicional Proceso Racional Unificado (RUP). AUP describe la forma de desarrollar aplicaciones de software de manera fácil de entender, usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. Con el objetivo de estandarizar el proceso de desarrollo de software, la dirección de producción de la UCI tiene definida como metodología a utilizar, una variación de AUP en unión con el modelo CMMI-DEV v1.3[7], denominada AUP-UCI. La misma establece prácticas centradas en el desarrollo de productos y servicios de calidad (Sánchez, 2015).

La metodología AUP propone organizar el proceso de desarrollo de software en cuatro fases (Inicio, Elaboración, Construcción y Transición). En el caso de la adaptación de esta metodología para los proyectos de la UCI se decide mantener la fase de inicio, pero modificando su objetivo, las tres restantes fases se unifican, quedando una sola denominada ejecución y se agrega una fase de cierre (Sánchez, 2015). A continuación, en la Tabla 1 se describen cada una de las fases de la variación AUP para la UCI.

Tabla 1. Descripción de las fases de la metodología AUP-UCI

Fases AUP	Fases Variación AUP-UCI	Objetivos de las fases (Variación AUP-UCI)
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización del cliente, que permite obtener información fundamental acerca del alcance del

[7] Modelo de Madurez de Capacidades Integrado para Desarrollo (CMMI-DEV) proporciona buenas prácticas para el desarrollo y mantenimiento de productos y servicios.

		proyecto, realizar estimaciones de tiempo, esfuerzo y costo, así como decidir si se ejecuta o no el proyecto.
Elaboración **Construcción** **Transición**	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura y el diseño, y por último se implementa y realizan pruebas al producto.
Cierre	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Fuente: (Sánchez, 2015)

Disciplinas de la variación de AUP-UCI

AUP propone 7 disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decide para los proyectos de la UCI tener 7 disciplinas también, pero a un nivel más atómico que el definido en AUP. Los flujos de trabajos: modelado de negocio, requisitos y análisis y diseño en AUP están unidos en la disciplina modelo, en la variación para la UCI se consideran a cada uno de ellos disciplinas. Se mantiene la disciplina implementación, en el caso de las pruebas se desagrega en 3 disciplinas: pruebas Internas, pruebas de liberación y pruebas de aceptación. Las restantes 3 disciplinas de AUP asociadas a la parte de gestión para la variación UCI se cubren con las áreas de procesos que define CMMI-DEV v1.3 para el nivel 2, serían CM (gestión de la configuración), PP (planeación de proyecto) y PMC (monitoreo y control de proyecto) (Sánchez, 2015).

Escenarios para la disciplina Requisitos

A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (Caso de Uso del Negocio (CUN), Diagrama de Proceso del Negocio (DPN) y Modelo Conceptual (MC)), existen tres formas de encapsular los requisitos (Caso de Uso del Sistema (CUS), Historias de Usuarios (HU), Diagrama de Requisitos por Proceso (DRP)). Teniendo en cuenta lo anterior, surgen cuatro escenarios para modelar los *software*, manteniendo en dos de ellos el MC, quedando de la siguiente forma:

- **Escenario No 1:** proyectos que modelen el negocio con CUN solo pueden modelar el sistema con CUS. Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado

obtengan que puedan modelar una serie de interacciones entre los trabajadores del negocio/actores del sistema (usuario), similar a una llamada y respuesta respectivamente, donde la atención se centra en cómo el usuario va a utilizar el sistema. Es necesario que se tenga claro por el proyecto que los CUN muestran como los procesos son llevados a cabo por personas y los activos de la organización.

- **Escenario No 2:** proyectos que modelen el negocio con MC solo pueden modelar el sistema con CUS. Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no sea necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio. Se recomienda este escenario para proyectos donde el objetivo primario es la gestión y presentación de información.
- **Escenario No 3:** proyectos que modelen el negocio con DPN solo pueden modelar el sistema con DRP. Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez, y su continuidad. Se debe tener presente que este escenario es muy conveniente si se desea representar una gran cantidad de niveles de detalles y la relaciones entre los procesos identificados.
- **Escenario No 4:** proyectos que no modelen negocio, solo pueden modelar el sistema con historias de usuario (HU). Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información (Sánchez, 2015).

La aplicación de la variación de AUP en la UCI permite estandarizar el proceso de desarrollo de software en la universidad, dando cumplimiento a las buenas prácticas que define CMMI-DEV v1.3. Estas disciplinas se desarrollan en la fase de ejecución y pueden estar organizadas en iteraciones, con el propósito de obtener resultados incrementales (Sánchez, 2015).

A partir del análisis realizado sobre la variación de la metodología AUP para la UCI, las autoras del presente trabajo de diploma deciden organizar el proceso de desarrollo del componente que da cumplimiento al objetivo general de la investigación, a partir de las fases y disciplinas que propone esta variación de la metodología. La decisión se adopta teniendo en cuenta que el desarrollo del componente responde a la arquitectura sobre la cual se desarrollan varios proyectos de la red de centros de la universidad.

El escenario a utilizar es el No.4, ya que aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido, estas características se adaptan a la solución propuesta.

1.5. Herramienta de Ingeniería del Software Asistidas por Computadoras

Para el desarrollo de algunas de las tareas ingenieriles definidas por la metodología antes descrita se utiliza una herramienta CASE (por sus siglas en inglés *Computer Aided Software Engineering).* La misma está destinada a aumentar la productividad en el desarrollo del software reduciendo el coste del mismo en términos de tiempo y de dinero. Esta herramienta ayuda en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el diseño de proyectos, cálculo de costos, implementación de parte del código a partir del diseño dado, compilación automática y documentación o detección de errores (Menéndez-Barzanallana Asensi, 2016). A continuación, se describe la herramienta CASE utilizada en las tareas ingenieriles de la presente investigación.

1.5.1. Visual Paradigm 8.0

Es una herramienta que utiliza lenguaje unificado de modelado (UML, por sus siglas en inglés) que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite modelar todos los tipos de diagramas de clases, generar código desde diagramas y generar documentación. La herramienta agiliza la construcción de aplicaciones con calidad y a un menor coste de tiempo. Posibilita la generación de bases de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos, así como obtener ingeniería inversa de bases de datos (Menéndez-Barzanallana Asensi, 2016).

A partir de los elementos antes expuestos las autoras del presente trabajo de diploma deciden utilizar Visual Paradigm teniendo en cuenta que esta herramienta propicia un conjunto de funcionalidades para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Además, la Universidad de las Ciencias Informáticas cuenta con una licencia para su uso.

1.6. Lenguajes de programación

La implementación de las clases y métodos representados a través de la herramienta CASE Visual Paradigm se realiza empleando lenguajes de programación. Es importante tener en cuenta que de los lenguajes de programación que existen para desarrollar aplicaciones orientadas a la web se encuentran dos grupos fundamentales de acuerdo con la arquitectura Cliente/Servidor, la programación del lado del servidor y la programación del lado del cliente. Esta última incluye aquellos lenguajes que son interpretados por una aplicación cliente como el navegador web, entre ellos se encuentra HTML. Los lenguajes de programación del lado servidor son reconocidos, ejecutados e interpretados por el propio servidor, el que se encarga de brindar información al cliente en un formato comprensible para él. Para el desarrollo de la solución propuesta se hace necesario emplear los lenguajes java y javascript, teniendo en cuenta que son los utilizados en el departamento Desarrollo de Componentes de CEGEL.

1.6.1. Java

Java es un lenguaje de programación que se caracteriza por ser orientado a objetos, distribuido y dinámico, robusto, seguro, multitarea y portable. Su código es similar al lenguaje C y C++ con un modelo de objetos más sencillo (Pérez, 2018).

Java tiene la característica de ser al mismo tiempo compilado e interpretado. El compilador es el encargado de convertir el código fuente de un programa en un código intermedio llamado *bytecode* que es independiente de la plataforma en que se trabaje y que es ejecutado por el intérprete de Java que forma parte de la máquina virtual de java (Zamitiz, 2016). El lenguaje antes descrito se utiliza en el desarrollo de la API GraphqQL.

1.6.2. JavaScript

Es un lenguaje de programación ligero, interpretado por la mayoría de los navegadores y que les proporciona a las páginas web, efectos y funciones complementarias a las consideradas como estándar HTML. Este tipo de lenguaje de programación es de código abierto, por lo que cualquier persona puede utilizarlo sin comprar una licencia. Con frecuencia es empleado en los sitios web, para realizar acciones en el lado del cliente, estando centrado en el código fuente de la página web (Venemedia, 2014). El lenguaje antes descrito se utiliza en el desarrollo del cliente GraphqQL.

1.7. Marcos de trabajo

Para la implementación del componente propuesto, a través de los leguajes de programación antes descritos, se utilizan marcos de trabajo. En el desarrollo de software, un marco de trabajo es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado (Alegsa, 2018). Para el desarrollo de la solución propuesta se hace necesario utilizar los marcos de trabajo Spring Boot y VueJs en sus versiones 2.0.5 y 2.6.4 respectivamente. La decisión de asumir estas tecnologías se justifica teniendo en cuenta que en el caso de Spring Boot es el marco de trabajo definido en el departamento Desarrollo de Componentes de CEGEL para implementar el *backend* de las aplicaciones. En el caso de VueJs por ser un marco de trabajo progresivo para el desarrollo del *frontend* con funcionalidades intuitivas, modernas y fáciles de usar, el cual posee una comunidad de desarrollo bien activa y un código bien documentado.

1.7.1. Spring Boot 2.0.5

Spring Boot es una infraestructura ligera que elimina la mayor parte del trabajo de configurar las aplicaciones basadas en Spring, el mismo facilita la creación de proyectos con marcos de trabajo Spring eliminando la necesidad de crear largos archivos de configuración XML (siglas en inglés de *eXtensible Markup Language*). Además, provee configuraciones por defecto para Spring y otras librerías. También provee un modelo de

programación parecido a las aplicaciones java tradicionales que se inician en el método principal "*main*" (Perry, 2017).

1.7.2. VueJs 2.6.4

VueJs es un marco de trabajo progresivo para crear interfaces de usuario. El mismo está diseñado para ser adaptable de forma incremental. Está enfocado solo en la capa de vista, y es fácil de captar e integrar con otras bibliotecas o proyectos existentes. Por otro lado, Vue también es perfectamente capaz de impulsar aplicaciones sofisticadas de una sola página cuando se usa en combinación con herramientas modernas y bibliotecas de soporte (VueJs, 2019).

1.8. Entornos de Desarrollo Integrado

La implementación del componente propuesto, a partir de los lenguajes de programación y marcos de trabajo antes descritos, se complementa con el uso de entornos de desarrollo integrado (IDE, por sus siglas en inglés de *Integrated Development Environment*). Los IDE constituyen un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación (Martin Olivera, y otros, 2016).

Para el desarrollo de la solución propuesta, se utilizan como IDE las herramientas Webstorm IDE e IntelliJ IDEA en sus versiones 2.4 del 2018 respectivamente. Webstorm se utilizó para el desarrollo en la parte del *frontend* e IntelliJ en el desarrollo en la parte del *backend*.

1.8.1. IntelliJ IDEA 2018.2.4

Este IDE es soportado por varios sistemas operativos: Windows, Linux o Mac OS. Su versión gratuita soporta lenguajes como Java, Groovy, XML/XSL, Kotlin y Python, Clojure, Dart, Erlang, Go, Haxe, Perl, Scala, Haskell, Lua, estos últimos vía plugin, mientras que su versión de pago además de soportar todos los lenguajes antes mencionados soporta JavaScript, HTML, CSS, SQL, Ruby, así como PHP vía plugin (JetBrains, 2018).

1.8.2. WebStorm 2018.2.4

Este es un entorno de desarrollo inteligente que ayuda a producir código de alta calidad de manera eficiente gracias al completamiento de código, se detectan errores sobre la marcha, la navegación y refactorizaciones automatizadas. Da soporte a las recientes tecnologías, funcionando de la mejor manera con las más modernas y populares para el desarrollo web, así como AngularJS, ECMAScript 6 y Compass. Es multiplataforma, funciona en Windows, Mac OS o Linux con una única clave de licencia. Además, WebStorm

agiliza el flujo de trabajo mediante la integración con todo lo necesario para el desarrollo productivo. Además, desde el IDE se pueden utilizar varias herramientas como el depurador y terminales (Jetbrains, 2018).

1.9. NPM

El administrador de paquetes de *Node* (NPM, de la traducción al inglés *Node Package Manager*) es un gestor de paquetes que combina un conjunto de herramientas *open-source.* NPM es usado por los desarrolladores para describir los paquetes de JavaScript, especialmente aquellos paquetes de Node.js (Chirichigno, 2018).

En el desarrollo del componente propuesto se utiliza la tecnología NPM en la gestión de dependencia durante la implementación del cliente GrahpQL. La decisión de aplicar NPM se justifica, teniendo en cuenta que esta es la herramienta definida en el departamento Desarrollo de Componentes de CEGEL para la gestión de dependencia en el *frontend* de las aplicaciones.

1.10. Maven

Maven es una herramienta de gestión y configuración de proyectos de software, a través de la cual se puede administrar la compilación, los informes y la documentación de un proyecto desde un lugar central. Esta herramienta en su versión 3 tiene como objetivo garantizar la compatibilidad con versiones anteriores, mejorar la facilidad de uso, aumentar el rendimiento, permitir la integración segura y allanar el camino para implementar características altamente demandadas. (Carlos, 2014)

Objetivos de Maven:

El objetivo principal de Maven es permitir que un desarrollador comprenda el estado completo de un esfuerzo de desarrollo en el menor tiempo posible (Carlos, 2014). Para lograr este objetivo Maven se caracteriza por:

- Hacer fácil el proceso de construcción.
- Proporcionar un sistema de construcción uniforme.
- Proporcionar información de calidad del proyecto.
- Proporcionar pautas para el desarrollo de mejores prácticas.
- Permitir la migración transparente a nuevas funcionalidades.

En el desarrollo del componente propuesto se utiliza la tecnología maven en la gestión de dependencia durante la implementación del API GrahpQL. La decisión de aplicar maven se justifica, teniendo en cuenta que esta es la herramienta definida en el departamento Desarrollo de Componentes de CEGEL para la gestión de dependencia en el *backend* de las aplicaciones.

1.11. Sistema Gestor de Bases de Datos

Un sistema gestor de bases de datos (SGBD) es un conjunto de programas no visibles que administran y gestionan la información que contiene una base de datos. A través de estos se maneja todo acceso a la base de datos con el objetivo de servir de interfaz entre esta, el usuario y el software (Ramírez, 1999). Para el desarrollo de la solución propuesta se hace necesario utilizar los sistemas gestores de base de datos PostgreSQL y MongoDB, teniendo en cuenta la compatibilidad de estos con los sistemas desarrollados en el departamento Desarrollo de Componentes del CEGEL. PostgreSQL se emplea en la administración de datos relacionados con la gestión de usuarios, roles y permisos. Por otra parte, MongoDB se emplea en la administración de los datos relacionados con la gestión de las trazas.

1.11.1. PostgreSQL 9.6

Es un sistema de base de datos objeto-relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada. Se ejecuta en los principales sistemas operativos que existen en la actualidad como Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows (Microbuffer, 2011). Además, constituye un sistema de gestión de bases de datos objeto-relacional basado en Postgres. Dentro de sus principales ventajas se encuentran:

- Soporte de protocolo de comunicación encriptado por SSL[8].
- Extensiones para alta disponibilidad, nuevos tipos de índices, datos espaciales y minería de datos.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos (The PostgreSQL Global Development Group, 2019).

