UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de APIs de microservicios en una aplicación Web para la gestión de residuos sólidos en la Ciudad de México

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Daniel Alberto Zarco Manzanares

DIRECTOR DE TESIS

Dr. Guillermo Gilberto Molero Castillo

Ciudad Universitaria, Cd. Mx., 2023

Agradecimientos

Personales

En primera instancia, mi agradecimiento es para los Orishas y muertos que me permitieron llegar a este punto de mi vida académica y personal. A **Shango** porque fue quien me acompañó en la carrera desde el primer día, brindándome las fuerzas y ayuda en cada una de las guerras que viví en las materias. A luchar sin miedo, sin importar lo difícil que fuera la situación o profesor al que me enfrentaba. A **Oggun** y **Ochosi** por ayudarme a enfocarme en lo más importante que puede hacer un ser humano, que es el trabajo, logrando el éxito en los proyectos, tareas y exámenes, sin importar su complejidad y dificultad.

A **Eshu-Eleggua** por darme las oportunidades en el momento correcto, logrando aprovechar cada instante de mi formación profesional. Además, por proporcionarme las personas correctas en los momentos adecuados de mi vida academia. A **Orunmila** por orientarme en cada paso que di, en cada decisión, por mostrarme y guiarme en el camino correcto. A **Olokun** por darme la estabilidad emocional en cada momento requerido y por permitirme encontrar el sosiego en cada situación. A **Obbatala** por darme la inteligencia y calma de afrontar cada reto de mi vida académica y personal. Y a todos los demás Orishas que estuvieron conmigo, brindándome su apoyo, cuidado y ashe.

A los muertos **Minerva**, **Martín**, **José Ángel**, **Agustín** y **Juana Morales** que me enseñaron a escoger correctamente cada materia y profesor para obtener los mejores resultados de aprendizaje y aprovechamiento, por acompañarme en cada una de mis clases, exposiciones, proyectos, exámenes y tareas. Por brindarme la inspiración necesaria cuando llegaba a callejones sin salida, crear soluciones, y por velar mi bienestar.

Otra muestra de agradecimiento y la más importante es para mi madre **Sandra Judith Manzanares Cuevas** por todo el apoyo que me ha brindado desde antes de

nacer, por todo el amor incondicional que me ha dado cada día de mi vida, por todo el trabajo arduo, por las noches sin dormir, por sembrar en mí desde la infancia el gusto por el estudio, la pasión por aprender y por el trabajo duro. Además, por enseñarme que todo lo que se cosecha es fruto de lo que se siembra. No omito el trabajo duro y sin descanso, de sol a sol, en el tianguis y la chachara para darme la oportunidad de estudiar y ser Ingeniero. Por guiarme en cada momento de mi vida y abstenerte de cosas para comprarme todo lo necesario: útiles, cuadernos, la tarjeta D10 Lite y cada componente electrónico de mi carrera, por hacerme un hombre de bien y alentarme cada día de mi vida y hacer una mejor versión de mí. Por enseñarme el valor del trabajo y del dinero a muy temprana edad, y lo más importante, por **haberme dado la vida**.

Otro agradecimiento es para mi papi **Juventino Manzanares Enriques** que me brindó su amor y apoyo desde que nací, porque gracias a ti pude cursar con éxito una de las materias más complejas y exigentes, *Computación Gráfica e Interacción Humano-Computadora*. Siempre recordaré cómo presumías que estudiaba Ingeniería con todo el mundo. Gracias por haber estado en mi vida y por el amor a manos llenas que me brindaste.

Agradezco también a mi abuelo **Herlindo Cruz Tabera**, por hacerme entender que lo más fácil fue entrar a la UNAM, pero lo verdaderamente difícil fue terminar la carrera. Gracias por apoyarme para que estudiara a través del trabajo constante, siempre ayudando sin esperar nada a cambio.

Otro agradecimiento es para la **C.D. Francisca Guadalupe Sánchez Rocha** y al **Sr. Rubén Arriaga Leyva**, por brindarnos su apoyo, a mi mamá y a mí, a través de su ayuda y ánimos a lo largo de estos años. Por el aliento para estudiar y superarme, el cual inició cuando cursé el nivel medio superior en el Colegio de Ciencias y Humanidades Plantel Sur, regalándome una serie de libros que fueron el bastión de mi aprendizaje. El que más recuerdo fue el libro de Física, que me ayudo hasta el primer semestre de mi carrera. Por sus consejos, por abrirnos las puertas de su casa y corazón, y por haberme dado uno de los consejos más importantes en mi vida: *Con el tiempo y un ganchito*; haciéndome entender que todo en esta vida se logra con trabajo y paciencia.

Agradezco también a la **T.S. Morelia Gonzáles Osornio**, por todo el apoyo brindado, por su compañía en los buenos y malos momentos. Por aconsejarnos y guiarnos, a mi madre y a mí, para tomar las mejores decisiones, y obtener así, los

mejores resultados en las diversas situaciones que compartimos juntos. Por alentarme a obtener las mejores calificaciones en cada semestre de mi carrera, seguir aprendiendo y superarme en mis habilidades de programación y liderazgo. Por los valores y modales para ser un caballero, y sobre todo, el cariño que nos brindó.

Académicos

Me gustaría mostrar mi agradecimiento al **M.I. Gabriel López Domínguez**, por sus valiosos consejos de vida, motivación y a la gran ayuda que me mostró cuando cursé su materia *Matemáticas Avanzadas*. Por estos consejos es posible que esté escribiendo estas palabras. Por motivarme y mostrarme el camino para que pudiera terminar mi carrera, y sobre todo, porque me mostró el camino que me permitió seguir disfrutando del amor más grande mi vida, que es mi madre. Además, por brindarme una de las mejores lecciones de mi vida: *Si creas un plan de trabajo, podrás alcanzar tu meta de terminar tu carrera. Si haces lo anterior, en poco tiempo estarás invitándome a tu titulación*. Además, por ayudarme a crecer como persona, no comparándome con nadie y que el momento más importante es el presente: *El ahora es lo más importante de una persona, solo importa el hoy y el mañana, el pasado no importa, la vida es como una transformada de Laplace, no nos importa lo que hay antes del cero, solamente importa lo que inicia en el cero, hacia adelante.*

También quiero agradecer al **Dr. Guillermo Fernández Anaya** porque me ayudo a superarme y a desarrollar mi potencial, cuando cursé su materia de *Estructuras Discretas* desarrollé como proyecto una red neuronal de retro-propagación sin haber cursado la materia de Inteligencia Artificial, el cual fue un trabajo sumamente complejo y difícil para un alumno de cuarto semestre, el cual logré con éxito.

Al **Ing. Carlos Aldair Roman Balbuena** que me ayudó y alentó a superar mis límites de abstracción en la programación, mejorar mi lógica y pulir mi forma de escribir código, mejorando su estructura y limpieza. Como también me ayudó en diversos momentos de mi vida como un amigo y mentor, brindándome sus consejos para afrontar las situaciones de la mejor manera posible, tanto académicas como personales.

Al **Dr. Guillermo Gilberto Molero Castillo**, quien me brindó la oportunidad de mejorar mis habilidades de investigación, redacción y de abstracción tecnológica,

y que me permitió desarrollar este trabajo. A su constante apoyo que me dio en este proyecto, el cual representó un reto para mí, por su constante aliento para generar mayor calidad en mi vida académica y mi futura vida laboral. Sin usted, esta tesis no sería realidad.

Finalmente, agradezco la participación como becario de licenciatura en el proyecto "PLATAFORMA SEDEMA", el cual fue financiado por la **Secretaría de Educación, Ciencia, Tecnología e Innovación de la Ciudad de México** (SECTEI), con número de referencia **SECTEI 247/2021**.

Resumen/Abstract

La producción de residuos sólidos urbanos en la Ciudad de México es un problema con un comportamiento exponencial con respecto al crecimiento de la población a lo largo del tiempo. **Problema de investigación.** Dada la producción de los residuos sólidos urbanos en la Ciudad de México, la Secretaría del Medio Ambiente (SEDEMA) buscó desarrollar una aplicación web que permita gestionar de manera eficiente los procesos de traslado, transporte, recolección y tratamiento de los diferentes tipos de residuos sólidos urbanos producidos en la Ciudad de México. **Objetivo.** La presente tesis se enfoca en el desarrollo de APIs de microservicios en una aplicación web para la gestión de residuos sólidos urbanos en la Ciudad de México, las cuales permiten una comunicación rápida y eficiente entre el Back-End y Front-End, permitiendo así una gestión supervisada de los residuos sólidos urbanos producidos en la Ciudad de México. **Motivación.** En la actualidad, los patrones de consumo de la población genera un creciente problema ambiental, el cual se vuelve crítico con el paso del tiempo. Como consecuencia, se requieren de diversas soluciones para una gestión eficiente de los residuos sólidos, esto con el fin de contener o disminuir la deuda ambiental. **Método.** El método utilizado es una solución basada en el ciclo de vida del desarrollo de software, el cual fue estructurado en cinco etapas: a) definición de requerimientos, b) diseño, c) desarrollo de catálogos, d) desarrollo de operadores CRUD, y e) pruebas de funcionalidad. **Resultados.** Una vez que las APIs fueron codificadas y probadas, se observó el correcto funcionamiento de estas, con tiempos óptimos de respuesta, de acuerdo con las especificaciones técnicas de GraphQL. Este aspecto de rapidez y eficiencia permitió también comprobar el funcionamiento del Back-End. Para así, posteriormente, seguir escalando más funcionalidades y peticiones simultáneas en la aplicación, esto desde el lado del cliente. **Conclusión.** La arquitectura basada en microservicios permitió tener una aplicación web funcional, para la cual se implementaron diferentes APIs, que contienen operadores CRUD, para diferentes catálogos de la base de datos, asegurando la funcionalidad y rapidez en la ejecución de estos, preservando así la inmediatez en el acceso a la información.

Índice general

Índice de figuras

Índice de tablas

Lista de códigos

Lista de abreviaciones

CRUD . . . . . Operaciones de creación, lectura, actualización y eliminación (Create, Read, Update, Delete).

DBM . . . . . Data Base Manager (Manejador de base de datos).

ER . . . . . Entidad-Relación. Hace referencia a diagramas entidad-relación.

LAU-CDMX . Licencia Ambiental Única para la Ciudad de México.

LGPGIR . . . Ley General para la Prevención y Gestión Integral de los Residuos.

PROFEPA . . Procuraduría Federal de Protección al Ambiente.

RAMIR . . . . Registro y Autorización de Establecimientos Mercantiles, de servicios y/o unidades de transporte relacionados con el manejo integral de residuos sólidos urbanos y/o de manejo especial de competencia local que operen y transiten en la Ciudad de México.

RME . . . . . Residuos de manejo especial.

RP . . . . . Residuos peligrosos.

RSU . . . . . Residuos sólidos urbanos.

SEDEMA . . . Secretaría del Medio Ambiente de la Ciudad de México.

SECTEI . . . . Secretaría de Educación, Ciencia, Tecnología e Innovación de la Ciudad de México.

SDLC . . . . . Software Develpoment Lyfe Cicle (Ciclo de vida de desarrollo de software).

WSDL . . . . . Web Services Description Language (Lenguaje de descripción de servicios Web).

WSML . . . . Web Services Modeling Language (Lenguaje de modelado de servicios Web).

WSMO . . . . Web Services Modeling Ontology (Ontología de modelado de servicios Web).

SWS . . . . . Semantic Web Services (Servicios web semánticos).

ISWM . . . . . Integrated Sustainable Waste Management (Integrado de Administración Sustentable de Residuo).

1 Introducción

Índice

1.1. Contexto de la investigación

Durante el siglo pasado, como consecuencia del crecimiento exponencial de la población mexicana, se generó una mayor demanda de materias primas para satisfacer el creciente consumo de bienes y servicios, creando la proliferación de la industria. Es aquí, donde los patrones de consumo de la población se tornan más agresivos y demandantes con los recursos disponibles en la región, junto al corto tiempo en el cual buscan ser redimidos. En este agresivo panorama creció, de forma directa, la generación de residuos de diferentes tipos, sumando el problema del manejo, control y ciclo de vida de estos. Ya que, al no tener una disposición adecuada, esto provoca afecciones a la salud humana y los ecosistemas (SEMARNAT, 2013).

Los residuos, que se definen en la Ley General para la Prevención y Gestión Integral de los Residuos (LGPGIR) emitida por la Procuraduría Federal de Protección al Ambiente (PROFEPA), son todos aquellos materiales o productos cuyo propietario o poseedor desecha; y que se encuentra en estado sólido o semisólido, líquido o gaseoso, y que se contienen en recipientes o depósitos destinados a este propósito. Estos residuos pueden ser susceptibles de ser valorizados o que se requieran sujetarse a algún tratamiento o disposición final, conforme a lo dispuesto en la LGPGIR (DOF, 2003).

En el mismo sentido, la producción de alimentos, así como la fabricación y el consumo de bienes para el hogar y la industria, son ejemplos de actividades cotidianas que producen una amplia cantidad de residuos. Dependiendo de su composición, tasa de generación y proceso de manejo, pueden tener efectos diversos en la población y en los ecosistemas en los que se encuentran. Específicamente, generando un daño ambiental, donde en algunos casos, estos residuos pueden convertirse en *peligroso*; lo que sucede cuando en su composición se encuentran elementos tóxicos, que podrían provocar un incorrecto manejo y control debido a una inadecuada gestión (SEMARNAT, 2019).

En general, los residuos necesitan estar sujetos a tratamiento o disposición final con base en lo dispuesto en la LGPGIR, y su clasificación se basa de acuerdo con las características y orígenes de los mismos. Esta clasificación está dada por tres grupos (SEMARNAT, 2019): residuos sólidos urbanos (RSU), residuos de manejo especial (RME) y residuos peligrosos (RP).

1.2. Problema de investigación

De acuerdo con algunas estimaciones, la producción mundial de residuos sólidos urbanos (RSU) alcanzó aproximadamente 1300 millones de toneladas diarias en 2010, y podría crecer hasta los 2200 millones en el 2025 (SEMARNAT, 2019). El volumen de generación de RSU, a nivel global, muestra disparidad respecto al desarrollo económico y la proporción de la población urbana en cada región. En 2010, cerca del 44 % de los residuos sólidos urbanos producidos en el planeta, correspondían a los países con economías desarrolladas, miembros de la OCDE (Organización para la Cooperación y el Desarrollo Económicos).

Los países de Latinoamérica y el caribe contribuyeron con el 12 % del total de la producción mundial de residuos sólidos urbanos, detrás de los países que integran las regiones del Pacífico y del Este de Asía (SEMARNAT, 2019). En México, según las cifras más recientes, publicadas en 2017, la generación de RSU alcanzó los 44.6

millones de toneladas diarias, lo que representó un aumento del 35.6 % con respecto a 2003, lo que significó 11.73 millones de toneladas generadas más que en ese periodo (SEMARNAT, 2019). Si se expresa por habitante, alcanzó 0.98 kilogramos en promedio diario en el mismo año.

La cantidad de RSU generados puede explicarse como el resultado de múltiples factores, como el crecimiento urbano, demográfico, industrial, modificaciones tecnológicas y los patrones agresivos de consumo de la población en México. Este crecimiento de RSU es una relación directamente proporcional con el gasto de consumo privado y el PIB (Producto Interno Bruto) nacional (SEMARNAT, 2019). Esta relación expresa que, a mayores ingresos, el nivel de consumo se incrementa, y, por lo tanto, la cantidad de residuos sólidos urbanos generados aumenta de la misma forma. Este fenómeno es compartido por otros países alrededor del mundo.

Esta relación se traslada a nivel federal, puesto que las que más contribuyeron al PIB nacional, también lo hicieron en la generación de RSU. Sin embargo, en algunos casos, la contribución a la producción total nacional de RSU de algunas entidades federativas se desvía completamente de la tendencia general, como en el caso del Estado de México (que produce más residuos respecto a la tendencia), o Campeche que produce menos de lo esperado. En el caso del Estado de México, su alta generación de RSU se debe a su alta actividad industrial y su alta densidad poblacional, mientras que, en el estado de Campeche, su comportamiento se debe a su intensa actividad petrolera y su baja densidad poblacional (SEMARNAT, 2019).

En consecuencia, hay una creciente necesidad de gestionar la generación de los residuos sólidos urbanos en la Ciudad de México, dado que es necesario contar con una plataforma tecnológica capaz de proveer información de las cantidades de residuos que se generan y son manejados por los actores en este flujo, como generadores, transportadores y recicladores. Esto con el fin de tener un mejor control e invertir recursos para cumplir cada una de las características de una economía circular.