1.11.2. MongoDB

MongoDB es una base de datos orientada a documentos. Esto quiere decir que, en lugar de guardar los datos en registros, guarda los datos en documentos, estos son almacenados en BSON[9]. Una de las diferencias más importantes con respecto a las bases de datos relacionales, no necesita seguir un esquema. Los documentos de una misma colección o concepto similar a una tabla de una base de datos relacional pueden tener datos diferentes. MongoDB está escrito en C++, aunque las consultas se hacen pasando objetos JSON como parámetro. Las aplicaciones que necesite almacenar datos semiestructurados puede usar MongoDB, es el caso de las típicas aplicaciones CRUD o de muchos de los desarrollos web actuales (Kristina, 2013).

[8] Seguridad de la capa de transporte (ALEGSA, 2018).

[9] BSON: JSON Binario.

1.12. Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características, una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores, otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias (Rojas, 2010).

Los patrones de diseño se pueden agrupar en dos grandes grupos; los GRASP (por sus siglas en inglés *General Responsibility Assignment Software Patterns*), que son patrones generales de software para asignación de responsabilidades y los GOF (por sus siglas en inglés *Gang of Four*), encargados de la inicialización, agrupación y comunicación de los objetos. En el Capítulo 2 de la presente investigación, se describen los patrones de diseño que fueron empleados en el desarrollo del componente propuesto.

1.13. Patrón arquitectónico Vuex

Un patrón arquitectónico es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado. Los patrones arquitectónicos son similares a los patrones de diseño, pero tienen un alcance más amplio. Estos describen un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución (Eugenia, 2011). En el diseño arquitectónico de la presente investigación se aplican los siguientes patrones:

- Flux: es un patrón arquitectónico que define un flujo de datos unidireccional. Con este patrón los datos viajan desde la vista por medio de acciones y llegan a un *store*[10] desde el cual se actualiza la vista nuevamente. Las vistas están constituidas por los componentes web, el *store* guarda los datos, es decir el estado de la aplicación. No hay métodos en el *store* que permitan modificarlo, esto se realiza a partir de las acciones, y una acción no es más que un objeto JavaScript que indica una intención de realizar algo y lleva datos asociados si es necesario (Facebook Corporation, 2019).

- Redux: es un patrón arquitectónico basado en Flux que define tres conceptos fundamentales: una única fuente de datos, esto significa utilizar un único *store*. El estado es solo lectura, no se puede modificar el estado directamente, pues solo se puede leer de él para representarlo en la vista, y si se desea modificar se realiza a través de acciones. Por último, los cambios en el estado se realizan a través de funciones puras llamadas *reducers*, ya que no se puede cambiar el estado directamente (Redux, 2019).

[10] Almacén

- Vuex: es un patrón arquitectónico basada en Redux y Flux y además una biblioteca para la gestión de estados de la aplicación. Este patrón define un *store* centralizado para todos los componentes de una aplicación de VueJs con reglas que garantizan que el estado solo se puede modificar a través de mutaciones (Vuex, 2019).

1.14. Patrón arquitectónico N-Capas

El patrón arquitectónico N-Capas es una extensión del patrón Capas tradicional (este ayuda a estructurar las aplicaciones que se pueden descomponer en grupos de subtareas en la que cada grupo de subtareas está en un nivel particular de abstracción). En el nivel más alto y abstracto, la vista de arquitectura lógica de un sistema puede considerarse como un conjunto de servicios relacionados agrupados en diversas capas (Escalante, 2014)

En el Capítulo 2 de la presente investigación, se describe la aplicación de estos patrones arquitectónicos en el diseño del componente propuesto.

1.15. Pruebas del software

Uno de los elementos principales para certificar la calidad de una aplicación informática lo constituye el resultado de las pruebas que se practican. Un control y una gestión de calidad implementados de forma adecuada, especialmente durante el proceso de desarrollo del software, aumentan la calidad del sistema final, reducen los costos de avance y acortan el tiempo necesario para el desarrollo. Para determinar el nivel de calidad se deben efectuar medidas o pruebas que permitan comprobar el grado de cumplimiento respecto a las especificaciones principales del sistema (Pressman, 2010).

Objetivo de las pruebas (Pressman, 2010):

- Validar y probar los requisitos que debe cumplir el software.
- Verificar que los requisitos fueron implementados correctamente.
- Encontrar y documentar los defectos que puedan afectar la calidad del software.
- Verificar que el software trabaje como fue diseñado.

Teniendo en cuenta la metodología seleccionada para guiar el desarrollo del componente propuesto en la presente investigación, a continuación, se describen las disciplinas de pruebas de software que esta plantea:

- Pruebas internas: son realizadas por sus propios desarrolladores, verificando el resultado de la implementación, probando cada construcción según sea necesario, así como las versiones finales a

ser liberadas. Los artefactos necesarios para la realización de estas pruebas son los casos de pruebas (Sánchez, 2015).

- Pruebas de liberación: son diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación (Sánchez, 2015).
- Pruebas de aceptación: son las únicas pruebas que son realizadas por los clientes. Consiste en comprobar si el producto está listo para ser implantado para el uso operativo en el entorno del usuario. Para la realización de esta tarea se define utilizar el proceso llamado prueba alfa, el cual se lleva a cabo en el sitio del desarrollador por un grupo representativo de usuarios finales; el software se utiliza en un escenario natural con el desarrollador registrando los errores y problemas de uso detectados por los clientes (Pressman, 2010).

Para complementar la ejecución de diferentes tipos de pruebas de software en cada una de las disciplinas antes descritas, es necesario organizar las pruebas en los niveles de unidad, integración, sistema y aceptación.

Las pruebas a nivel de unidad se concentran en el esfuerzo de verificación de la unidad más pequeña del diseño: el componente o módulo de software. Tomando como guía la descripción del diseño a nivel de componente, se prueban importantes caminos de control para describir errores dentro de los límites del módulo. El alcance restringido que se ha determinado para las pruebas de unidad limita la relativa complejidad de las pruebas y los errores que éstas descubren (Pressman, 2010).

Las pruebas a nivel de integración tienen como objetivo identificar errores introducidos por la combinación de programas o componentes probados unitariamente, para asegurar que la comunicación, enlaces y los datos compartidos ocurran apropiadamente. Se diseñan para descubrir errores o completitud en las especificaciones de las interfaces (Pressman, 2010).

Las pruebas a nivel de sistema tienen como objetivo verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las operaciones apropiadas funcionando como un todo. Es similar a la prueba de integración, pero con un alcance más amplio (Pressman, 2010).

Las pruebas a nivel de aceptación son las únicas pruebas que son realizadas por los clientes. Consiste en comprobar si el producto está listo para ser implantado para el uso operativo en el entorno del usuario. (Pressman, 2010).

Para el caso de la presente investigación se establece como estrategia, realizar pruebas funcionales y de usabilidad solo a nivel de unidad, teniendo en cuenta que el alcance del trabajo de diploma está acotado al

desarrollo del componente sin tener en cuenta la integración de este con otras aplicaciones. En la estrategia también se define la ejecución de pruebas a nivel de aceptación con el propósito de corroborar que el componente propuesto cumple con las necesidades del cliente. En el caso de la presente investigación el cliente lo constituye el equipo de arquitectos de software del departamento Desarrollo de Componentes de CEGEL.

1.16. Conclusiones parciales

El desarrollo del marco teórico referencial relacionado con el estudio de la tecnología GraphQL, facilita una mejor comprensión del objeto de estudio de la presente investigación. Por otra parte, el análisis de las características de la variación de la metodología AUP para la UCI, así como la fundamentación teórica de las tecnologías utilizadas, demuestran la correspondencia de la selección de cada uno de estos elementos con la implementación del componente propuesto. El estudio teórico realizado sobre patrones de diseño y arquitectónicos, así como de las pruebas de software, justifica la aplicación de cada patrón en el diseño del componente propuesto y de la estrategia de prueba diseñada para la validación de este.

CAPÍTULO 2: ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA

2.1. Introducción

En este capítulo se realiza una descripción de cómo fueron aplicadas las disciplinas de requisitos, análisis y diseño e implementación de la metodología AUP variación UCI en el desarrollo del componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas. Además, se describen los entregables obtenidos en cada una de estas disciplinas.

En la disciplina de requisitos se explican las técnicas aplicadas en la captura de estos, hasta llegar al listado final de requisitos funcionales y no funcionales. Los funcionales son encapsulados a través de historias de usuarios. En la disciplina también se describen las técnicas utilizadas en la validación de los requisitos. Por otra parte, en la disciplina de análisis y diseño se describe la arquitectura de la solución propuesta, los patrones arquitectónicos y de diseño aplicados, así como la estructura del diagrama de clases del diseño y el modelo de datos. En los epígrafes finales del capítulo se describen los estándares de codificación empleados en la disciplina de implementación y las características de la solución obtenida.

2.2. Descripción del negocio de la solución

La solución propuesta se centra en la obtención de un API y un cliente GraphQL que administre los usuarios, roles, permisos y trazas de una aplicación web. La solución hace uso del marco de trabajo Spring Boot para el desarrollo de la API, por otra parte, para el desarrollo del cliente se utiliza el marco de trabajo VueJs. La API tiene como objetivo brindar los datos necesarios relacionados con la información de los usuarios, roles, permisos y trazas que se gestionan en aplicaciones web. En el cliente se obtienen los datos expuestos desde la API para ser gestionados a través de componentes definidos por el marco de trabajo VueJs. La implementación del API y el cliente a través de GraphQL tiene como principal ventaja la posibilidad de obtener datos necesarios sin tener que realizar múltiples llamadas secuenciales o en paralelas.

2.3. Disciplina de requisitos

El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir y comprende la administración de los requisitos funcionales y no funcionales del producto (Rodríguez, 2015). Teniendo en cuenta que el componente responde al cumplimiento del objetivo general de la presente investigación, tiene un negocio bien definido y fácil de comprender, sin necesidad de realizar un modelado de este, las autoras del presente trabajo de diploma deciden utilizar el escenario número cuatro (HU). En este epígrafe se presentan las técnicas para la identificación de los requisitos, la especificación de los mismos, así como las historias de usuario obtenidas (Pressman, 2010).

2.3.1. Técnicas para la captura de requisitos

Para identificar las necesidades del cliente se utilizan técnicas que permiten determinar y documentar los requisitos para el desarrollo de la solución. Esta actividad es continua durante el ciclo de desarrollo y combina, en diferentes puntos, diversas técnicas de identificación para obtener la visión más completa de las necesidades del usuario final (Pressman, 2010). Para la captura de los requisitos del componente propuesto se utilizan las siguientes técnicas:

- **Entrevista:** es de gran utilidad para obtener información cualitativa, requiere seleccionar bien a los entrevistados para obtener la mayor cantidad de información en el menor tiempo posible. Es muy aceptada y permite acercarse al problema de una manera natural (Sommerville, 2011). Esta técnica fue aplicada a los arquitectos de CEGEL. En el Anexo 1, se encuentra la guía de preguntas utilizada en el desarrollo de la entrevista.
- **Tormenta de ideas:** es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios (Pressman, 2010). Esta técnica se evidencia durante las reuniones con el cliente donde, luego de un diálogo entre ambas partes, se obtuvo como resultado un conjunto de acuerdos con el fin de refinar las necesidades del cliente. En el Anexo 2 del presente documento se relacionan los acuerdos tomados durante la tormenta de ideas.

2.3.2. Especificación de requisitos

Los requisitos constituyen un punto clave en el desarrollo de aplicaciones informáticas. Un gran número de proyectos de software fracasan debido a una mala definición, especificación o administración de requisitos. Factores tales como requisitos incompletos o mal manejo de los cambios de los requisitos, llevan a proyectos completos al fracaso total. Los requisitos se enfocan en las especificaciones de lo que se desea desarrollar y tienen dos clasificaciones: requisitos funcionales y no funcionales. Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema y de cómo se debe comportar en distintas situaciones. Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema (Sommerville, 2011).

En la presente investigación, una vez realizado el levantamiento de información e identificadas las necesidades del cliente, se identificaron un total de 15 requisitos funcionales (RF). A continuación, en la Tabla 2 se presentan cada uno de estos y sus respectivas descripciones (Pressman, 2010).

Tabla 2. Requisitos Funcionales

No.	Nombre	Descripción
RF1	Listar usuario	Lista todos los usuarios que están registrados en el sistema, mostrando el nombre completo del mismo, el usuario, si está activo o no, la fecha de inicio, así como las acciones definidas para cada uno de ellos.
RF2	Buscar usuario	Permite dado un criterio de búsqueda (primer nombre, segundo nombre, primer apellido, segundo apellido, usuario y carné de identidad) obtener los usuarios que coincidan con el mismo.
RF3	Modificar usuario	Permitir modificar en el sistema los siguientes atributos que caracterizan un usuario: primer nombre, segundo nombre, primer apellido, segundo apellido, usuario, correo electrónico, carné de identidad, roles y permisos asignados, así como seleccionar si se encuentra activo o no.
RF4	Registrar usuario	Permite registrar un usuario en el sistema, llenando los siguientes campos: primer nombre, segundo nombre, primer apellido, segundo apellido, usuario, correo electrónico, carné de identidad, roles y permisos asignados, así como señalar si está activo o no.
RF5	Visualizar usuario	Permite mostrar el usuario, el nombre completo, si está activo, los roles y permisos asignados.
RF6	Listar rol	Lista todos los roles que están registrados en el sistema, mostrando el nombre, la descripción, si se encuentra activo o no y las acciones editar y modificar.
RF7	Buscar rol	Permite dado el nombre del rol obtener los roles que coincidan con el mismo.
RF8	Modificar rol	Permitir modificar los datos de un rol del sistema.
RF9	Registrar rol	Permite registrar un rol en el sistema, llenando los siguientes campos: nombre, descripción y permisos asignados, así como seleccionar si se encuentra activo o no.
RF10	Listar permiso	Lista todos los permisos que están registrados en el sistema, mostrando el nombre del permiso, la descripción, si se encuentra activo o no y las acciones editar.
RF11	Buscar permiso	Permite, dado el nombre del permiso, obtener los permisos que coincidan con el mismo.

RF12	Modificar permiso	Permite modificar los siguientes atributos que caracterizan un permiso: descripción y seleccionar si se encuentra activo o no.
RF13	Cambiar contraseña	La funcionalidad permite el cambio de contraseña para un usuario
RF14	Listar traza	Lista todas las trazas que están registrados en el sistema, mostrando el nombre del usuario que dejó trazas en el sistema, la dirección IP[11] de donde se produjo la misma, la fecha y la acción visualizar trazas.
RF15	Buscar traza	Permite dado un criterio de búsqueda (usuario, la dirección IP, fecha de inicio y fecha de fin para buscar en un rango de fecha) obtener las trazas que coincidan con el mismo.
RF16	Visualizar traza	Permite mostrar el usuario, la descripción, la dirección IP y la fecha en que se produjo la traza seleccionada.

Fuente: elaboración propia

En el caso de los requisitos no funcionales (RnF) se definieron un total de 11, los cuales son clasificados y descritos a continuación.

Requisitos no funcionales de usabilidad:

RnF 1: el componente debe ser fácil de utilizar por los desarrolladores.
RnF 2: cada funcionalidad debe brindar información referida a su comportamiento.
RnF 3: en el componente se deben visualizar todos los mensajes en idioma español. La tipografía debe ser uniforme, de tamaño adecuado.

Requisito no funcional de eficiencia:

RnF 4: el componente no debe demorarse más de 6 segundos en responder las peticiones.

Requisito no funcional de confiabilidad:

RnF 5: el componente debe estar disponible las 24 horas del día.

Requisitos de interfaz y apariencia:

RnF 6: se puede utilizar la biblioteca de componente *Vuetify*[12] para que el componente cuente con los estilos *Mateial Desing*[13].