1.3. Pregunta de investigación

Tomando en cuenta la problemática descrita en la sección anterior, se plantea la siguiente pregunta de investigación que se pretende responder con base en los resultados del trabajo de tesis:

- ¿Qué características se deben incluir en el desarrollo de componentes de software para una plataforma Web que gestione los diferentes residuos sólidos urbanos producidos en la Ciudad de México?

1.4. Hipótesis

A partir de la problemática planteada y la pregunta de investigación, se establece la siguiente hipótesis:

- Es posible automatizar, a través de APIs de servicios Web, el flujo de trabajo de los procesos inherentes en la gestión de diferentes tipos de residuos sólidos urbanos producidos en la Ciudad de México.

Para probar la hipótesis se diseñó y desarrolló APIs de servicios Web para una aplicación de gestión de residuos sólidos urbanos de carácter optativo en la Ciudad de México, cumpliendo así los requerimientos de software de la Secretaría del Medio Ambiente (SEDEMA) de la Ciudad de México, y las regulaciones de licenciamiento, contenidas en la Licencia Ambiental Única para la Ciudad de México (LAU-CDMX) y en el Registro y Autorización de Personas Físicas y Morales para el Manejo Integral de Residuos de Competencia Local que Operen y Transiten en la Ciudad de México (RAMIR).

1.5. Objetivo general y específicos

1.5.1. General

- Desarrollar APIs de servicios Web para la gestión y control de los residuos sólidos urbanos de carácter optativo en la Ciudad de México.

1.5.2. Específicos

- Realizar el análisis y diseño de los componentes de software requeridos por el usuario final.
- Construir las APIs como parte de las funcionalidades de la aplicación del lado del servidor (Back-End).
- Validar el correcto funcionamiento de las APIs previamente implementadas.

1.6. Motivación

Uno de los mayores indicadores de impacto ambiental es la *huella ecológica*, la cual es interpretada como la demanda humana respecto a la superficie ambiental, que se

requiere para generar los recursos[1] que se consume, como la demanda necesaria para albergar los asentamientos humanos y la infraestructura requerida para absorber el dióxido de carbono (CO_2), liberado a la atmósfera por la quema de combustibles fósiles (SEMARNAT, 2019).

No obstante, por más de cuarenta años, la demanda humana sobre los recursos naturales ha excedido lo que el planeta es capaz de regenerar. Así, esta huella ecológica, que se mide en hectáreas, requiere suministrar los bienes ecológicos y servicios que supera la biocapacidad, esto es, la tierra disponible que actualmente se usa para proveer de bienes y servicios. Tanto la biocapacidad como la huella ecológica son expresadas en una unidad común, denominada *hectárea global* (gha, por sus siglas en inglés) (McLellan et al., 2014).

En este sentido, el problema de la generación de residuos sólidos urbanos, que aumenta de manera exponencial de la mano con el crecimiento poblacional, implica sacrificar los recursos naturales, renovables y no renovables, provocando serios daños al ecosistema y a la salud de las personas (Kumar, 2014). Además, la producción de bienes y servicios, así como su consumo, son las fuerzas más importantes que generan cambios en el ambiente. Por lo que, es importante reconocer la dependencia del sistema socioeconómico con respecto de la integridad del lugar que se habita.

Ante esto, es necesario generar y gestionar información útil que permita conocer con mayor detalle la magnitud y efecto del manejo de los residuos sólidos urbanos que se generan en las grandes metrópolis, como el de la Ciudad de México, donde se tiene una amplia diversidad de actividades humanas, que en conjunto afectan el ambiente y sus respectivos ecosistemas. Esta gestión de los residuos sólidos podrían ser útiles para avanzar hacia un manejo adecuado de estos, los cuales se generan a diario en la Ciudad de México.

Por lo tanto, se eligió como parte de la solución tecnológica el desarrollo de una aplicación Web, la cual permite gestionar la información de la generación de los residuos sólidos urbanos en la Ciudad de México. El acceso a esta aplicación Web es a través de diferentes dispositivos, esto es, posee la característica de ser responsiva. Es importante mencionar que una aplicación Web es considerada como un sistema de información híbrida de hipermedia, cuyo contenido se determina en función de las necesidades del usuario. Una de las ventajas de las aplicaciones Web y de sistemas basados en

[1]Los recursos, desde un aspecto económico pueden ser divididos en cuatro categorías: trabajo, capital, naturales y capacidad empresarial, y dentro de los recursos naturales se tiene a los renovables y los agotables; donde los primeros son aquellos que, mediante una buena administración, control y manejo, pueden generarse de nuevo en un determinado tiempo, y los segundos son limitados y que no pueden generarse nuevamente en un determinado tiempo.

Web es su accesibilidad a través de los navegadores Web, y estos proveen portabilidad y una interfaz multimedia homogénea. Estas características han permitido que las aplicaciones Web sigan teniendo un crecimiento importante en la industria (Duan y Edwards, 2005).

1.7. Estructura y contenido de la tesis

El documento de tesis está dividido en cinco capítulos. En este primer capítulo se describe el contexto de la investigación, se incluye una descripción del problema, se plantea la pregunta de investigación y la hipótesis; y se especifican los objetivos y la motivación de la investigación. Dando lugar al desarrollo de este trabajo de investigación, cuyos resultados se presentan en los capítulos siguientes.

En el *Capítulo 2* se presenta los fundamentos de la arquitectura de cliente-servidor con implementación de microservicios, los aspectos fundamentales que provee la Ingeniería de Software, la importancia y características del aseguramiento de la calidad (QA, por sus siglas en inglés), y las características que debe de poseer una aplicación web. Finalmente, se dan a conocer los trabajos relacionados.

El *Capítulo 3* presenta la propuesta de solución del desarrollo de APIs de servicios Web para la gestión y control de los residuos sólidos urbanos generados en la Ciudad de México. Para esto se utilizó un método de desarrollo ágil, que cuenta con cinco etapas: definición de requerimientos; diseño del sistema y software; desarrollo y pruebas unitarias; integración y pruebas del sistema; y operación y mantenibilidad.

El *Capítulo 4* muestra los resultados obtenidos de las pruebas de las APIs desarrolladas de los dos catálogos seleccionados, también el funcionamiento del microservicio para el registro del domicilio de una empresa con autocompletado de campos, todos mostrados en el capítulo anterior.

El *Capítulo 5* presenta las conclusiones generales y particulares del trabajo de investigación realizado, y se establece la base para incluir nuevas funcionalidades requeridas por los usuarios finales, esto es, usuario SEDEMA.

Finalmente, se presentan Anexos, donde se muestran los códigos de los catálogos desarrollados y sus respectivas pruebas unitarias y de integración (*Anexos A y B*). También se muestra información ampliada sobre la implementación de un servidor local en el lenguaje de programación Go (*Anexo C*).

2

Marco Teórico y Estado del Arte

Índice

En este capítulo se presentan los fundamentos teóricos sobre Ingeniería de Software, factores de calidad del software, aplicaciones Web, arquitectura basada en microservicios y las tendencias actuales de las aplicaciones Web. Por otro lado, se abordan estudios sobre los problemas en el manejo de los residuos sólidos urbanos en México y sus estados, haciendo énfasis en ciudades como la Ciudad de México, debido al crecimiento y generación de este tipo de residuos.

2.1. Ingeniería de Software

La Ingeniería de Software (IS) es la disciplina concerniente con todos los aspectos de la producción de software, desde sus etapas iniciales en la especificación de requerimientos, atravesando la etapa de *diseño*, para después usar la aplicación que fuese desarrollada. A manera de ejemplo, más del 75 % de la población global posee un

teléfono celular, el cual está controlado por software, que fue previamente desarrollado, de los cuales la mayoría de estos dispositivos tienen acceso a Internet (Somerville, 2016).

Por lo que, en el mundo actual, la Ingeniería de Software es una herramienta tecnológica esencial en el gobierno, sociedad, empresas y en diversas instituciones públicas y privadas en general. Gran parte de los flujos de trabajo son automatizados a través de la IS, los cuales son controlados por sistemas basados en computadoras y software especializado. Sus aplicaciones son variadas, desde la gestión y control de la manufactura, sistemas financieros hasta la industria musical, los juegos de computadora, entretenimiento, y otros; donde el uso de software es sumamente importante. Entre las características esenciales que garantizan que el software que fuese desarrollado sea de calidad destacan -Tabla 2.1- (Somerville, 2016):

Tabla 2.1: Características esenciales que garantizan el buen desarrollo de software.

Características	Descripción
Mantenibilidad	El software debe de ser escrito en la medida que pueda evolucionar ante nuevos requerimientos de los usuarios. Este es uno de los atributos críticos, ya que el software, en su condición natural, se encuentra en constante cambio.
Confiabilidad y Seguridad	La confiabilidad del software incluye características sobre la fiabilidad y seguridad, tanto en su construcción como en su uso.
Eficiencia	La eficiencia incluye la capacidad de respuesta, el tiempo de ejecución, el uso de memoria, entre otros aspectos técnicos relacionados con la misma.
Aceptabilidad	El software debe ser aceptable en términos del tipo de usuario y obedecer el diseño con el cual fue concebido. Es decir, debe ser usable, entendible y compatible con otras aplicaciones que los usuarios han utilizado.

La importancia de la Ingeniería de Software se caracteriza por dos aspectos significativos (Somerville, 2016): i) la dependencia de las personas con el software crece de manera continua, haciendo que cada vez se cubran mayores necesidades y requerimientos de los usuarios finales; y ii) en términos de costo-beneficio, se usen técnicas adecuadas para hacer herramientas eficientes y rentables a lo largo del tiempo.

En el desarrollo de software se utilizan enfoques sistemáticos, conocidos también como *procesos de software*. Estos procesos son una secuencia de actividades que

conducen y lideran la producción del software, normalmente estructurados en etapas, como (Somerville, 2016):

1. **Especificación y diseño**, donde los usuarios, en conjunto con los ingenieros de software, definen los requerimientos y funcionalidades del software, que serán desarrollados y las restricciones de su operación.
2. **Desarrollo**, donde el software es construido mediante frameworks, lenguajes de programación y demás herramientas de desarrollo.
3. **Validación**, donde el software que fue programado es revisado para asegurar los requerimientos de los usuarios finales.
4. **Evolución**, donde el software que fue producido es modificado para reflejar cambios basados en los requerimientos de los usuarios.

Los diferentes tipos de sistemas requieren de procesos de desarrollo de software, por ejemplo, hay aplicaciones que requieren ser completamente especificadas antes de su desarrollo; mientras que en otras, como en las de comercio electrónico, tanto la especificación y desarrollo se pueden realizar de manera paralela. En consecuencia, las actividades de desarrollo pueden estar organizadas y descritas en diferentes niveles de detalle, dependiendo del tipo del software que será desarrollado (Somerville, 2016).

2.2. Factores de calidad de software

El término *buena calidad* es utilizado como expresión de satisfacción de algún producto o servicio que se haya adquirido. En contraste, *mala calidad* sirve para expresar insatisfacción, descontento o enojo sobre algún producto o servicio (Chemuturi, 2011). En consecuencia, en la sociedad existen diferentes connotaciones de calidad:

- Para los usuarios finales de un producto, la calidad denota una ausencia de defectos en la funcionalidad del mismo. Además, muestra confianza, es de fácil uso, tiene niveles aceptables en la tolerancia a fallos, y seguridad contra lesiones o daños hacia los usuarios del producto.
- Para los usuarios finales de un servicio, la calidad denota confiabilidad de desempeño, facilidad de obtener el servicio, resolución de problemas asociados al mismo, y protección contra daños que puedan generar al usuario.
- Para un productor de bienes, la calidad denota el cumplimiento de las especificaciones del producto, que pueden ser definidas por estándares o una organización.
- Para un proveedor de servicios, la calidad denota conocer los tiempos límite de entrega del servicio que conforman las especificaciones y estándares.

- Para una asociación industrial o grupo de estándares, la calidad denota el acto de salvaguardar la reputación de la industria y la protección ante fraudes y demandas.

Aunado a lo anterior, según la norma ISO 9000:2005, *calidad* es el grado en el que un conjunto de características cumplen con los requisitos necesarios para proveer adecuadamente la confianza de los productos o servicios (ISO, 2015). Así, en términos de desarrollo de software, la calidad es un patrón sistemático y planeado de todas las acciones necesarias para establecer los requerimientos técnicos y las actividades para los productos desarrollados o manufacturados (IEEE, 1990). De este modo, al garantizar que los requerimientos del proyecto se cumplan de manera satisfactoria, se provee la confianza y confiabilidad, garantizando la calidad de la aplicación (Buckley & Poston, 1984).

En la actualidad, existen diferentes modelos de calidad de software, como el propuesto por McCall, que consiste de 11 factores de calidad, proveyendo un método práctico y actual para la clasificación de los requerimientos de software (Galin, 2004). El modelo de McCall agrupa los factores en tres categorías (Galin, 2004): operación, revisión, y transición del producto, tal como se muestra en la Figura 2.1.

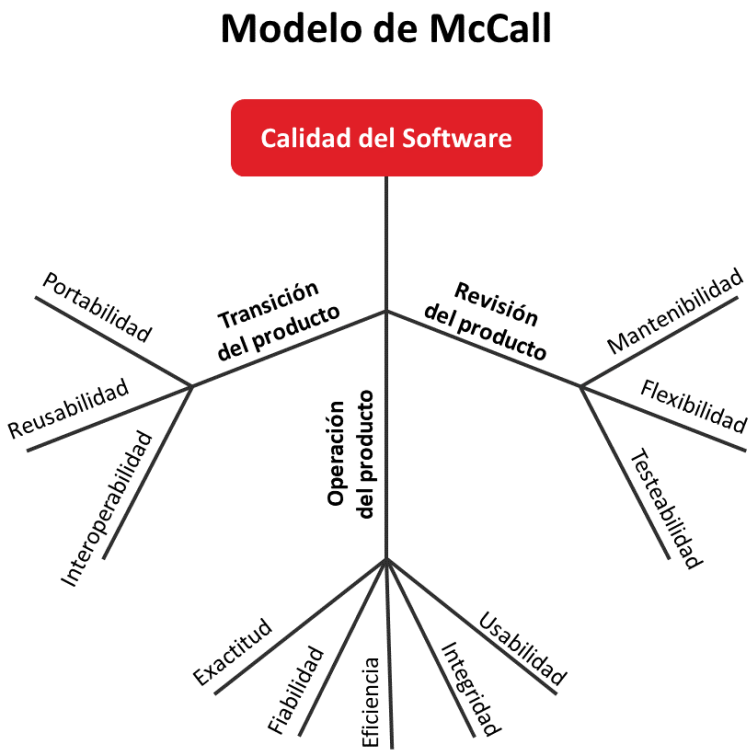

Figura 2.1: Modelo de factores de McCall. Adaptado de (Galin, 2004).

- **Operación del producto**. Abarca la exactitud, fiabilidad, eficiencia, integridad y usabilidad del software.
- **Revisión del producto**. Hace referencia a la mantenibilidad, flexibilidad y testeabilidad, esto es, la capacidad de ser sometido a pruebas.
- **Transición del producto**. Conjunta la portabilidad, reusabilidad e interoperabilidad del software.

2.3. Aplicaciones Web

World Wide Web (la Web) fue lanzada a principios de los 90, con el objetivo de permitir el acceso a la información desde cualquier recurso. Por lo que, la Web, en la actualidad, es una poderosa plataforma que ofrece una serie de herramientas y aplicaciones útiles para los usuarios. Además, la nueva generación de aplicaciones basadas en la Web ofrecen la posibilidad de comunicación y colaboración, e incluso, ofrecen capacidades de una aplicación nativa (Jazayeri, 2007).

A través de la Web se gestiona la información que se comparte por Internet, siendo necesario tener una conexión a este. Entre las características de una aplicación Web destacan (Jazayeri, 2007):

1. Pueden gestionar datos estructurados, por ejemplo, los registros de una base de datos; y los no estructurados, como los recursos multimedia.
2. Permite el acceso exploratorio a través de navegadores Web, con el propósito de navegar a través de sus diversos elementos.
3. Puede tener un alto nivel de calidad gráfica.
4. Permite la personalización y posibilidad de adaptación dinámica del contenido y estilos de presentación.
5. Permite soportar comportamientos proactivos, como el filtrado y recomendación de información.