[11] IP: Protocolo de internet
[12] Vuetify: librería para componentes visuales.
[13] Materia Desing: lenguaje visual que sintetiza los principios clásicos del buen diseño.

RnF 7: las ventanas del sistema tienen que ser con los colores de los estilos de *Material Desing*.
RnF8: todos los botones de confirmación deben estar a la izquierda y el de negación a la derecha.

Requisitos de software:

RnF9: se debe tener instalado en la computadora la máquina virtual de java en su versión 8 o superior para el funcionamiento del componente.
RnF10: el componente debe funcionar sobre los sistemas operativos Windows y Linux.

Requisitos de hardware

RNF11: para ejecutar el componente la computadora debe contar con las siguientes características:
Servidor web Apache 2.4. Procesador Core 2 duo a 2.0 GHz como mínimo 2 Gb de memoria RAM, además de una tarjeta de red.

2.3.3. Validación de los requisitos

Con el objetivo de garantizar que el componente a desarrollar se corresponde con las necesidades del cliente, cada uno de los requisitos identificados fueron validados antes de llegar a la disciplina de análisis y diseño. La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos. Para la validación de estos se utilizaron las siguientes técnicas:

- **Generación de casos de prueba:** los requerimientos deben ser comprobables. Si las pruebas para los requerimientos se diseñan como parte del proceso de validación, se puede revelar con frecuencia problemas en los requerimientos. Si una prueba es difícil o imposible de diseñar, significa que los requerimientos son difíciles de implementar, por lo que deben reconsiderarse. (Pressman, 2010). En el caso de la presente investigación se generaron un total de 10 descripciones de casos de prueba (DCP), en la Tabla 3 se describe la DCP correspondiente al RF 9, el resto de las DCP generadas para la validación del componente se encuentran entre los artefactos entregables de la tesis.

Tabla 3. DCP correspondiente al RF 9: registrar rol

Escenario	Nombre del rol	Descripción	Activo	Permisos	Respuesta del sistema	Flujo central
EC 1.1: Registrar rol con	V	V	V	V	El sistema ha registrado un rol satisfactoriamente.	Inicio/Administración/Rol
	Administrador	Administra el sistema	sí	sí		

campos correctos						
EC 1.2: Registrar rol con campos incompletos	I	V	V	V	El sistema muestra un mensaje indicando que: "Este campo es obligatorio" y no habilita el botón Guardar.	
	(Vacío)	Administra el sistema	sí	sí		
	V	I	V	V	El sistema muestra un mensaje indicando que: "Este campo es obligatorio" y no habilita el botón Guardar.	
	Administrador	(Vacío)	sí	sí		
	V	V	V	I	El sistema muestra un mensaje indicando que: "Este campo es obligatorio" y no habilita el botón Guardar.	
	Administrador	Administra el sistema	sí	no		
Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.						

Fuente: elaboración propia

- **Construcción de prototipos de interfaz de usuario:** esta técnica permite hacer simulaciones del componente implementado y brinda la posibilidad a los especialistas de tener una idea de cómo serán las interfaces del componente una vez desarrollado. En la Tabla 5 del epígrafe 2.3.4 se muestra el prototipo no funcional correspondiente al RF 8: registrar rol.

- **Revisiones formales de los requisitos:** se realizaron revisiones formales de cada requisito, por parte del cliente y el equipo de desarrollo, quedando validado que la interpretación de cada una de las

descripciones no es ambigua, ni presenta omisiones o errores que hagan que la descripción del requisito no se corresponda con las necesidades del cliente.

Métrica Calidad de la especificación: para medir la calidad de la especificación de los requisitos de software se aplicó la métrica Calidad de la Especificación (CE). El empleo de esta métrica permite obtener un alto nivel de entendimiento y precisión de los requisitos, para llegar a ello, se debe primeramente calcular el total de requisitos de software como se muestra a continuación:

Nr = Nf + Nnf

Nr: total de requisitos de software.
Nf: cantidad de requisitos funcionales.
Nnf: cantidad de requisitos no funcionales.

Sustituyendo los valores en la ecuación, se obtiene:

Nr = 16 + 11
Nr = 27

Para calcular la especificidad de los requisitos (ER) o ausencia de ambigüedad en los mismos se realiza la siguiente operación:

Q1 = Nui / Nr
Nui: número de requisitos para los cuales todos los revisores tuvieron interpretaciones idénticas.

Para sustituir los valores de las variables en la ecuación se tuvo en cuenta que, de los requisitos especificados para el desarrollo del componente, dos de ellos causaron contradicción en sus interpretaciones (RF 6 y RF 11). Por tanto, la variable Q1 obtiene el siguiente valor:

Q1 = 25 / 27
Q1 = 0.93

Es importante aclarar que mientras más cerca de 1 está el valor de Q1, menor es la ambigüedad. Teniendo en cuenta el resultado anterior, igual a 0.93, se concluye que el 93% de los requisitos es entendible. Los dos requisitos identificados como ambiguos fueron modificados y validados para garantizar su correcta comprensión, llegando al resultado ideal de Q1=1. Este resultado demuestra que el 100 % de los requisitos son fáciles de comprender.

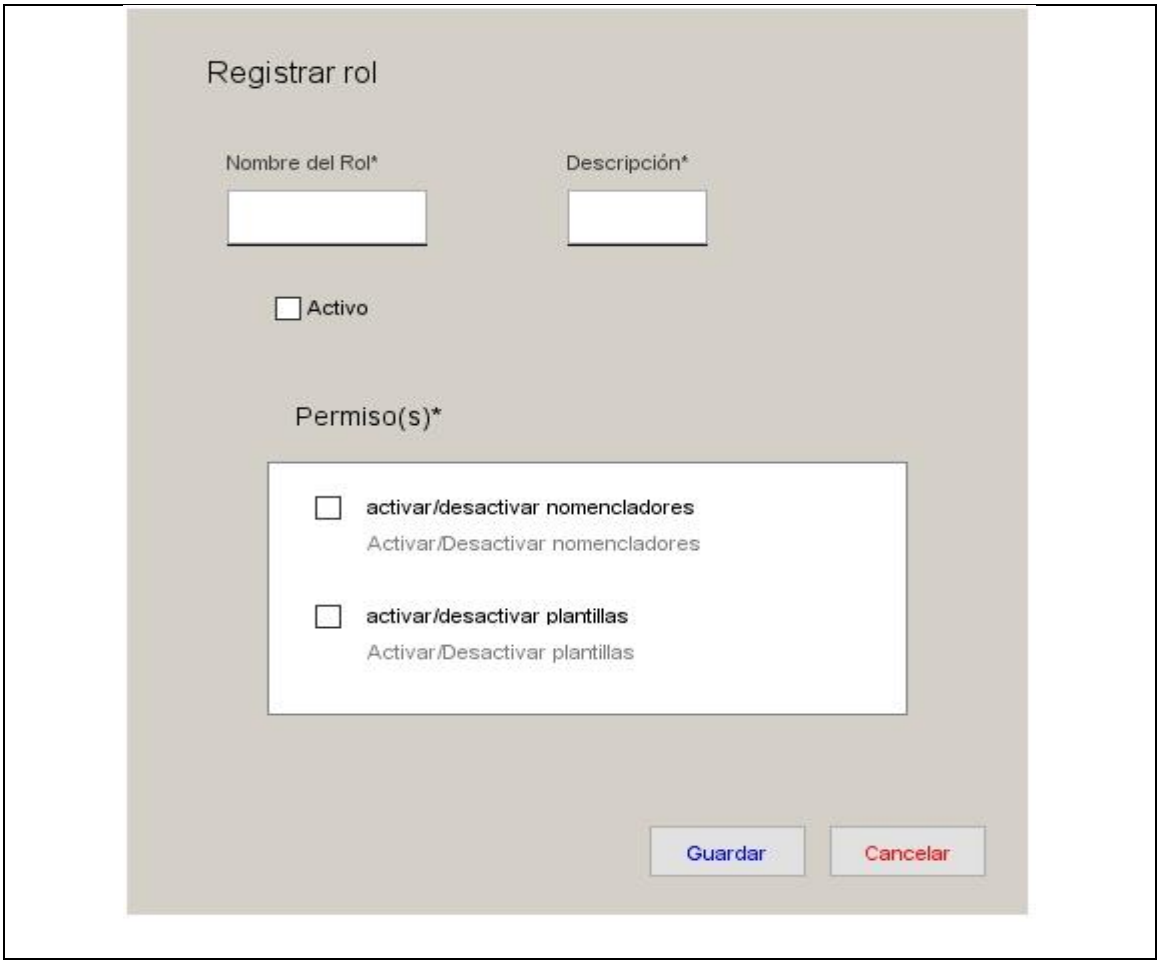
2.3.4. Historias de usuarios

Las historias de usuario (HU) son descripciones, siempre muy cortas y esquemáticas, que resumen la necesidad concreta de un usuario al utilizar un producto o servicio, así como la solución que la satisface. Su función principal es identificar problemas percibidos, proponer soluciones y estimar el esfuerzo que requieren implementar las ideas propuestas (Hoc, 2016).

Para el desarrollo del componente propuesto se definieron un total de 16 HU, una por cada RF. A continuación, en la Tabla 4 se describe la HU correspondiente al RF9 Registrar rol, teniendo en cuenta que este es uno de los RF de mayor complejidad de implementación. El resto de las HU se pueden consultar en el Anexo 3.

Tabla 4. Historia de Usuario Registrar Rol

Número: 9	**Nombre del requisito:** Registrar rol
Programadores: Dayanis Pérez Pérez Migyara M. González Labori	**Iteración Asignada:** 1
Prioridad: media	**Tiempo Estimado:** 3 días
Riesgo en Desarrollo: media	**Tiempo Real:** 2 días
Descripción: permite registrar un rol en el sistema, llenando los campos: nombre, descripción, permisos asignados, así como seleccionar si se encuentra activo o no. Una vez introducidos los datos se selecciona la opción Guardar para guardar los datos o la opción Cancelar para cerrar la interfaz y regresar a la interfaz de Gestionar rol.	
Observaciones:	
Prototipo de interfaz:	



Fuente: elaboración propia

2.4. Disciplina de análisis y diseño

En esta disciplina los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo la arquitectura). Además, en esta disciplina se modela el componente y su forma para que soporte todos los requisitos, incluyendo los requisitos no funcionales. (Sánchez, 2015)

2.4.1. Diseño arquitectónico

El diseño arquitectónico garantiza cómo debe organizarse un sistema y cómo tiene que diseñarse la estructura global del mismo. En el modelo del proceso de desarrollo de software, el diseño arquitectónico es la primera etapa en el proceso de diseño del software. Es el enlace crucial entre el diseño y la ingeniería de requerimientos, ya que identifica los principales componentes estructurales en un sistema y la relación entre ellos. La salida del proceso de diseño arquitectónico consiste en un modelo que describe la forma en que se organiza el sistema como un conjunto de componentes en comunicación (Sommerville, 2011).

La solución propuesta está estructurada por un cliente y una API GraphQL, utilizando una arquitectura que define un *frontend* como la parte del software que interactúa con los usuarios y un *backend* como la parte que procesa las entradas desde el *frontend*. El cliente GraphQL utiliza el patrón arquitectónico para gestión de estado de aplicación Vuex. Además, se utiliza la biblioteca *apollo-client* para el manejo de estado de los datos. En la Figura 1 se ilustra la arquitectura definida para el *frontend* del componente propuesto.

En el diagrama de la Figura 1 se representa el marco de trabajo para interfaces de usuario *Vuetify.* El cual se utiliza para definir los componentes visuales para los roles, trazas, permisos y usuarios. En la gestión del estado de la aplicación se emplea la biblioteca Vuex, la cual define una acción que lanzan una mutación correspondiente a la misma y esta modifica el estado deseado, que a su vez es reflejado en los componentes visuales. Por otra parte, se utiliza la biblioteca *apollo-client* para la gestión del estado de los datos. Esta biblioteca define mutaciones, consultas y suscripciones que se utilizan para comunicarse con la API GraphQL. Además de un *store* que guarda el estado de los datos para ser utilizados en los componentes visuales. *Apollo-client* también define una cache para brindar datos que no hayan cambiado durante las consultas a la API GraphQL.

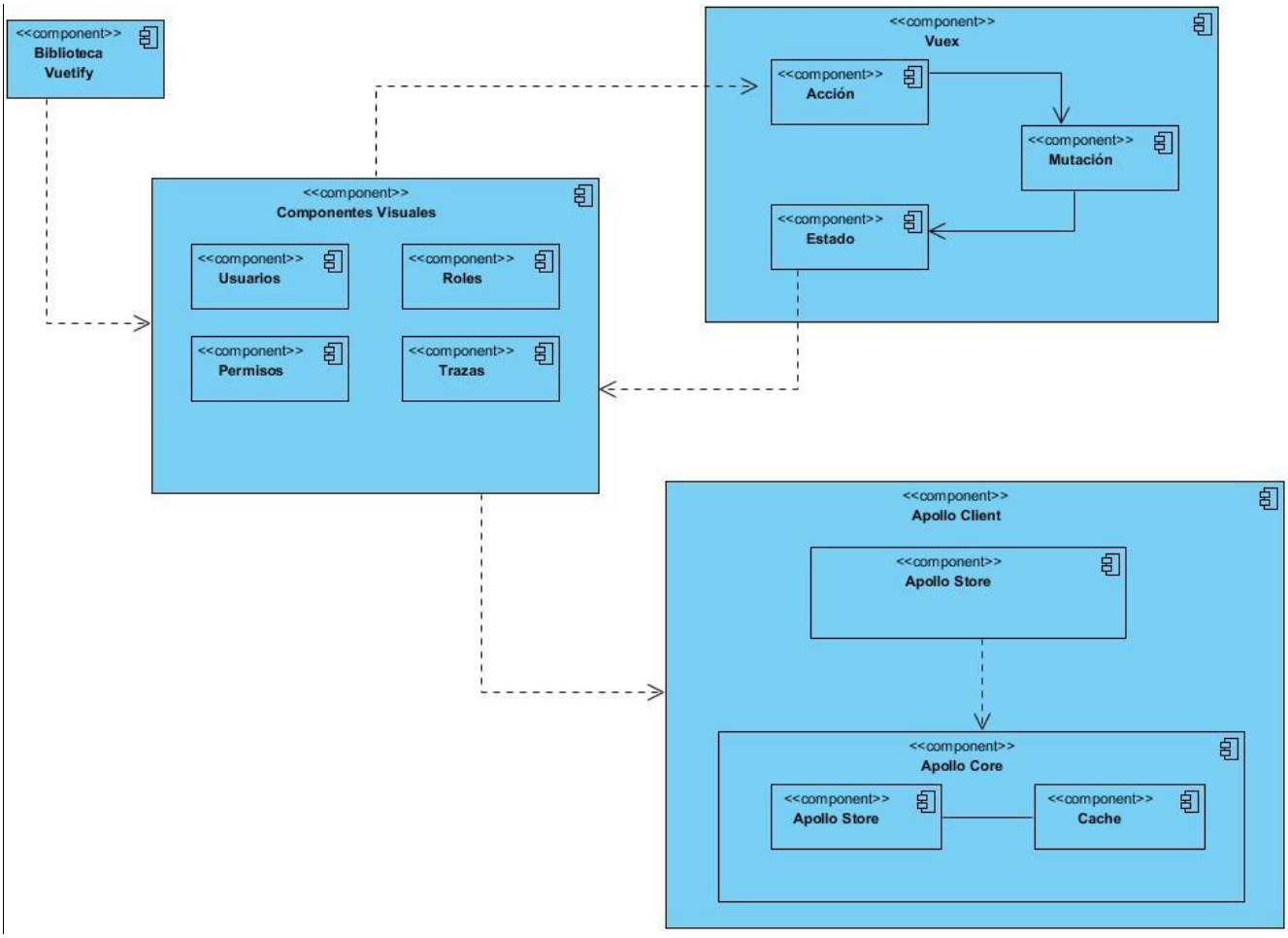


Figura 1. Arquitectura del Cliente-GraphQL
Fuente: elaboración propia

Por otra parte, en la API GraphQL del componente propuesto se aplica el patrón arquitectónico N-Capas, definiéndose las capas web, infraestructura y dominio.