La Web continúa evolucionando, tomando como punto de referencia las aplicaciones web tradicionales, las cuales abarcan desde la Web 1.0, pasando a la Web 2.0, que son dinámicas e invitan al usuario a interactuar con estas, como si se tratara de una aplicación de escritorio (Jazayeri, 2007). La Figura 2.2 muestra la arquitectura cliente-servidor de una aplicación Web 1.0, donde se necesita el navegador web y el servidor, donde se aloja el dominio de la aplicación Web.

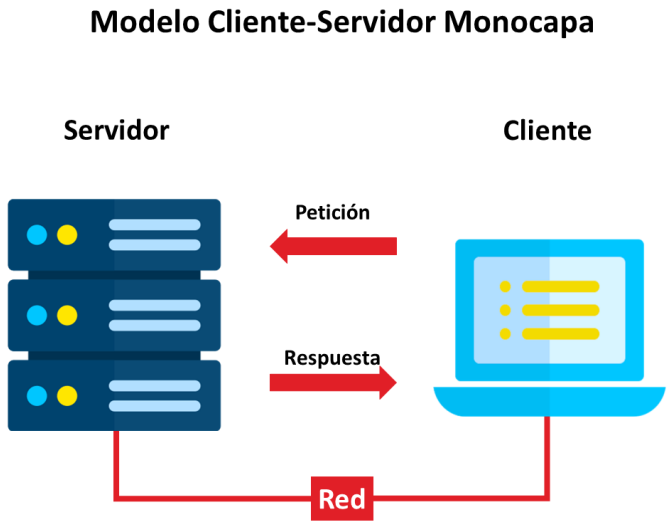

Figura 2.2: Arquitectura cliente-servidor monocapa para la ejecución de la Web 1.0.

En la arquitectura cliente-servidor, la comunicación se realiza mediante buses de datos, los cuales permiten intercambiar mensajes a través de llamadas al servidor desde el cliente, el cual obtiene una respuesta del servidor. Estas llamadas se realizan mediante el protocolo HTTP y HTTPS. Este último orientado a las aplicaciones Web 2.0, dotando la encriptación de los datos para garantizar una capa adicional de seguridad (Serain, 1995).

Por otro lado, la arquitectura de tres capas es una extensión del modelo monocapa, ya que en este se tiene al cliente, el servidor y la base de datos que interactúa con el sistema (Figura 2.3). Este tipo de arquitectura tiene un patrón de diseño conocido como Servicio Múltiple Cliente-Servidor, el cual consiste en que varios clientes realicen una petición a un servicio, y el servicio se encarga de manejar las llamadas de los clientes.

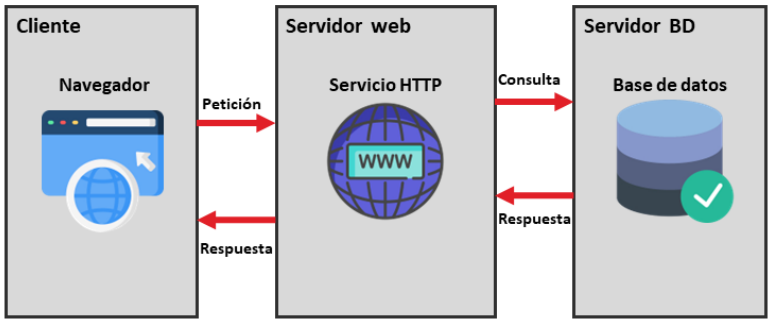

Figura 2.3: Arquitectura cliente-servidor tricapa.

Mientras que una arquitectura de Múltiples Clientes/Único Servidor (Figura 2.4) se da cuando diversos clientes utilizan un servicio alojado en un único servidor, atendiendo las llamadas de los clientes, el cual se encuentra alojado en un nodo de una red local (Gomaa, 2011).

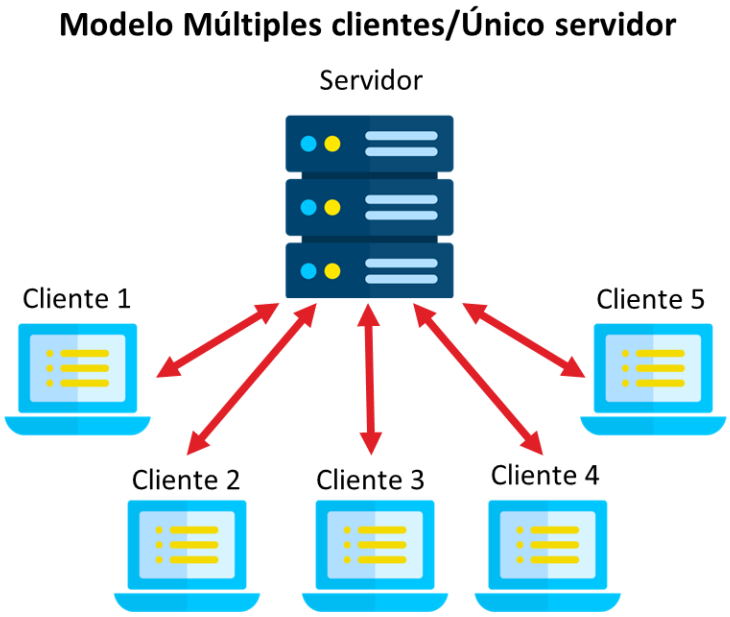

Figura 2.4: Patrón de Múltiples Clientes/Único Servidor.

2.4. Arquitectura basada en microservicios

Un *microservicio* es la modularización del software, cuya práctica se ha utilizado con éxito en proyectos de gran tamaño, para posteriormente ser dividido en módulos más pequeños, y así facilitar su diseño e implementación. Por lo que, los microservicios utilizan módulos para ejecutar diferentes procesos, haciendo que la arquitectura basada en microservicios ganara popularidad en los últimos años en la industria del desarrollado de software (Wolff, 2017).

Algunas características significativas de los microservicios son (Al-Debagy & Martinek, 2020):

1. Tienen como propósito dividir sistemas de software complejos en partes más pequeñas.

2. Pueden ser lanzados de manera independiente y pueden ser cambiados en producción sin afectar a otros microservicios.

3. Pueden ser implementados en diferentes tecnologías y no existe una restricción del lenguaje de programación.

4. Poseen almacenamiento de datos propios, esto es, una base de datos privada o un esquema individual en una base de datos compartida.

5. Pueden tener sus propios servicios de apoyo, como motor de búsqueda o una base de datos específica. Además, cuenta con una plataforma común para su ejecución, por ejemplo, máquinas virtuales.

6. Se comunican a través de la red, utilizando protocolos que soportan un acoplamiento, como REST o mensajería.

Una arquitectura basada en microservicios representa una ventaja ante una arquitectura monolítica (Figura 2.5), donde la implementación de un software de gran tamaño y complejidad debe ser lanzado en una sola pieza, o un solo ejecutable; reduciendo la flexibilidad e incrementando el costo de tiempo de carga (Sarita & Sebastian, 2017; Wolff, 2017). Mientras que al utilizar una arquitectura basada en microservicios permite dividir el software en módulos, permite realizar cambios de manera sencilla y eficiente (Wolff, 2017).

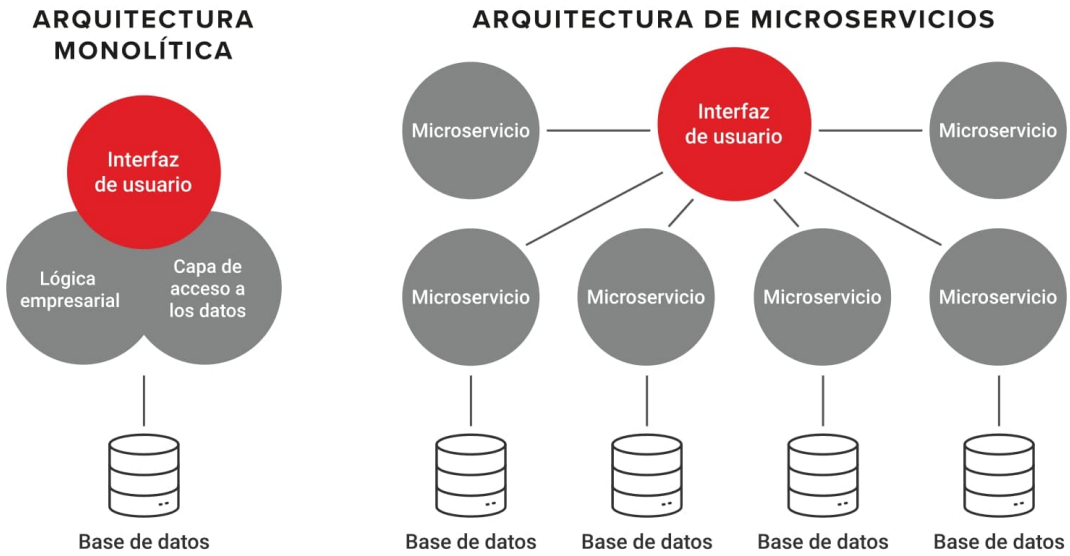

Figura 2.5: Arquitecturas monolítica y basada en microservicios.