La arquitectura basada en N-Capas se enfoca principalmente en el agrupamiento de funcionalidad relacionada dentro de una aplicación en distintas capas que son colocadas verticalmente una encima de otra. La funcionalidad dentro de cada capa se relaciona con una responsabilidad especifica. El dividir en capas una aplicación, permite la separación de responsabilidades lo que proporciona una mayor flexibilidad y un mejor mantenimiento (Muñoz Serafin, 2018).

En la Figura 2 se muestra el componente *GraphQLController* de la capa Web que es donde se encuentra el único punto de acceso de Graphql. En la capa Dominio se encuentran las clases de persistencia, y los componentes repositorios los cuales son los encargados de la interacción con la base de datos. En la capa Infraestructura se encuentran las implementaciones de los servicios que definen las funciones necesarias

para devolver los datos solicitados por el cliente GraphQL. Esta capa facilita además todas las interfaces de los servicios, las cuales son utilizadas por *GraphQLController.*

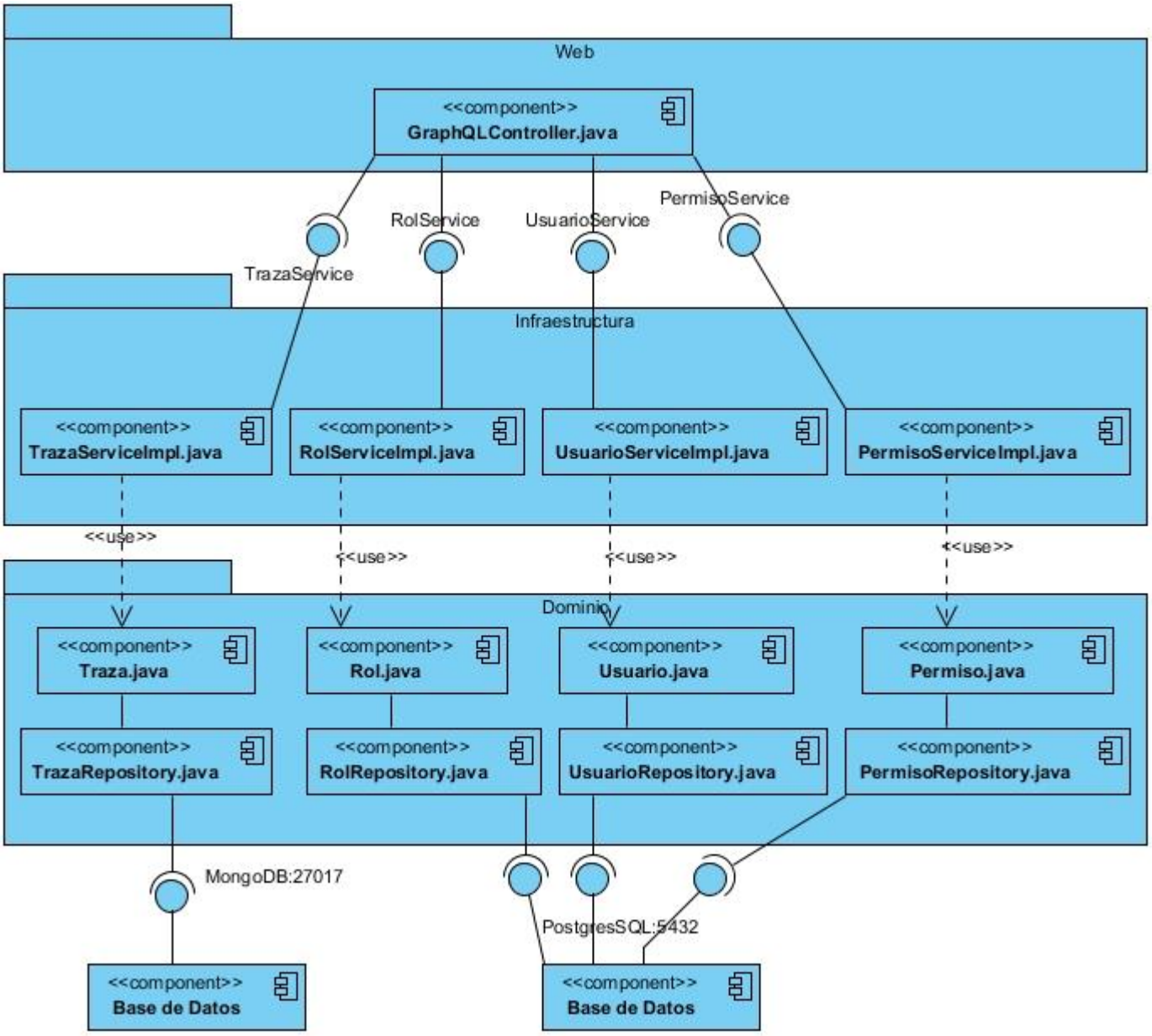


Figura 2. Arquitectura API-GraphQL
Fuente: elaboración propia

2.4.2. Patrones de diseño

Para diseñar el componente se emplearon un conjunto de patrones de diseño, los cuales son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software. A continuación, se describen los patrones empleados:

Patrones GRASP:

- Creador: el uso de este patrón se evidencia en las clases *TrazaServiceImpl*, *RolServiceImpl*, *UsuarioServiceImpl* y *PermisoServiceImpl*, teniendo estas la responsabilidad de instanciar objetos para cumplir sus funciones. En la Figura 3, se evidencia el uso de este patrón en la clase *UsuarioServiceImpl* cuando se crea un objeto de tipo usuario dado los parámetros introducidos en el método registrar usuario.

```
@GraphQLMutation(name = "setUsuario", description = "Inserta y actualiza los usuarios en la base dato")
@Override
public Usuario registrarUsuario(@GraphQLArgument(name = "usuario") UsuarioDTO usuarioDTO) {
    if (usuarioDTO.getId() == null) {
        if (usuarioDTO.getUsername() != null && !usuarioRepository.findByUsername(usuarioDTO.getUsername())
                .isPresent()) {
            Usuario usuario = usuarioFactory.createUsuario(usuarioDTO);
            usuario.setFechainicio(LocalDate.now());
            Usuario save = usuarioRepository.save(usuario);
            notifier.publish(TiposTraza.GENERIC_REGISTRAR,  descripcion: "el usuario : " + save.getUsername());
            return save;
        } else {
            throw new ExceptionPitcher("El usuario " + usuarioDTO.getUsername() + " ya está en uso");
        }
    } else {
        Usuario old = usuarioRepository.getOne(usuarioDTO.getId());
        Usuario usuario = usuarioFactory.createUsuario(usuarioDTO);
        usuario.setFechainicio(old.getFechainicio());
        usuario.setId(usuarioDTO.getId());

        Usuario save = usuarioRepository.save(usuario);
        ntifier.publish(TiposTraza.GENERIC_MODIFICAR, "el usuario : " + save.getUsername());
        return save;

    }
}
```

Figura 3. Aplicación del patrón Creador en la clase *UsuarioServiceImpl*
Fuente: elaboración propia

- Controlador: este patrón se evidencia en la clase *GraphQLController* que se encarga de crear el contexto de GraphQL y publicar el único punto de acceso a la API GraphQL, ver Figura 4.

```
public GraphQLController(PermisoService permisoQuery, RolService rolService, TrazaService trazaService,
                         UsuarioService usuarioService) {
    // Schema generated from query classes
    GraphQLSchema schema = new GraphQLSchemaGenerator().withResolverBuilders(
            // Resolve by annotations
            new AnnotatedResolverBuilder(),
            // Resolve public methods inside root package
            new PublicResolverBuilder( ...basePackages: "cegel.uci.cu.sigquobase"))
            // Query Resolvers
            .withOperationsFromSingleton(permisoQuery)
            .withOperationsFromSingleton(rolService)
            .withOperationsFromSingleton(trazaService)
            .withOperationsFromSingleton(usuarioService)
            .withValueMapperFactory(new JacksonValueMapperFactory())
            .generate();

    graphQL = GraphQL.newGraphQL(schema).build();
}

@PostMapping(value = "/graphql", consumes = MediaType.APPLICATION_JSON_UTF8_VALUE,
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
```

Figura 4. Aplicación del patrón Controlador en la clase *GraphQLController*
Fuente: elaboración propia

- Experto: este patrón se evidencia en la interfaz *TrazaRepository*, *RolRepository*, *UsuarioRepository* y *PermisoRepository*. Estas son las encargadas de comunicar las clases del negocio con la base de datos. En la Figura 5 se muestra la interfaz *RolRepository*, que además del método mostrado presenta todos los métodos necesarios para la comunicación.

```
@Repository
public interface PermisoRepository extends JpaRepository<Permiso, Long>, JpaSpecificationExecutor<Permiso> {
    Optional<Permiso> findBByPermiso(String permiso);
}
```

Figura 5. Aplicación del patrón Experto en la clase *PermisolRepository*
Fuente: elaboración propia

- Alta cohesión: el patrón se evidencia en el 52 % de las clases del componente, de tal forma que se disminuye la sobrecarga de responsabilidades entre las clases. Este resultado se demuestra en el Capítulo 3 con los resultados de la validación del diseño a partir de la aplicación de métricas.
- Bajo acoplamiento: teniendo en cuenta la importancia que se le atribuye a realizar un diseño de clases independientes que pueda soportar los cambios de forma fácil y a su vez permitan la reutilización, el patrón bajo acoplamiento se evidencia en el 52 % de las clases diseñadas en el componente, tal y como se demuestra en el Capítulo 3 con los resultados de la validación del diseño a partir de la aplicación de métricas.

Patrones GOF:

- Instancia única (*Singleton*): teniendo en cuenta que para el desarrollo del componente propuesto se utiliza el marco de trabajo Spring Boot, esto asegura que todas las clases del componente mantengan una sola instancia y proporcionan un punto de acceso global a ellas durante la ejecución.
- Mediador: se aplica en la clase *Notifier*, la cual es la encargada de coordinar la comunicación entre la clase *TrazaServiceImpl* y el resto de las implementaciones de los servicios del componente con el propósito de que trabajen conjuntamente y permitan almacenar las trazas. En la Figura 6 se evidencia el uso de este patrón.

```
@GraphQLMutation(name = "changePasswordUsuario",description ="permite el cambio de contraseña para un usuario")
@Override
public Boolean cambiarContrasena(@GraphQLArgument(name = "cambio") ContrasenaDTO contrasenaDTO) {
    Optional<Usuario> byId = usuarioRepository.findById(contrasenaDTO.getIduser());
    if (byId.isPresent()) {
        Usuario user = byId.get();
        if (new BCryptPasswordEncoder().matches(contrasenaDTO.getOldPassword(), user.getPassword())) {
            if (contrasenaDTO.getPassword().equals(contrasenaDTO.getConfirmpassword())) {
                user.setPassword(new BCryptPasswordEncoder().encode(contrasenaDTO.getPassword()));
                Usuario save = usuarioRepository.save(user);
                if (save != null) {
                    notifier.publish(TiposTraza.GENERIC_CAMBIA_CONTRASENA, contrasenaDTO.getIduser().toString());
                    return true;
                } else {
                    return false;
                }
            } else {
                throw new ExceptionPitcher("La nueva contraseña no coicide con la confirmada");
            }
        } else {
            throw new ExceptionPitcher("La antigua contraseña no coincide con la introducida");
        }
    }
    throw new ExceptionPitcher("El usuario no se encuentra en el servidor");
}
```

Figura 6. Aplicación del patrón Mediador en la clase *Notifier*
Fuente: elaboración propia

- Método de fabricación: separa la clase que crea los objetos de la jerarquía de objetos instanciados, el uso de este patrón se evidencia en las clases *UsuarioFactory*, *RolFactory*, *PermisoFactory* y *TrazaFactory*. En la Figura 7 se muestra el uso de este patrón en la clase *PermisoFactory*.

```
public class PermisoFactory {

    public Permiso createPermiso(PermisoDTO permisoDTO){
        return Permiso.builder()
                .description(permisoDTO.getDescription())
                .permiso(permisoDTO.getPermiso())
                .activo(permisoDTO.isActivo())
                .build();
    }
}
```

Figura 7. Aplicación del patrón Mediador en la clase *PermisoFactory*
Fuente: elaboración propia

2.4.3. Diagrama de clases del diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y las interfaces que participan en el sistema con sus relaciones estructurales y de herencia. Los diagramas de este tipo contienen las definiciones de las entidades del software en vez de conceptos del mundo real. Además son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea una vista lógica de la información que se maneja en el sistema, los componentes que se encargan del funcionamiento y la relación entre uno y otro (Larman, 2016).

A continuación, en la Figura 8 se muestra una parte del diagrama de clases del diseño, en la cual se encuentran las clases relacionadas con los RF 14, 15 y 16. El diagrama de clases del diseño con la totalidad de las clases se encuentra entre los artefactos entregables del trabajo de diploma.

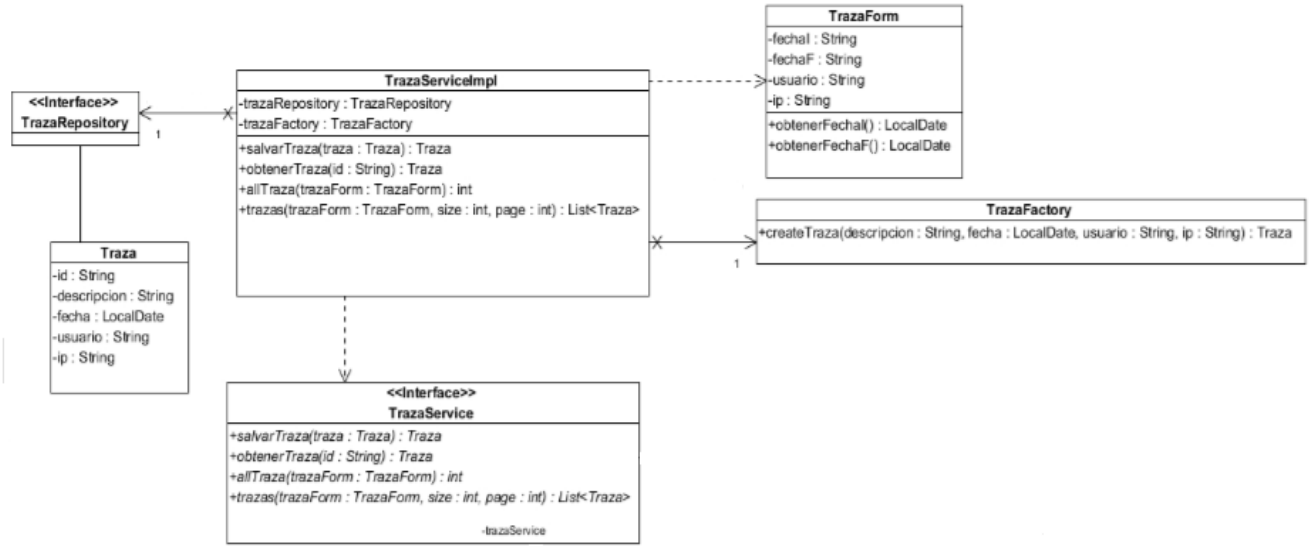


Figura 8. Diagrama de clases de diseño relacionado con los RF 14, 15 y 16
Fuente: elaboración propia

El diagrama de la Figura 8 está compuesto por las siguientes clases y relaciones:

- Clase Traza: es una clase de persistencia que representa los datos que contiene una traza, que presenta una relación de asociación bidireccional con TrazaRepository.
- Clase *TrazaForm*: representa los parámetros de búsqueda de una traza.
- Interfaz TrazaFactory: es la encargada de convertir un objeto de tipo *TrazaForm* en un objeto de tipo Traza, para así aliviar de responsabilidades a la clase *TrazaServiceImpl*.
- Interfaz *TrazaService*: define todas las abstracciones necesarias para la gestión de trazas.

- Interfaz *TrazaRepository*: es la interfaz encargada del acceso a datos, la cual permite la comunicación entre la clase *TrazaServiceImpl* del componente y la base de datos, la misma presenta una relación de asociación bidireccional con la clase Traza.
- Clase *TrazaServiceImpl*: implementa las abstracciones definidas en la clase *TrazaService* y permite la gestión de trazas, la misma presenta una relación de uso con la interfaz *TrazaService,* y la clase *TrazaForm*, además presenta una relación de asociación unidireccional con la interfaz *TrazaFactory* y la clase *TrazaRepository.*

2.4.4. Modelo de base de datos

La Figura 9 muestra el modelo de la base de datos relacional PostgreSQL del componente propuesto. El mismo contiene cada una de las tablas, así como sus atributos y relaciones, las cuales son de mucho a mucho, representados en un diagrama Entidad-Relación.

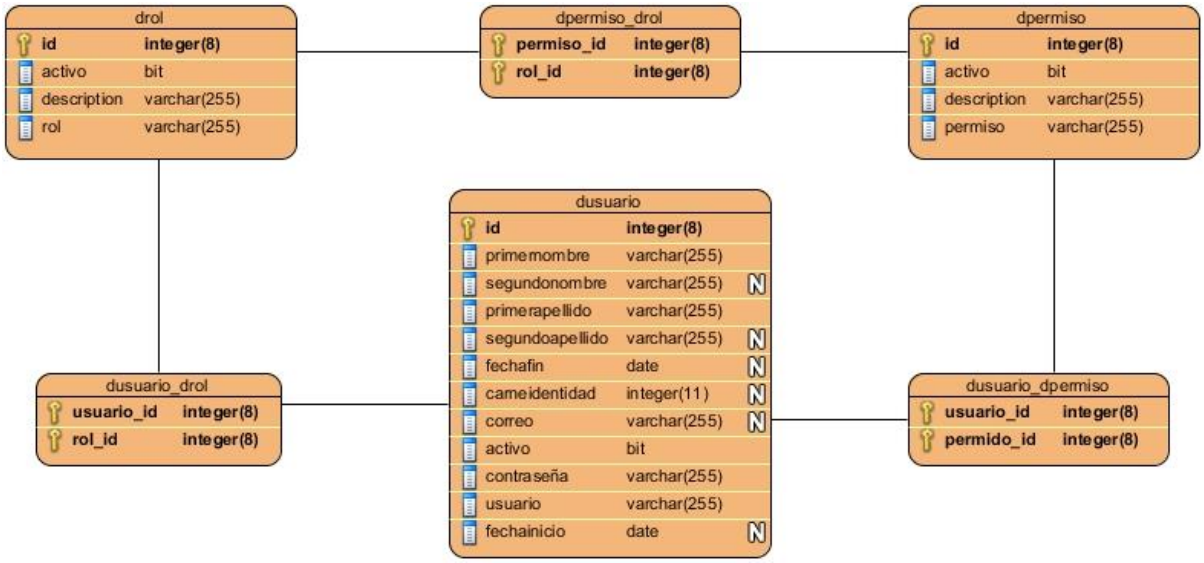


Figura 9. Modelo de datos de la solución propuesta
Fuente: elaboración propia

2.5. Disciplina implementación

En esta disciplina a partir de los resultados del Análisis y el diseño se construye la solución que da lugar a la presente investigación. En el proceso de implementación se tienen en cuenta un conjunto de estándares de codificación definidos por el equipo de arquitectos del departamento Desarrollo de Componentes de CEGEL.

2.5.1. Estándares de codificación

Los estándares de codificación constituyen buenas prácticas o conjunto de reglas no formales que han ido surgiendo en las comunidades de desarrolladores de software con el paso del tiempo. Estos tienen el

propósito de obtener un código fuente más legible, portable, seguro, eficiente, robusto y cohesionado, permitiendo mayor velocidad en el desarrollo, mejor coordinación entre los equipos de trabajo. Además logran que para un desarrollador sea más fácil integrarse a un proyecto ya comenzado, se eviten bugs[14] y se faciliten los procesos de mantenibilidad y revisión de código (Hommel, 2019). A continuación, se describen los estándares de codificación utilizados para el desarrollo del componente propuesto:

- Los nombres de las clases serán con mayúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma, ejemplo: *UsuarioService*.
- Los nombres de los métodos serán con minúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán con mayúscula, ejemplo: registrarRol.
- Los identificadores para las variables y los parámetros serán con letras en minúsculas y en caso de ser un nombre compuesto las siguientes palabras se escribirán con mayúscula, ejemplo: usuario y *usuarioDTO*.
- Los nombres de variables o funciones deben ser lo suficientemente descriptivos, sin exceder de 30 caracteres.
- El nombre de la clase controladora terminará con la palabra *Controller*, ejemplo: *GraphQLController*.
- Las interfaces de repositorio comenzarán nombrándose por el nombre de la entidad y seguido la palabra *Repository*, ejemplo: *RolRepository*.
- Las interfaces de los servicios comenzarán nombrándose por el nombre de la entidad y seguido la palabra *Service*, ejemplo: *RolService*.
- Las implementaciones de las interfaces de servicios comenzarán nombrándose por el nombre de la entidad y seguido la palabra *ServiceImpl*, ejemplo: *RolServiceImpl* y *UsuarioServiceImpl.*

2.6. Descripción del sistema

Una vez concluida la disciplina de Implementación se logra un componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas con cada una de las funcionalidades descritas en el epígrafe 2.3.2 del presente capítulo.

En la Figura 10 se muestra la funcionalidad Registrar usuario, a través de la cual se registran los datos de un usuario haciendo uso del componente propuesto. Por otra parte, la Figura 11 muestra una vista de la funcionalidad Visualizar traza. Es importante aclarar que para probar el componente una vez concluida su implementación, se utiliza la plantilla definida en el departamento Desarrollo de Componentes de CEGEL,

[14] Error de software que desencadena un resultado indeseado

para el Sistema de Gestión de Índice de Precios que se desarrolla a la Oficina Nacional de Estadística e Información.

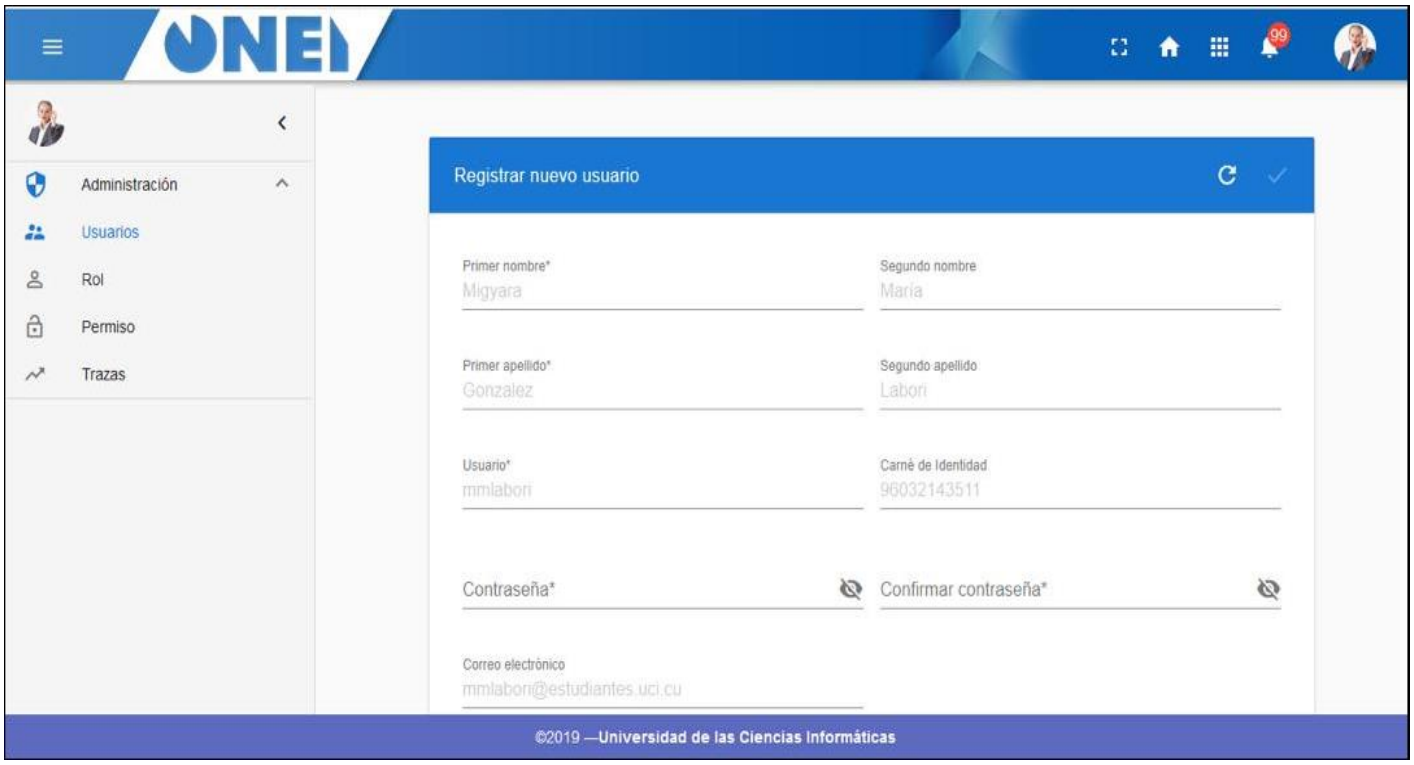


Figura 10. Funcionalidad Registrar usuario
Fuente: elaboración propia

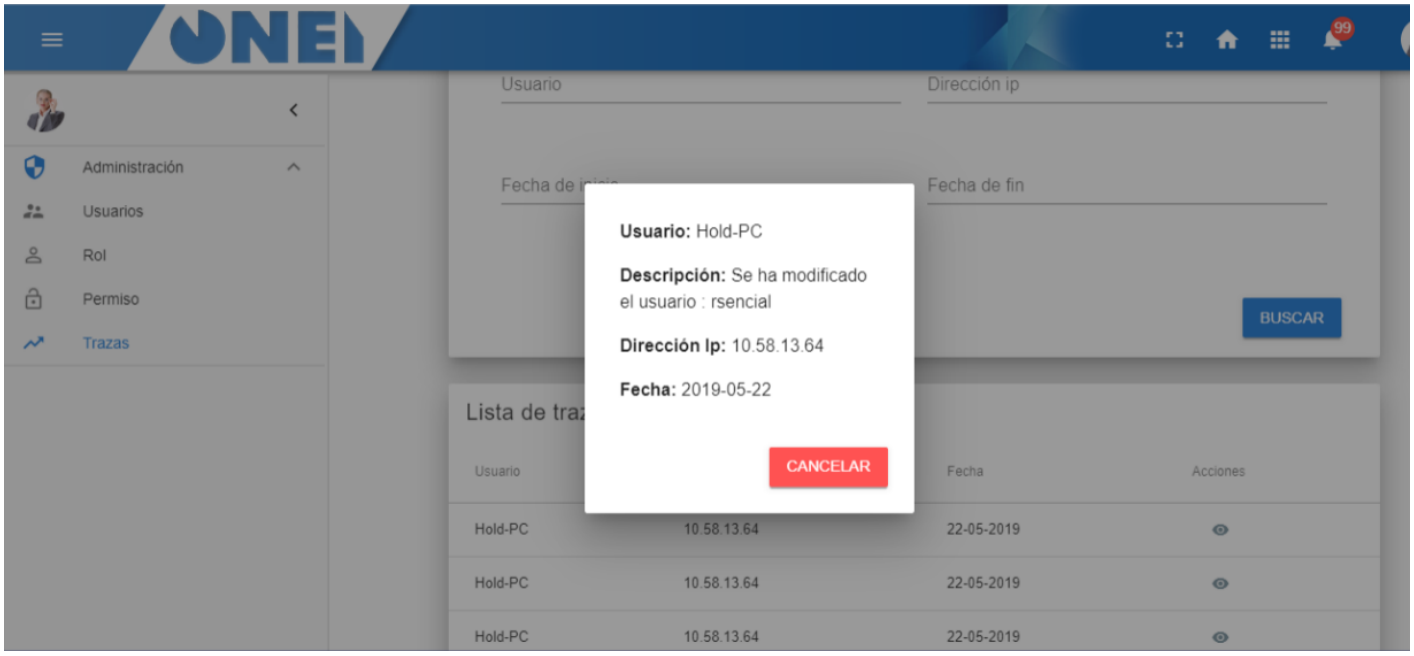


Figura 11. Funcionalidad Visualizar traza
Fuente: elaboración propia

2.7. Conclusiones parciales

El empleo de la metodología AUP en su variación para la UCI en función de describir el proceso de desarrollo del componente propuesto, permitió organizar el desarrollo de la solución y generar los artefactos necesarios. La aplicación de técnicas para la validación de los requisitos corroboró la obtención de requisitos no ambiguos que cumplen con las especificaciones del cliente. Por otra parte, la aplicación del patrón arquitectónico Vuex, para el manejo del estado de la aplicación, junto a la biblioteca *apollo-client* utilizada para el manejo de los datos en la arquitectura del cliente GraphQL y el patrón arquitectónico N-Capas, permitieron separar responsabilidades entre los elementos de la arquitectura, haciendo que el componente propuesto sea flexible al mantenimiento y a la introducción de cambios.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1. Introducción

En este capítulo se muestran los resultados obtenidos en la validación del diseño de la solución aplicando las métricas Tamaño Operacional de Clases y Relaciones entre Clases. Además, se establece una estrategia de prueba organizada en los niveles de unidad y aceptación, en los cuales se realizan pruebas funcionales y de usabilidad, aplicando métodos y técnicas. Todo el proceso de pruebas de software se organiza a partir de las disciplinas definidas por la metodología empleada. El capítulo concluye con la evaluación de la relación causa-efecto de la variable independiente sobre las variables dependientes de la investigación, a partir de la definición de un conjunto de indicadores que permiten chequear el comportamiento de esta relación antes y después del desarrollo del componente.

3.2. Validación del diseño

Con el objetivo de verificar la calidad del diseño se emplearon las métricas de diseño relaciones entre clases (RC) y tamaño operacional de clase (TOC).

3.2.1. Métrica Tamaño Operacional de Clase (TOC)

La métrica TOC fue aplicada a cada una de las clases del diseño con el objetivo de medir la calidad de las mismas con respecto a su grado de responsabilidad, complejidad de implementación y reutilización. A continuación, se describen los pasos que se llevaron a cabo para aplicar la métrica:

- Cálculo del umbral. El umbral se toma del tamaño general de una clase que se determina sumando todas las operaciones que posee el diseño.
- Calcular el promedio de los umbrales.
- Teniendo en cuenta los valores antes obtenidos, se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la Tabla 5.