En este sentido, los microservicios ofrecen el concepto de *modularización fuerte*, es decir, se utilizan diferentes componentes de software, como Ruby GEMs, JAVA JARs, .NET assemblies o Node.js NPMs. Así, los microservicios se comunican a través de interfaces explícitas, las cuales funcionan a través de mecanismos como mensajes o REST. Lo anterior, permite que la barrera técnica para el uso de microservicios sea alta, pero se reducen las dependencias no deseadas. Otra ventaja es su *flexibilidad*, dado que tienen la ventaja de reemplazar microservicios existentes (Wolff, 2017).

En los microservicios, las ventajas de *modularización fuerte* y *flexibilidad* permiten un adecuado desarrollo de software, además contrarrestan la erosión de la arquitectura de un sistema, ya que poseen la ventaja de no estar ligados a una tecnología específica, y pueden ser reemplazados uno a uno para solventar posibles problemas identificados. Además, otra ventaja de trabajar con una arquitectura basada en microservicios es la posibilidad de agregar diferentes microservicios a un sistema legado (Wolff, 2017), logrando así la modularización sobre diferentes tecnologías (Figura 2.6).

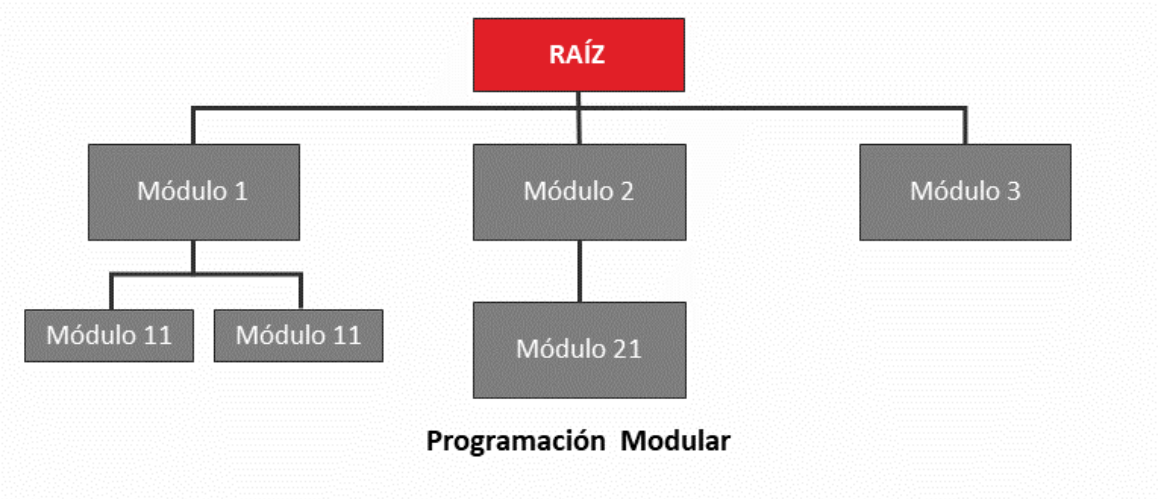

Figura 2.6: Esquema de modularización basada en microservicios.

Por otro lado, los microservicios pueden ser escalados independientemente de los otros componentes y microservicios adyacentes. Esto implica remover la limitación de tener que escalar todo el sistema, lo que simplifica la operatividad de la aplicación implementada. Además, el factor de aislamiento en los microservicios permite elegir tecnologías específicas que brindan mayor facilidad en el desarrollo de algún requerimiento o función, por ejemplo, una base de datos. Esta ventaja permite reducir la incertidumbre al tomar decisiones sobre qué tecnologías usar, y más importante, permite probar nuevas tecnologías que sean innovadoras y generen valor en la implementación (Wolff, 2017).

Cuando las aplicaciones están en producción, los microservicios ofrecen la ventaja de entregas continuas, ya que estos pueden ser desplegados de manera independiente. Al generarse entregas continuas se tiene un menor riesgo comparado con el lanzamiento de una aplicación basada en una arquitectura monolítica. No obstante, desde el punto de vista de la Ingeniería de Software no todo son ventajas, los microservicios tienen también diferentes retos que se busca superar cuando son integrados en sistemas que se encuentran en producción o sistemas que son desarrollados completamente como microservicios. La Tabla 2.2 resume algunos de los retos a los que se enfrentan las aplicaciones basadas en microservicios (Wolff, 2017):

Tabla 2.2: Retos a los que se enfrenta la arquitectura basada en microservicios

Reto	Descripción
Relaciones ocultas	Las relaciones de los microservicios entre sí no son completamente claras al generar llamadas.
Refactorización compleja	Mover funcionalidades a través de los diferentes microservicios representa una tarea difícil de realizar.
Arquitectura de dominio	Si el sistema cuenta con una arquitectura de dominio sólida, se podrá asegurar la independencia de desarrollo a lo largo de los diferentes microservicios.
Complejidad de ejecución	Un sistema basado en microservicios monitorea el lanzamiento, control y ejecución de los mismos, lo cual incrementa la complejidad de las operaciones.
Complejidad de sistemas distribuidos	Las llamadas de los microservicios pueden fallar debido a la red donde se encuentran o se posee un reducido ancho de banda.

2.5. Tendencias en las aplicaciones Web

Hoy en día, a través de Internet, las aplicaciones y servicios Web es posible compartir información de manera instantánea, generando procesos de colaboración, comunicación y coordinación entre diferentes personas, distintas localidades y en tiempo real (Benslimane et al., 2008). Estos servicios Web surgieron como una necesidad de interactuar entre diferentes sistemas distribuidos, generando una relación de continuo desarrollo y automatización de las interacciones.

Como consecuencia, varios estándares soportan el desarrollo Web, como el Lenguaje de Descripción de Servicios Web (WSDL, por sus siglas en Inglés), UDDI (Universal

Description Discovery and Integration), SOAP (Simple Object Access Protocol), entre otros. Estos estándares dan sentido a la definición de servicios Web y al desarrollo de aplicaciones Web, creando una frontera de propósitos de invocación y uso. En consecuencia, el enfoque de "peso ligero" en los servicios Web se centra específicamente en dar un dominio a las APIs web y RESTful (Representaciones de estados de transferencia) (Benslimane et al., 2008).

En el mismo sentido, tomando a la Web como contexto, surge el concepto de *mashup*, que en español sería *triturado*, que busca generar un amplio número de aplicaciones Web con una combinación de recursos existentes, como las APIs web. Los mashups están enfocados en la distribución y agregación de información para dar soporte a los contenidos publicitarios en las aplicaciones Web.

La Figura 2.7 muestra el desglose de comunicación de un mashup de una aplicación web 2.0, utilizando los diferentes servicios Web a través de APIs públicas de diversos proveedores y una arquitectura por capas, denotando la suma de diferentes servicios Web de la forma $\sum_{i=0}^{n} servicio_i = mashup$, donde los servicios son consumidos por APIs públicas, alimentando a la aplicación Web con respuestas a las diferentes peticiones que realiza el usuario.