Tabla 5. Rango de valores para medir la afectación de los atributos de calidad (TOC)

Atributos de calidad	Clasificación	Criterio
Responsabilidad	Baja	Umbral<= Promedio
	Media	Promedio < Umbral < = 2* Promedio
	Alta	Umbral > 2* promedio

Complejidad de implementación	Baja	Umbral <= Promedio
	Media	Promedio < Umbral < = 2* Promedio
	Alta	Umbral > 2* promedio
Reutilización	Baja	Umbral > 2* promedio
	Media	Promedio < Umbral < = 2* Promedio
	Alta	Umbral <= Promedio

Fuente: (Lorenz, 1994)

En la Figura 12 se muestran los resultados obtenidos luego de aplicar la métrica TOC sobre el diseño de clases del componente propuesto.

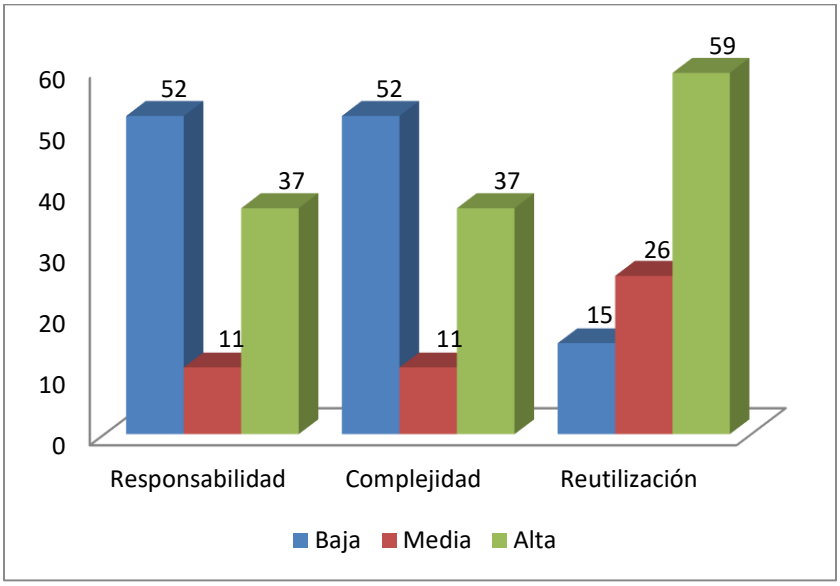


Figura 12. Representación en (%) de los resultados de la aplicación de la métrica TOC
Fuente: elaboración propia.

La figura anterior demuestra que con la aplicación de la métrica TOC se obtuvieron resultados positivos en relación a los siguientes atributos evaluados:

Responsabilidad: los resultados fueron satisfactorios, teniendo en cuenta que el 52 % de las clases tienen una responsabilidad baja.

Complejidad de implementación: los resultados fueron positivos, pues se demostró que el 52 % de las clases tienen una complejidad baja.

Reutilización: los resultados obtenidos fueron satisfactorios, demostrándose que el 59 % de las clases tienen una reutilización alta

Todo lo anterior facilita la obtención de una solución informática con poca dependencia entre clases, flexible al mantenimiento y a la introducción de cambios.

3.2.2. Métrica Relaciones entre clases (RC)

La métrica RC está dada por el número de relaciones de uso de una clase con otra. El primer paso en la aplicación de esta es evaluar los siguientes atributos de calidad: acoplamiento, complejidad de mantenimiento, reutilización de cada clase y la cantidad de pruebas que cada clase requiere (Pressman, 2010).

A continuación, se exponen los pasos que se llevaron a cabo para aplicar la métrica a todas las clases del componente propuesto en la presente investigación:

- Determinar la Cantidad de Relaciones de Uso (CRU) que poseen las clases a medir.
- Calcular el promedio de las CRU.
- Teniendo en cuenta los valores antes obtenidos se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la Tabla 6.

Tabla 6. Rango de valores para medir la afectación de los atributos de calidad (RC)

Atributos de calidad	Clasificación	Criterio
Acoplamiento	Ninguna	CRU=0
	Baja	CRU=1
	Media	CRU=2
	Alta	CRU>2
Complejidad de mantenimiento	Baja	CRU <= Promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU > 2* promedio
Reutilización	Baja	CRU > 2* promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU <= Promedio
Cantidad de pruebas	Baja	CRU <= Promedio

	Media	Promedio < CRU < = 2* promedio
	Alta	CRU > 2* promedio

Fuente: (Lorenz, 1994)

En la Figura 13 se muestran los resultados obtenidos luego de aplicar la métrica RC sobre el diseño de clases del componente propuesto.

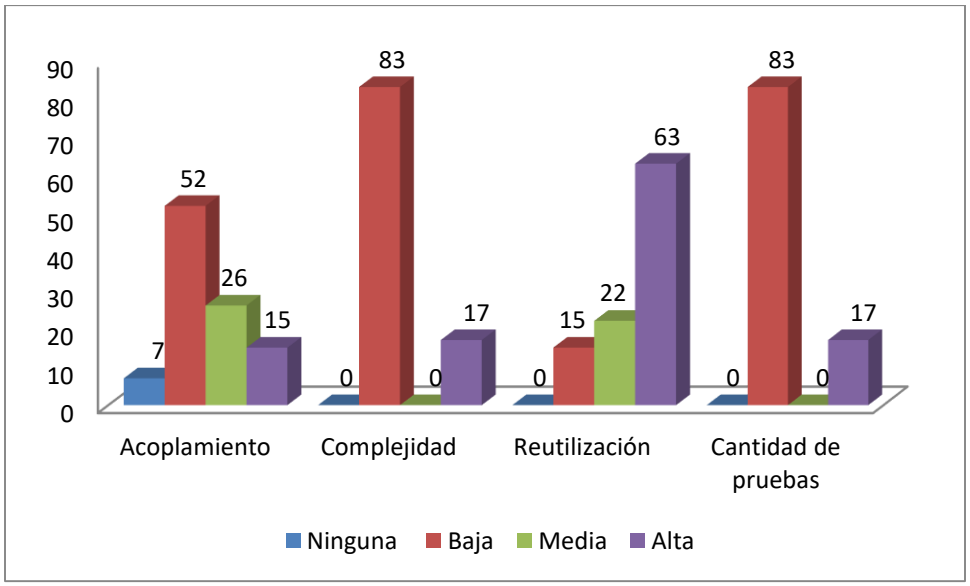


Figura 13. Representación en (%) de los resultados de la aplicación de la métrica RC
Fuente: elaboración propia.

La figura anterior demuestra que con la aplicación de la métrica RC se obtuvieron resultados positivos en relación a los siguientes atributos evaluados:

Acoplamiento: los resultados obtenidos son positivos teniendo en cuenta que el 52 % de las clases poseen un bajo acoplamiento.

Complejidad de mantenimiento: los resultados mostrados en la figura anterior, demuestran que el 83 % de las clases del componente propuesto presentan una complejidad de mantenimiento baja, facilitando así las futuras actividades de soporte sobre el mismo.

Reutilización: los resultados obtenidos fueron satisfactorios, demostrándose que el 63 % de las clases tienen una reutilización alta.

Cantidad de pruebas: los resultados demuestran que el 83 % de las clases poseen un bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones o pruebas sobre ellas.

Con los resultados obtenidos una vez aplicada la métrica RC se concluye que el diseño del componente propuesto tiene una baja complejidad de mantenimiento y acoplamiento entre sus clases. Además, se requiere de un bajo grado de esfuerzos para realizar cambios sobre la mayoría de las clases y estás a su vez presentan un elevado por ciento de reutilización. Todo lo anterior facilita la obtención de una solución informática escalable, con poca dependencia entre clases, flexible al mantenimiento y a la introducción de cambios.

3.3. Pruebas de Software

Las pruebas de software son un conjunto de actividades que pueden ser planificadas con antelación y ejecutarse sistemáticamente durante la implementación o al finalizar el desarrollo del software. Estas comprenden un conjunto de actividades que se realizan para identificar posibles fallos de funcionamiento, configuración o usabilidad de un programa o aplicación. Las pruebas de software se ejecutan a partir de la aplicación de métodos y técnicas. La organización del proceso de pruebas se realiza a través de cuatro niveles: unidad, integración, sistema y aceptación (Pressman, 2010).

Para la validación del componente propuesto en la presente investigación se define una estrategia de prueba de software a partir de las disciplinas establecidas por la metodología AUP variación UCI. En la estrategia se define que en las disciplinas de pruebas internas y pruebas de liberación solo se valide a nivel de unidad y en la disciplina de pruebas de aceptación se utiliza el nivel de igual nombre, en todos los casos con la aplicación de los respectivos métodos y técnicas. Los niveles de integración y sistema no se aplican, teniendo en cuenta que el alcance de la tesis solo comprende el desarrollo del componente, sin llegar a su integración con alguna aplicación.

3.3.1. Pruebas internas

Para la validación del componente propuesto las pruebas internas se ejecutan a nivel de unidad. A continuación, se detalla cómo se realizó el desarrollo de las mismas.

Pruebas a nivel de unidad

Las pruebas a nivel de unidad se concentran en el esfuerzo de verificación de la unidad más pequeña del diseño, el componente o módulo de software. Tomando como guía la descripción del diseño a nivel de componente, se prueban importantes caminos de control para describir errores dentro de los límites del módulo. El alcance restringido que se ha determinado para las pruebas de unidad limita la relativa complejidad de las pruebas y los errores que éstas descubren (Pressman, 2010). En el caso de la presente tesis en este nivel de prueba se aplicaron los métodos de caja blanca y caja negra.

Métodos de caja blanca:

El método de caja blanca posibilita el desarrollo de casos de prueba que garanticen la ejecución, al menos una vez, de los caminos independientes (Pressman, 2010). En el caso del presente trabajo de diploma el método fue ejecutado aplicando la técnica de ruta básica.

La técnica de ruta básica tiene como objetivo comprobar que cada camino se ejecute independiente de un componente o programa, obteniéndose una medida de la complejidad lógica del diseño. Esta técnica debe ser utilizada para evaluar la efectividad de los métodos asociados a una clase, con el objetivo de asegurar que cada camino independiente sea ejecutado por lo menos una vez en el sistema (Pressman, 2010).

En la validación del componente propuesto la técnica de ruta básica se aplicó a todos los métodos de las clases controladoras, teniendo en cuenta que estas agrupan las principales funcionalidades de la solución. A continuación, se describe la aplicación del método de caja blanca sobre la funcionalidad *registrarRol*, perteneciente a la clase *RolServiceImpl* (ver Figura 14). Se toma como muestra esta funcionalidad teniendo en cuenta que es una de las de mayor prioridad y responde a uno de los principales requisitos funcionales de la solución.

```
public Rol registrarRol(@GraphQLArgument(name = "rol") RolDTO rolDTO){
    if (rolDTO.getId() == null) {                                    1
        if (rolDTO.getRol() != null &&          2
!rolRepository.findByRol(rolDTO.getRol()).isPresent()) {    3
            Rol save =
rolRepository.save(rolFactory.createRol(rolDTO));    4
            notifier.publish(TiposTraza.GENERIC_REGISTRAR, "el rol : "
+ save.getRol());
            return save;    5
        } else {
            throw new ExceptionPitcher("El rol que intenta registrar
ya existe");    6
        }
    } else {
        Rol rol. = rolFactory.createRol(rolDTO);    7
        rol.setId(rolDTO.getId());

        Rol save = rolRepository.save(rol);
        notifier.publish(TiposTraza.GENERIC_MODIFICAR, "el rol : " +
save.getRol());
        return save;    8
    }

}    9
```

Figura 14. Funcionalidad *registrarRol*
Fuente: elaboración propia

Una vez definido el código sobre el cual se aplica el método, los pasos a seguir para desarrollar la técnica de ruta básica son los siguientes:

1) Confeccionar el grafo de flujo: este muestra el flujo de control lógico (Pressman, 2010). Está compuesto por los siguientes elementos:

- Nodos: son círculos que representan una o más sentencias procedimentales.
- Aristas: son flechas que representan el flujo de control y son análogas a las flechas del diagrama de flujo.
- Regiones: son las áreas delimitadas por aristas y nodos.

En la Figura 15 se muestra el grafo de flujo obtenido con la ejecución del método de caja blanca sobre la funcionalidad *registrarRol*.

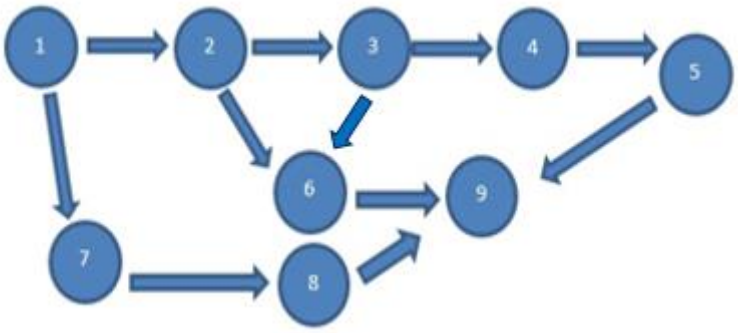


Figura 15. Grafo de camino básico del método *registrarRol*
Fuente: elaboración propia

2) Calcular la complejidad ciclomática:

El valor calculado por la complejidad ciclomática define el número de rutas independientes del conjunto básico de un programa y brinda una cota superior para el número de pruebas que se deben realizar a fin de asegurar que todos los enunciados se ejecutaron al menos una vez (Pressman, 2010).

La complejidad ciclomática se calcula de tres formas diferentes, las cuales deben llegar al mismo resultado para comprobar que el cálculo es el correcto (Pressman, 2010).

- El número de regiones del grafo de flujo corresponde a la complejidad ciclomática.
- La complejidad ciclomática V(G) para un grafo de flujo G se define como:
  V(G)=E-N+2
  Donde E es el número de aristas del gráfico de flujo y N el número de nodos del gráfico de flujo.
- La complejidad ciclomática V(G) para un grafo de flujo G también se define como
  V(G)=P+1

Donde P es el número de nodos predicado (nodos de donde parten al menos dos aristas) contenidos en el grafo de flujo G.

En el grafo de flujo de la Figura 15, la complejidad ciclomática puede calcularse usando cada una de las vías anteriormente descritas:

1. El grafo de flujo tiene 4 regiones, por tanto, V(G) = 4
2. V(G) = (11 aristas - 9 nodos) +2 = 4
3. V(G) = 3 nodos predicado + 1= 4

Por tanto, la complejidad ciclomática del grafo de flujo de la Figura 15 es 4.

3) Determinar un conjunto básico de caminos linealmente independientes:

El valor de V(G) proporciona la cota superior sobre el número de rutas linealmente independientes a través de la estructura de control del programa (Pressman, 2010). En el caso de la funcionalidad *registrarRol*, se definen 4 caminos básicos:

Camino básico # 1: 1, 2, 3, 4, 5, 9
Camino básico # 2: 1, 2, 6, 9
Camino básico # 3: 1, 2, 3, 6, 9
Camino básico # 4: 1, 7, 8, 9

4) Obtención de casos de prueba (CP):

Una vez definidos los caminos, se procede a diseñar los casos de prueba para cada uno de los caminos básicos obtenidos. A continuación, en las Tablas 7, 8, 9 y 10 se presentan los casos de prueba definidos para los caminos obtenidos con la técnica ruta básica.

Tabla 7. Caso de prueba del camino básico 1

Caso de prueba para la ruta básica # 1	
Descripción: el método recibe como parámetro un rol y se verifica si tiene id. En caso positivo se verifica si este parámetro no tiene un rol y además no se encuentre registrado en la base de datos. De cumplirse estas condiciones se procede a registrar el rol, se notifica su registro y por último se retorna el rol registrado.	
Entrada:	Un *rolDTO*
Resultados esperados	Un rol registrado

Condiciones	El id y el rol no deben ser nulos. Además el rol no debe estar registrado en la base de datos.

Fuente: elaboración propia

Tabla 8. Caso de prueba del camino básico 2

Caso de prueba para la ruta básica # 2	
Descripción: el método recibe como parámetro un rol y se verifica si tiene id. En caso positivo se verifica si este parámetro tiene un rol. De cumplirse esta condición, el sistema lanza una excepción indicando que el rol que intenta registrar ya existe.	
Entrada:	Un *rolDTO*
Resultados esperados	Una excepción indicando que el rol que intenta registrar ya existe.
Condiciones	El id y el rol no deben ser nulos.

Fuente: elaboración propia

Tabla 9. Caso de prueba del camino básico 3

Caso de prueba para la ruta básica # 3	
Descripción: el método recibe como parámetro un rol y se verifica si tiene id. En caso positivo verifica si este parámetro tiene un rol y si este rol se encuentra registrado en la base de datos. De cumplirse estas condiciones el sistema lanza una excepción indicando que el rol que intenta registrar ya existe.	
Entrada:	Un *rolDTO*
Resultados esperados	Una excepción indicando que el rol que intenta registrar ya existe.
Condiciones	El id y el rol exista. Y además el rol debe estar registrado en la base de datos.

Fuente: elaboración propia

Tabla 10. Caso de prueba del camino básico 4

Caso de prueba para la ruta básica # 4	
Descripción: el método recibe como parámetro un rol y se verifica si tiene id. En caso negativo se crea un nuevo rol, se notifica su registro y se devuelve el rol registrado.	
Entrada:	Un *rolDTO*
Resultados esperados	Un rol registrado.
Condiciones	El id del rol no debe ser nulo.

Fuente: elaboración propia

Una vez ejecutados todos los casos de pruebas obtenidos con la técnica empleada, se concluye que los mismos fueron probados satisfactoriamente, corrigiéndose los hallazgos surgidos en una primera iteración y comprobándose su corrección en una segunda. Al concluir la prueba se demuestra que todas las funcionalidades del componente se ejecutan satisfactoriamente, quedando libres de código repetido o innecesario.

Métodos de caja negra:

El método de caja negra se centra en los requisitos funcionales del software. Es decir, permite al ingeniero de software derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales de un programa. El método de caja negra no es una opción frente a caja blanca. Es, en cambio, un enfoque complementario que tiene probabilidades de describir una clase diferente de errores de los que se identifican con los métodos de caja blanca (Pressman, 2010)

El método de caja negra se ejecuta a partir del desarrollo de pruebas funcionales, con la intención de identificar errores en las siguientes categorías (Pressman, 2010)

- Funciones incorrectas o faltantes.
- Errores de interfaz.
- Errores en estructuras de datos o en acceso a bases de datos externas.
- Errores de comportamiento o desempeño.
- Errores de inicialización y término.

Para desarrollar el método de caja negra se encuentra el uso de las técnicas (Pressman, 2010):

- Partición de equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Análisis de valores límites: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

Con el propósito de evaluar a nivel de equipo de desarrollo el correcto funcionamiento del componente propuesto, se realizaron pruebas funcionales a nivel de unidad que permitieron comprobar que el componente responde a cada uno de los RF sobre los cuales fue implementado. El desarrollo de estas pruebas se realizó aplicando el método de caja negra con el uso de las técnicas partición de equivalencia y análisis de valores límites. Como herramienta de apoyo se utilizaron las DCP, en el Tabla 3 se puede consultar la DCP correspondiente al RF: Registrar rol. El resto de las DCP generadas para la validación del componente se encuentran entre los artefactos entregables de la tesis. A continuación, en la Figura 16 se muestran los

resultados de las pruebas funcionales a nivel de unidad realizadas por el equipo de desarrollo del componente propuesto.

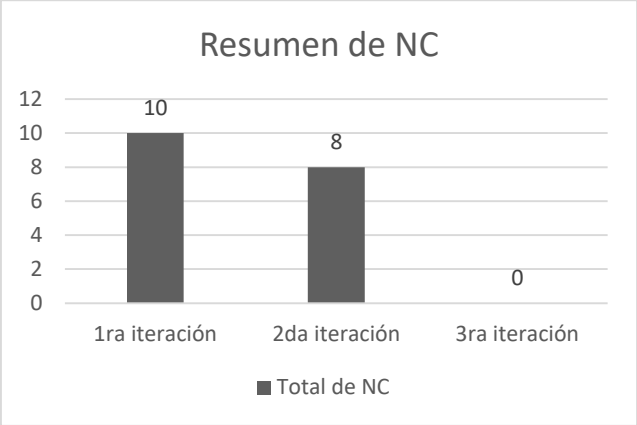


Figura 16. No conformidades detectadas al aplicar el método de caja negra por el equipo de desarrollo
Fuente: elaboración propia

Como muestra la Figura 16, en la primera iteración se detectaron un total de 10 No Conformidades (NC), clasificadas en 4 de ortografía, 3 de validación, 3 de funcionalidad. Luego en una segunda iteración persistieron 2 de ortografía y 1 de funcionalidad, apareciendo 2 de consistencia y estándares y 3 de opciones que no funcionan, teniendo en cuenta que en la segunda iteración también se realizaron pruebas de usabilidad utilizando como herramienta de apoyo la lista de chequeo que utiliza el equipo de calidad de CEGEL. En la tercera iteración los resultados fueron satisfactorios, obteniéndose cero NC. Este resultado demuestra que las funcionalidades del componente cumplen con cada uno de los RF.

3.3.2. Pruebas de liberación

Con el propósito de validar el correcto funcionamiento del componente propuesto se realizaron pruebas de libración de tipo funcionales y de usabilidad, utilizando el método de caja negra con el empleo de las técnicas partición de equivalencia y análisis de valores límites, aplicando las mismas DCP usadas en las pruebas internas. Estas pruebas fueron realizadas por el equipo de calidad de CEGEL. A continuación, en la figura 17 se muestran los resultados obtenidos con el desarrollo de estas pruebas.

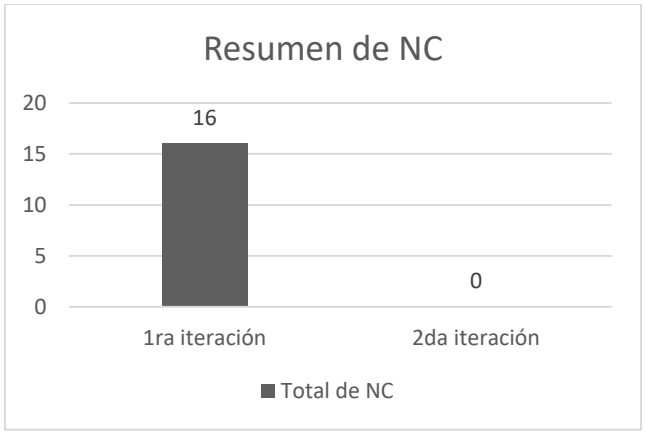
4. 

Figura 17. No conformidades detectadas en las pruebas de liberación
Fuente: elaboración propia

Como muestra la Figura 17, en la primera iteración se detectaron un total de 16 No Conformidades (NC), clasificadas en 2 de ortografía, 2 validación, 3 de estética y diseño, 5 de correspondencia con otro artefacto y 4 de funcionalidades. Luego en la segunda iteración los resultados fueron satisfactorios, obteniéndose cero NC, generándose así el Acta de Liberación por parte de la Asesora de Calidad de CEGEL (ver Anexo 3).

3.3.3. Pruebas de aceptación

Al componente propuesto se le realizaron pruebas a nivel de aceptación, con el propósito de verificar que el software está listo y cumple con cada una de las funcionalidades definidas con el cliente durante la etapa de levantamiento de requisitos. Estas pruebas fueron ejecutadas por arquitectos del departamento Desarrollo de Componentes de CEGEL y los tutores de la presente investigación, quienes también ejercen como clientes. Como resultado de estas pruebas se generó un Acta de Aceptación validada por dos de los arquitectos principales del área donde se desarrolló el trabajo de diploma (Ver Anexo 4).

3.4. Validación de los resultados de la investigación

Teniendo en cuenta que en la presente investigación se define como idea a defender: “el desarrollo de un componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas reduce las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor”, para analizar la relación causa efecto de la variable independiente “un componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas” sobre las variables dependientes “reduce las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor”. Las autoras de la presente investigación, definen criterios de medida que permiten certificar cómo a través del componente propuesto se logra la relación entre ambas variables. La elección de estos criterios se realiza, a partir de las principales deficiencias descritas en la situación problemática que da lugar al desarrollo de la investigación.

Criterios de medida definidos:

- Peticiones múltiples sobre recursos relacionados: este criterio determina la cantidad de llamadas que se hacen necesarias entre el cliente y el servidor, cuando los recursos tienen varias relaciones entre ellos.
- Carga de datos innecesarios: este criterio mide la carga de datos innecesarios que puede incurrir en cada petición.

A continuación, en la Tabla 11 se muestran los resultados de evaluar los criterios de medida antes definidos. La valoración se realiza a través de un antes (componente implementado utilizando REST) y un después (componente implementado utilizando GraphQL), con el propósito de verificar cómo mediante la solución propuesta se reducen las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor. Los datos tenidos en cuenta para evaluar los criterios fueron tomados a partir de la realización de una comparación entre componente propuesto y el componente para la administración de usuarios, roles, permisos y trazas que utiliza el Sistema de Gestión para la Atención a la Población implementado en el departamento Desarrollo de Componentes de CEGEL.

Tabla 11. Evaluación de los criterios de medidas definidos

Criterios de medida	Antes	Después
Peticiones múltiples sobre recursos relacionados.	Cuantas más relaciones tenga un recurso, más lenta es la carga del mismo, debido a que mientras el grado de relación de los recursos aumente se hace necesario realizar varias peticiones para obtener los mismos de forma íntegra. Ejemplo: en el editar usuario para obtener el usuario en cuestión se hace necesario hacer 3 peticiones, una para obtener el usuario, otra para obtener sus roles y una tercera para obtener los permisos asignados a este.	Se puede obtener un recurso fuertemente relacionado en una sola llamada. Además, se pueden solicitar los atributos necesarios para esa petición. Ejemplo: al editar un usuario se realiza una única petición para obtener de forma íntegra los datos necesarios.
	REST presenta la característica de obtener datos innecesarios cada vez	GraphQL permite solicitar únicamente los campos necesarios, provocando

Carga de datos innecesarios.	que se le hace una petición. En muchas ocasiones no se necesita toda la información y se acaba ignorando muchos de los datos. Este problema hace que se consuma más ancho de banda y la carga de los datos sea más lenta. Ejemplo: en la funcionalidad listar usuario se deben mostrar los siguientes datos: usuario, nombre completo, fecha de inicio y activo. Además, es necesario conocer el identificador único del usuario. Sin embargo, la API presenta una estructura fija, lo cual tiene como consecuencia que al devolver los datos relacionados con la funcionalidad listar usuario, se carguen datos innecesarios.	el mínimo de coste posible en la utilización del ancho de banda y asegurando no cargar datos innecesarios. Ejemplo: en el listar usuario se muestran los datos: usuario, nombre completo, fecha de inicio y activo, también es necesario el identificador único del usuario. Teniendo en cuenta que GraphQL es un lenguaje de consulta de datos, se realiza la consulta y este devuelve solo los datos necesarios.

Fuente: elaboración propia

La comparación realizada en la tabla anterior, a través de los criterios de medida antes definidos, demuestra cómo utilizando el componente propuesto en la presente investigación, se reducen las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor.

3.5. Conclusiones parciales

La aplicación de técnicas para la validación del diseño certifica la obtención de un componente flexible al mantenimiento y a la introducción de cambios. De igual forma, la realización de pruebas internas y de liberación en el nivel de unidad, con el empleo de sus respectivos métodos y técnicas, certifican que el componente obtenido cumple con cada uno de los RF que dieron lugar a su implementación, avalado por el acta de liberación emitida por el grupo de calidad de CEGEL. Por otra parte, la realización de pruebas de aceptación con arquitectos de CEGEL, demuestran que el componente cumple con las necesidades del cliente definidas durante la identificación de los requisitos. La evaluación de indicadores a través de un antes y un después demuestran el cumplimiento de la relación causa efecto de la variable independiente “un componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas”, sobre las

variables dependientes “reduce las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor”.

CONCLUSIONES GENERALES

- El estudio de los referentes teóricos relacionados a las tecnologías REST y GraphQL, demostró que GraphQL tiene como ventaja la posibilidad de obtener datos necesarios sin tener que realizar múltiples llamadas secuenciales o en paralelas, característica que permitió a las autoras de la presente investigación adoptar la decisión de utilizar GraphQL como lenguaje de consulta en el desarrollo del componente propuesto.

- El proceso de identificación y validación de requisitos permitió definir las funcionalidades del componente propuesto. El cual presenta un diseño arquitectónico basado en el uso de los patrones Vuex y N-Capas, así como el empleo de estándares de codificación durante proceso de implementación, elementos que contribuyen la obtención de una solución informática mantenible y escalable.

- El desarrollo de pruebas de software en los niveles de unidad y aceptación, organizadas por las disciplinas que estable la metodología AUP variación UCI para la etapa de pruebas, así como los resultados arrojados por las métricas de validación de diseño de software empleadas, corroboran la validez del diseño e implementación del componente propuesto.

- Con la verificación de la relación causa efecto de la variable independiente sobre las variables dependientes de la investigación, se demuestra que con el desarrollo del componente propuesto se reducirán las peticiones múltiples sobre objetos relacionados y la carga de datos innecesarios durante la obtención de recursos desde el servidor.

RECOMENDACIONES

- Integrar el componente propuesto a las nuevas soluciones informáticas que se desarrollen en CEGEL.

- Implementar el cliente basado en GraphQL en otros marcos de trabajo como Angular.

BIBLIOGRAFÍA REFERENCIADA

Alegsa. 2018. [En línea] 2018. http://www.alegsa.com.ar/Dic/framework.php.

Briging the semantic web and web2.0 winth representational state transfer (REST). Battle, R & Benson. 2008.

Carlos. 2014. What is Maven. [En línea] 2014. Benson

Chirichigno, Agustín. 2018. *Predicción de dependencias en paquetes de NPM.* 2018.

Conceptos, Enciclopedia de. 2018. Concepto.de. *Concepto.de.* [En línea] 2018. https://concepto.de/http/.

Diego Lazaro . 2018. Introducción a los Web Services. *Introducción a los Web Services.* [En línea] 2018. https://diego.com.es/introduccion-a-los-web-services.

Enríquez Ruiz, José Luis, Farías Palacín, Elías y Flores Flores, Eder. 2017. *Metodología de desarrollo de software.* Rectorado, Universidad Católica Los Ángeles - Chimbote. Chimbote - Perú : s.n., 2017. pág. 39.

Escalante, Lain Cardenas. 2014. *El patron de arquitecturas n-capas con orientacion al dominio.* 2014.

Eugenia, Bahit. 2011. *El paradigma de la Programación.* 2011.

Facebook Corporation. 2019. Flux. [En línea] 2019. https://facebook.github.io/flux/docs/in-depth-overview.html.

Fielding, Roy. 2000. *Especificacion HTTP.* 2000.

Hoc, Solving Ad. 2016. solvingadhoc. [En línea] 2016. [Citado el: 23 de 5 de 2019.] https://solvingadhoc.com/las-historias-usuario-funcion-agilidad/.

Hommel, Scott. 2019. *Estandares_de_codificacion_para_Java.* 2019.

Jetbrains. 2018. [En línea] 2018. https://www.jetbrains.com/webstorm/.

JetBrains. 2018. [En línea] 2018. https://www.jetbrains.com/idea/.

Kristina. 2013. *MongoDB: the definitive guide, powerfull and scalable data storage.* s.l. : OReily Media. Inc, 2013.