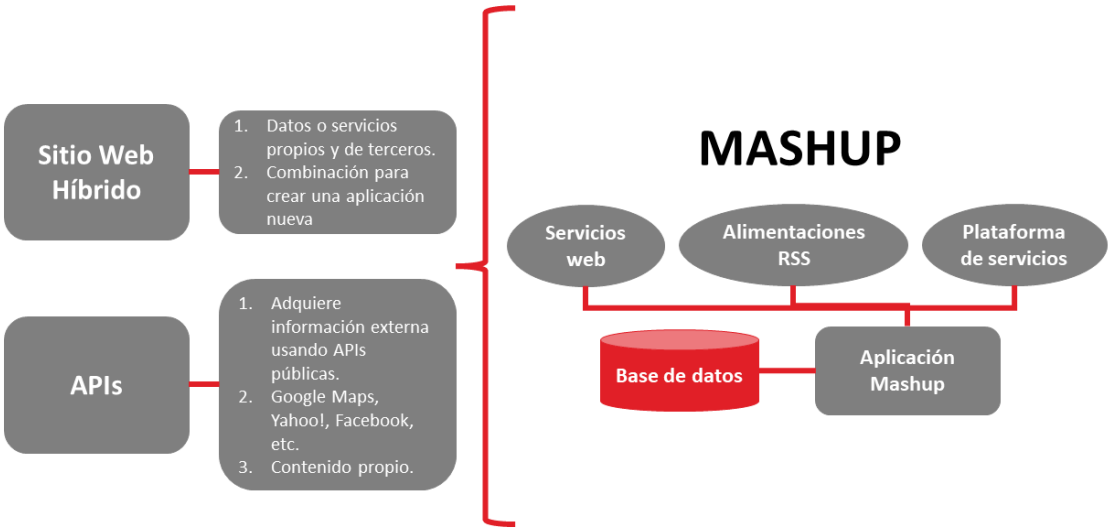

Figura 2.7: Estructura de un mashup en una aplicación Web, que hace uso de servicios Web a través de APIs de diferentes dominios.

Por otro lado, en «Towards re-engineering Web applications into Semantic Web Services» se realizó un análisis de los Servicios Web Semánticos (SWS, por sus siglas en inglés), que enriquecen a las aplicaciones Web mediante procesos semánticos, enfocándose en el uso de reingeniería para la construcción de un framwework, que

facilite la implementación de SWS en el código fuente de una aplicación Web (Bouchiha & Malki, 2010). Como resultado, se propuso un framework basada en la Ontología de Modelado de Servicios Web (WSMO, por sus siglas en Inglés), Figura 2.8, que incluye un conjunto de herramientas que se probaron en casos experimentales para validar y evaluar su desempeño (Bouchiha & Malki, 2010).

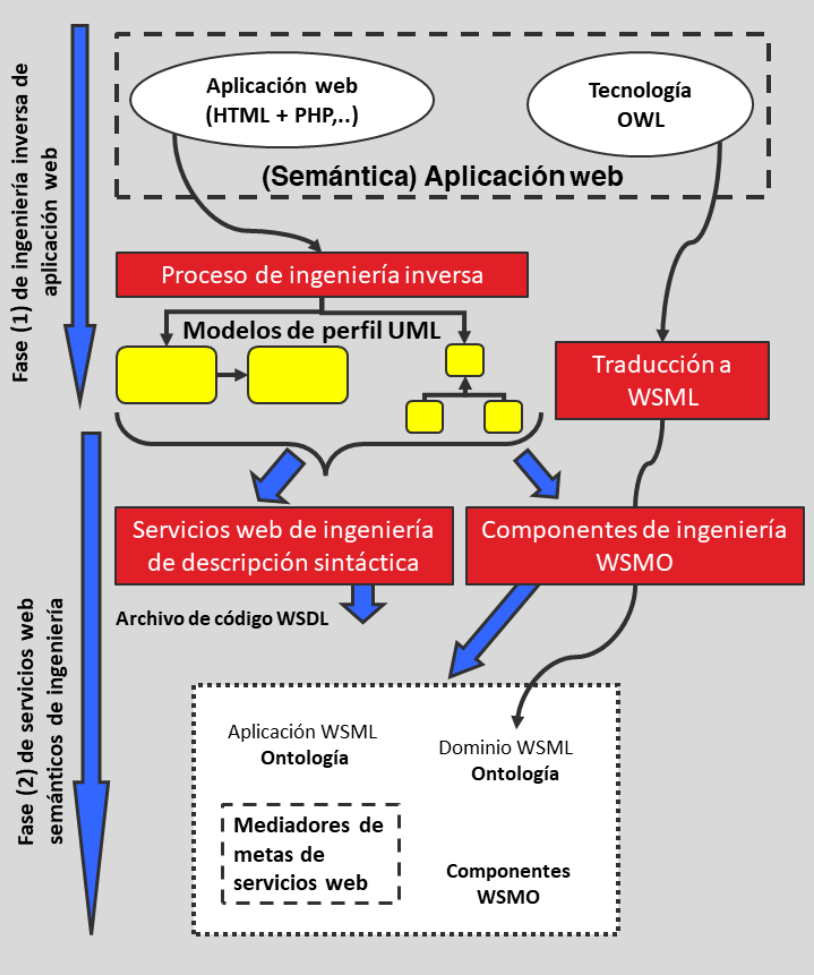

Figura 2.8: Esquema del framework basado en ingeniería inversa y servicios Web semánticos. Adaptado de (Bouchiha & Malki, 2010).

En consecuencia, debido al rápido desarrollo de aplicaciones Web, se tienen también retos en cuanto a la seguridad y reducción de vulnerabilidades, especialmente sobre datos en caché y temporales, donde se alojan datos sensibles de los usuarios. Debido a esto, la recomendación es contemplar desde el inicio el diseño y desarrollo de aplicaciones que den seguridad a los usuarios mientras hacen uso de estas. La Figura 2.9 muestra los riesgos de seguridad de una aplicación Web, en el que se involucran determinados

ataques, para los cuales se utilizan diversos recursos para causar daños a las líneas de negocio, y suprimir pérdidas en la organización (Yadav et al., 2018).

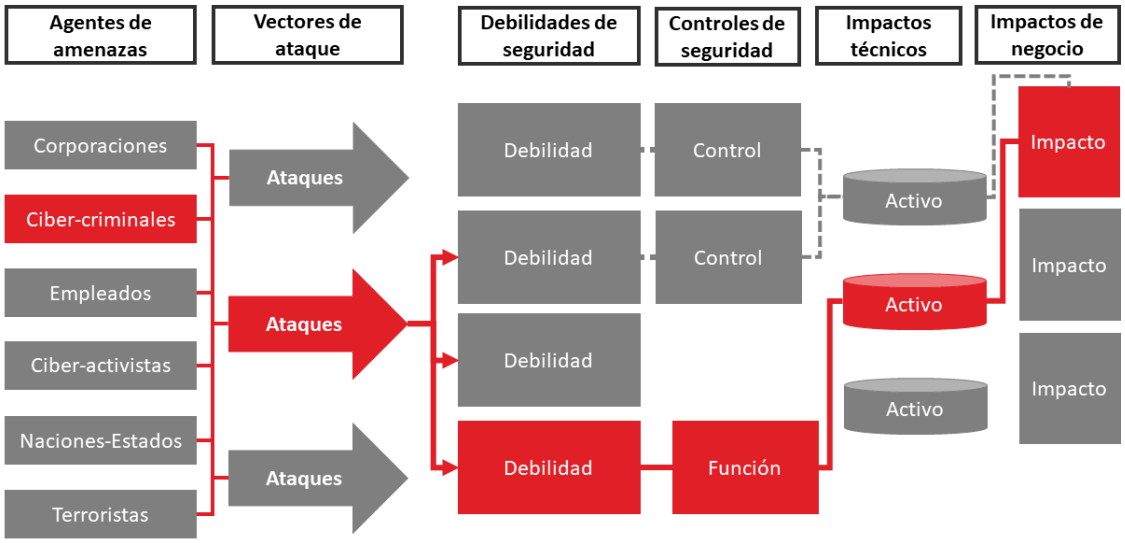

Figura 2.9: Vulnerabilidades y riesgos de seguridad que afectan a las aplicaciones Web. Adaptado de (Yadav et al., 2018).

Estos ataques enfocados en las vulnerabilidades de las aplicaciones Web, como fraudes y ataques de diversos tipos, generalmente son ejecutados por ciberdelincuentes, por lo que, es necesario tener un control de usuarios no autorizados, construyendo mecanismos de seguridad dentro de las aplicaciones Web, y proveer datos de entrada adecuados, restringiendo el uso del servidor Web, recibiendo solicitudes basadas en el protocolo HTTP y usuarios definidos para acceder a archivos en directorios raíces (*root*, por su nombre en Inglés).