Larman, Craig. 2016. *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 3ra . s.l. : Prentice-Hall, 2016.

Lorenz, M., & Kidd, J. 1994. *Object-oriented software metrics: a practical guide.* New Jersey, Prentice-Hall : s.n., 1994.

Martin Olivera, Yonnys Pablo y Zamora Sánchez, Gey. 2016. *ENTORNO DE DESARROLLO INTEGRADO LIBRE Y MULTIPLATAFORMA PARA DESARROLLAR SOFTWARE EDUCATIVO EN FORMATO MULTIMEDIA.* 2016.

Menéndez-Barzanallana Asensi, Rafael. 2016. *Ingeniería del software. .* 2016.

Microbuffer. 2011. [En línea] 2011. https://microbuffer.wordpress.com/2011/05/04/que-es-.

Muñoz Serafin, Miguel. 2018. *Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft.* 2018.

Pérez, Felipe U. 2018. La Revista Informática.com. *La Revista Informática.com.* [En línea] 2018. http://www.larevistainformatica.com/Java.htm.

Perry, J Steven. 2017. [En línea] 11 de 05 de 2017. http://www.ibm.com.

Pressman, Roger S, Ph.D. 2010. *Ingenieria de Software. Un enfoque práctico. Septima Edición.* Mexico, D.F : The Me Graw Hill Companies, 2010.

Pressman, Roger S. 2010. *Ingeniería de Software. Un enfoque práctico.* Septima . México DF : McGraw-Hill INTERAMERICA EDITORES, 2010.

Ramírez, T. 1999. *Proyecto de Investigación. .* Caracas: Panapo : s.n., 1999.

Redux. 2019. [En línea] 2019. https://redux.js.org/.

Rodríguez, T. 2015. *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana : s.n., 2015.

Rodríguez, Txema. 2017. GENBETA. *GENBETA.* [En línea] diciembre de 2017. https://www.genbeta.com/desarrollo/por-que-deberiamos-abandonar-rest-y-empezar-a-usar-graphql-en-nuestras-apis.

Rojas, M. J. 2010. *Patrones de Diseño.* 2010.

Sánchez, Tamara Rodriguez. 2015. *Metodologia UCI .* 2015.

Sommerville. 2011. *Ingeniería de Software, 9na Edición.* 2011.

The PostgreSQL Global Development Group. 2019. PostgreSQL. *PostgreSQL.* [En línea] 2019. https://www.postgresql.org/docs/10/release-10-1.html.

Vargas, Juan Gabriel Jimenéz. 2017. *Informe para optar por el grado de Bachiller en Ingenieria de Computacion.* 2017.

Venemedia. 2014. [En línea] 2014. http://conceptodefinicion.de/javascript/.

Victor S, Wendi L. 2018. *Self-documentation for representational state transfer (REST) application programming interface (API).* 2018.

VueJs. 2019. [En línea] 2019. https://vuejs.org/.

Vuex. 2019. [En línea] 2019. https://vuex.vuejs.org/.

Zamitiz, Ing. Carlos Alberto Román. 2016. Java Basico. *Java Basico.* [En línea] 2016.

ANEXOS

Anexo 1: Entrevista

Guía de preguntas utilizadas en el desarrollo de la entrevista con los profesionales de la ONEI.

1. ¿Cómo se realiza actualmente el proceso de administración de usuarios, roles, permisos y trazas en los sistemas del departamento Desarrollo de Componentes de CEGEL?

2. ¿Cómo cree usted que se puede mejorar el proceso?

3. Mencione los aspectos fundamentales que para usted debe poseer un usuario en el sistema. (Se refiere a los atributos que caractericen un usuario)

4. Mencione los aspectos fundamentales que para usted debe poseer un rol en el sistema. (Se refiere a los atributos que caractericen un rol)

5. Mencione los aspectos fundamentales que para usted debe poseer un permiso en el sistema. (Se refiere a los atributos que caractericen un permiso)

6. Mencione los aspectos fundamentales que para usted debe poseer una traza en el sistema. (Se refiere a los atributos que caractericen una traza)

7. Le interesa a usted el uso de un componente que reduzca las peticiones múltiples sobre recursos relacionados y la carga de datos innecesarios en la administración de usuarios, roles, permisos y trazas.

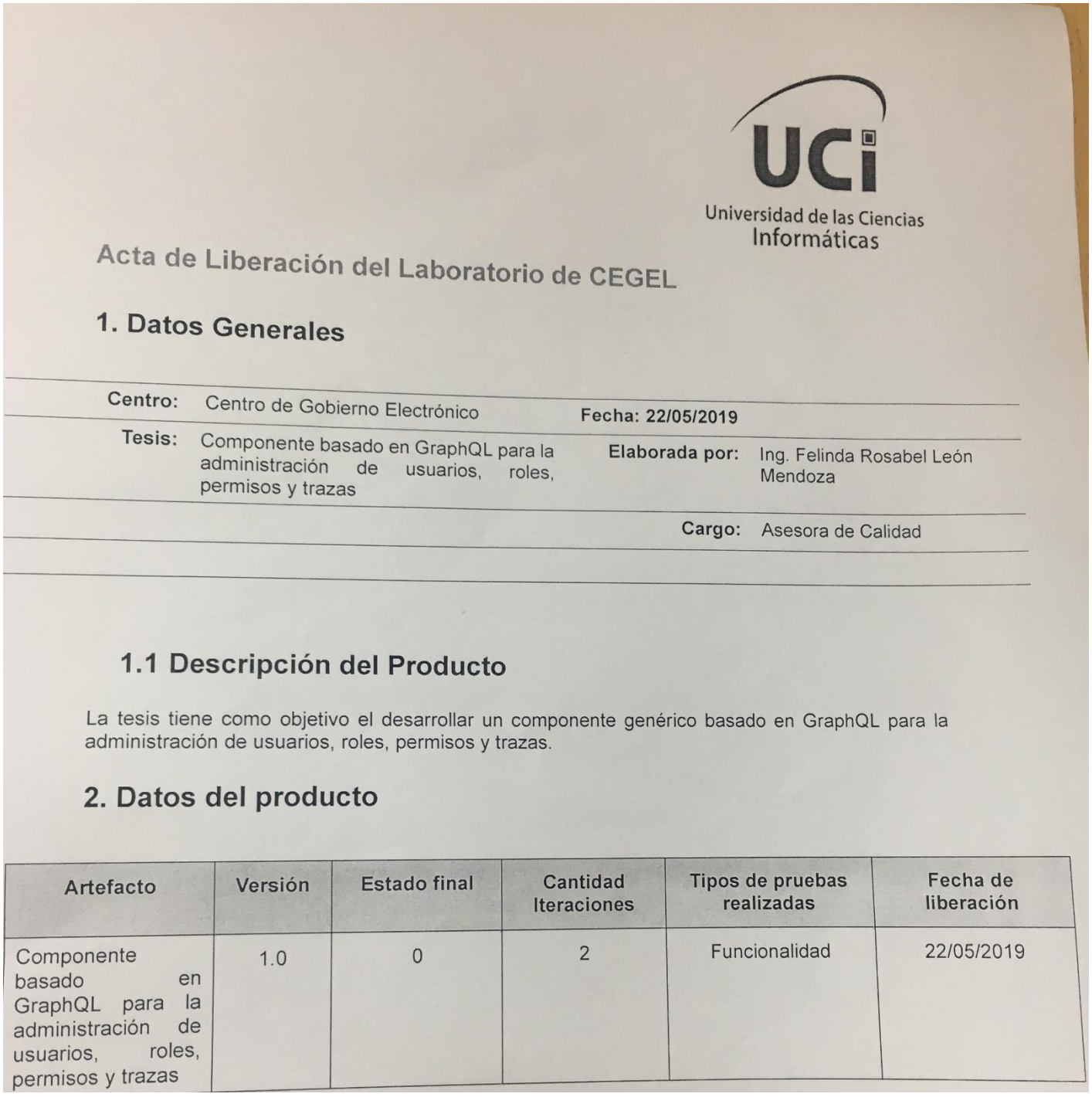
Anexo 2: Acuerdos tomados en las tormentas de ideas con el cliente

Tabla 12. Acuerdos tomados en las tormentas de ideas con el cliente

Nº	Acuerdo	Responsables	Fecha de cumplimiento
1	El componente debe gestionar usuarios, roles, permisos y trazas.	Dayanis Pérez Pérez Migyara M. González Labori	05/04/2019
2	Se debe implementar la API GraphQL para gestionar los usuarios.	Dayanis Pérez Pérez	12/04/2019
3	Se debe implementar el cliente GraphQL para gestionar los usuarios.	Migyara M. González Labori	12/04/2019
4	Se debe implementar la API GraphQL para gestionar las trazas.	Dayanis Pérez Pérez	26/04/2019
5	Se debe implementar el cliente GraphQL para gestionar las trazas.	Migyara M. González Labori	26/04/2019
6	Se debe implementar la API GraphQL para gestionar los permisos.	Dayanis Pérez Pérez	03/05/2019
7	Se debe implementar el cliente GraphQL para gestionar los permisos.	Migyara M. González Labori	03/05/2019
8	Se debe implementar la API GraphQL para gestionar los roles.	Dayanis Pérez Pérez	10/05/2019
9	Se debe implementar el GraphQL para gestionar los roles.	Migyara M. González Labori	10/05/2019

Fuente: elaboración propia

Anexo 3: Acta de liberación por el grupo de calidad de CEGEL



Universidad de las Ciencias Informáticas

Acta de Liberación del Laboratorio de CEGEL

1. Datos Generales

Centro:	Centro de Gobierno Electrónico	**Fecha: 22/05/2019**	
Tesis:	Componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas	**Elaborada por:**	Ing. Felinda Rosabel León Mendoza
		Cargo:	Asesora de Calidad

1.1 Descripción del Producto

La tesis tiene como objetivo el desarrollar un componente genérico basado en GraphQL para la administración de usuarios, roles, permisos y trazas.

2. Datos del producto

Artefacto	Versión	Estado final	Cantidad Iteraciones	Tipos de pruebas realizadas	Fecha de liberación
Componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas	1.0	0	2	Funcionalidad	22/05/2019

Figura 18. Acta de liberación por el grupo de calidad de CEGEL (Parte 1)
Fuente: elaboración propia

Universidad de las Ciencias Informáticas

3. Participantes en las pruebas de liberación.

Nombre y Apellidos	Rol que desempeña
Felinda Rosabel León Mendoza	Asesora de Calidad/ Probador

Felinda Rosabel León Mendoza
Asesora de Calidad CEGEL

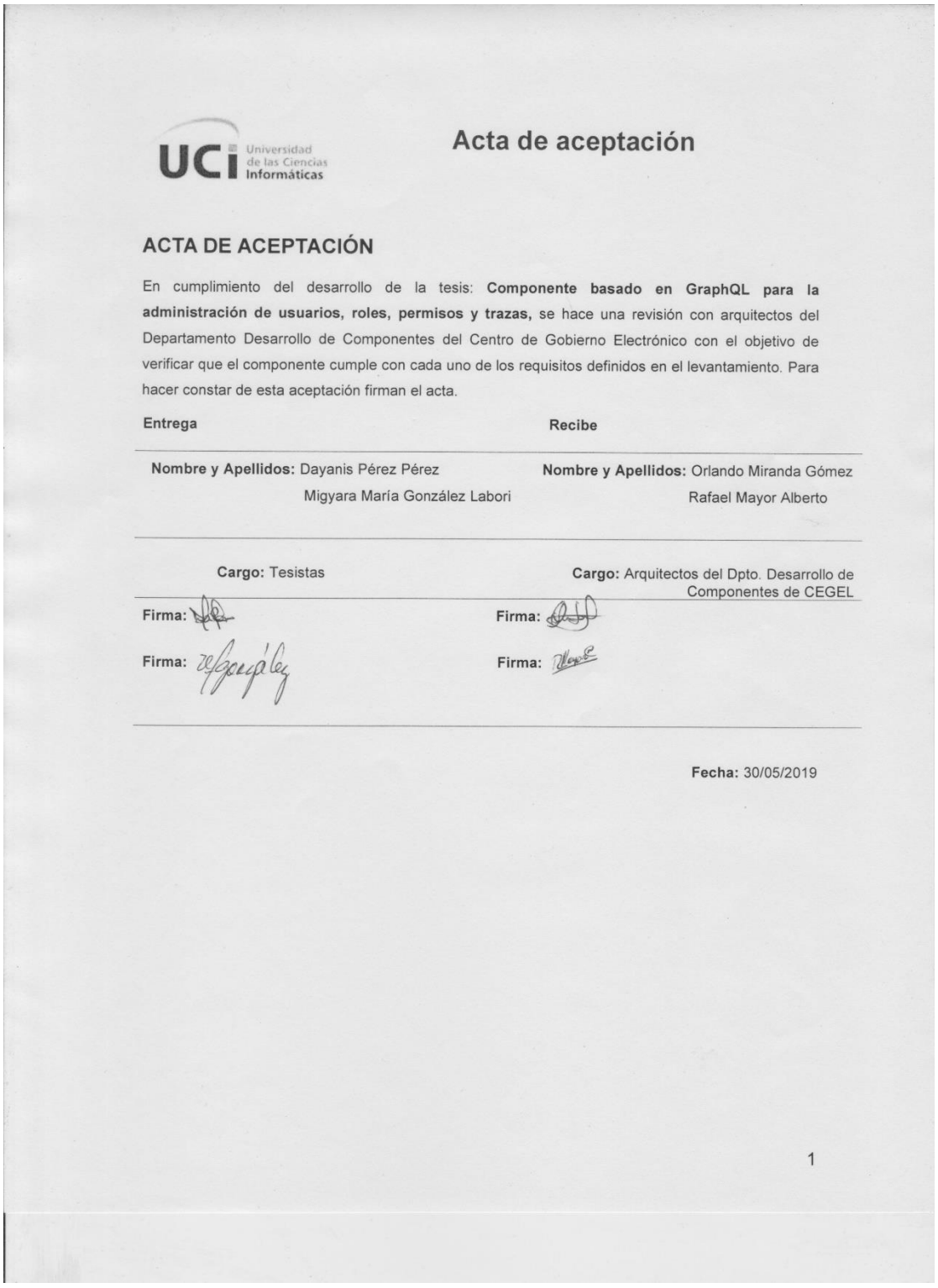
Migyara María González Labori
Autora de la Tesis

Dayanis Pérez Pérez
Autora de la Tesis

CEGEL
Universidad de las Ciencias Informáticas
CENTRO DE GOBIERNO ELECTRÓNICO

Figura 19. Acta de liberación por el grupo de calidad de CEGEL (Parte 2)
Fuente: elaboración propia

Anexo 4: Acta de aceptación



UCI Universidad de las Ciencias Informáticas

Acta de aceptación

ACTA DE ACEPTACIÓN

En cumplimiento del desarrollo de la tesis: **Componente basado en GraphQL para la administración de usuarios, roles, permisos y trazas,** se hace una revisión con arquitectos del Departamento Desarrollo de Componentes del Centro de Gobierno Electrónico con el objetivo de verificar que el componente cumple con cada uno de los requisitos definidos en el levantamiento. Para hacer constar de esta aceptación firman el acta.

Entrega	**Recibe**
Nombre y Apellidos: Dayanis Pérez Pérez Migyara María González Labori	**Nombre y Apellidos:** Orlando Miranda Gómez Rafael Mayor Alberto
Cargo: Tesistas	**Cargo:** Arquitectos del Dpto. Desarrollo de Componentes de CEGEL
Firma:	**Firma:**
Firma:	**Firma:**

Fecha: 30/05/2019

1

Figura 20. Acta de aceptación
Fuente: elaboración propia