2.6. Residuos sólidos urbanos

La clasificación de los residuos sólidos, respecto a sus características y origen, están delimitados por su generación dentro de los límites de una determinada zona, entre los que destacan: i) urbanos, ii) industriales; y iii) rurales. Los residuos sólidos urbanos son generados en los asentamientos poblacionales urbanos, donde se tiene dos subcategorías: residenciales (viviendas) y no residenciales. También están los de origen comercial; institucional y servicios; y construcción, demolición y especiales. Los residuos especiales son todos aquellos generados por hospitales y farmacias, los cuales tienen la clasificación

de *residuos de riesgo* (Buenrostro et al., 2001; DOF, 2003).

En este sentido, el manejo adecuado de los residuos sólidos urbanos ha sido siempre un gran problema en las grandes ciudades, y la Ciudad de México no es la excepción. En México es una necesidad creciente dar a conocer los trabajos realizados en cuanto a la gestión de los residuos sólidos urbanos. Uno de los trabajos publicados fue el del municipio de Morelia, en Michoacán, donde se tomaron muestras de los residuos provenientes de áreas residenciales y no residenciales. Con base en estas muestras se hicieron análisis sobre la eficiencia de su recolección y posterior tratamiento, seguido de mediciones que reflejan el porcentaje de lo depositado en tiraderos y áreas designadas para este proceso (Buenrostro et al., 2001).

Para el caso específico de la Ciudad de México, debido a su acelerada urbanización, en esta se generan aproximadamente 12 mil toneladas de RSU por día (generando más RSU que la ciudad de New York), siendo una de las ciudades que más produce RSU en el mundo (Moreno et al., 2012). Dos de los problemas más significativos que tiene la Ciudad de México es la incapacidad de reciclar los diversos materiales, y darle el correcto tratamiento a los materiales orgánicos que son contenidos en los RSU. Por lo que, la valorización del reciclado de diversos materiales y de los orgánicos es de alrededor de 45 %, que de acuerdo con Gutiérrez Galicia et al., 2019 este valor es el mínimo para las ciudades de ingresos económicos medios y altos.

Con base en un análisis en el 2015, sobre 33 mega ciudades en el mundo, 27 de estas presentaron problemas en la generación y manejo de residuos sólidos urbanos, las cuales fueron localizadas en países en desarrollo, como es el caso de la Ciudad de México, siendo uno de los factores más importantes la gestión de los RSU, puesto que impacta en la vida cotidiana de los ciudadanos y en la sustentabilidad que la propia ciudad (Gutiérrez Galicia et al., 2019). Además, estas grandes ciudades, como la Ciudad de México, deben tener proyectos de inversión para la construcción de vertederos y rellenos sanitarios, donde la proporción es de 10 millones de dólares para una capacidad de un millón de habitantes. Este enfoque para el manejo de RSU debe ser reflejado en la mayoría de los estados de la República Mexicana (Gutiérrez Galicia et al., 2019; Rueda-Avellaneda et al., 2021).

Por otra parte, de acuerdo con las Naciones Unidas (2020), la Ciudad de México concentra el 30 % de la población total de la República Mexicana, siendo la tercera ciudad con más aglomeración en el mundo. Además, es el centro principal de un amplio número de actividades científicas, económicas, culturales y políticas de todo el país (Gutiérrez Galicia et al., 2019). Ante estas actividades, como es de esperarse, la administración de los RSU es uno de los mayores retos para el Gobierno de la Ciudad de México, al igual que la distribución de agua potable, generación de suelos para vivienda

y negocios, entre otros. Los problemas que se generan, en términos de la administración de los RSU, son la densidad de tráfico, la cantidad insuficiente de áreas para el desecho, tratamiento y reciclaje de los RSU, y la aplicación de adecuadas normas y regulaciones ambientales (Gutiérrez Galicia et al., 2019).

Las políticas y prácticas que actualmente se ejercen en la Ciudad de México se centran en la prevención y reducción de acciones para que los RSU no terminen en las áreas de desechos, sino darles el tratamiento correspondiente (Rueda-Avellaneda et al., 2021). No obstante, la Ciudad de México refleja los siguientes problemas: debilidad institucional; complicidad en el desacato al cumplimiento de las escasas normas ambientales y de sustentabilidad; infraestructura deficiente para el transporte, tratamiento y reciclaje de los RSU; e insuficiencia en la calidad de los servicios de recolección gubernamentales (Gutiérrez Galicia et al., 2019; Muñoz-Cadena et al., 2009).

Los diferentes RSU inorgánicos que se generan en la Ciudad de México son plásticos (50 %), papel (44.5 %), textiles (0.4 %) y vidrio (0.5 %). Estos porcentajes mensuales son orientados a los RSU de casa habitación en las diferentes alcaldías de la Ciudad de México (Muñoz-Cadena et al., 2009; Rueda-Avellaneda et al., 2021). Uno de los métodos de análisis para la obtención de índices que permitan medir la relación entre la generación de RSU con los involucrados en la generación de los mismos, es el *Sistema Integrado de Administración Sustentable de Residuos* (ISWM, por sus siglas en inglés). Este sistema provee un conjunto de indicadores centrados en diferentes aspectos de la ciudad, como: ambiente, ambiente social, y aspectos legales involucrados. La Figura 2.10 muestra los componentes del Sistema Integrado de Administración Sustentable de Residuos, el cual es utilizado como indicadores en la generación de RSU en la Ciudad de México.

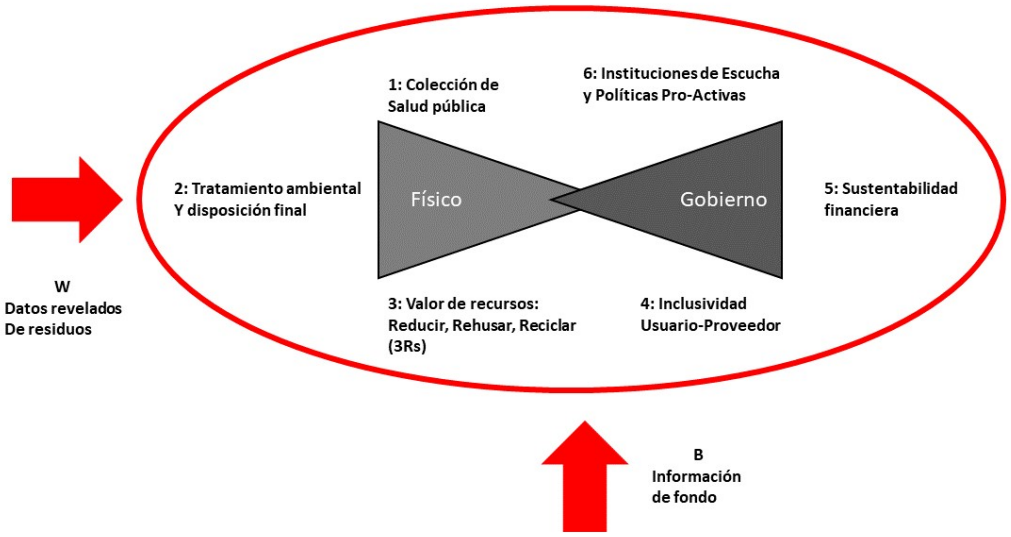

Figura 2.10: Componentes del Sistema Integrado de Administración Sustentable de Residuos. Adaptado de (Gutiérrez Galicia et al., 2019).

Este sistema fue desarrollado entre 2012 y 2013 por investigadores de diferentes instituciones, apoyados por la Corporación Alemana y el Ministerio Federal de Cooperación Económica y Desarrollo de Alemania. Los indicadores de generación y presencia de RSU proveen una visión del desempeño de las ciudades en la administración de los RSU. Los actores que se toman son los *generadores de residuos*, que se compone por pequeñas-medianas empresas y amas de casa que gobiernan la administración de los recursos en los hogares. Finalmente, se toman tres aspectos fundamentales: prevención, rehúso y reciclaje. Además, se cuenta con cinco niveles de cumplimiento: bajo = rojo (0-49 %); bajo-medio = rojo/amarillo (50-69 %); medio = naranja (70-89 %); medio-alto = amarillo/verde (90-98 %); y alto = verde (99-100 %) (Gutiérrez Galicia et al., 2019).

Con base en los indicadores del sistema, la Ciudad de México obtuvo como porcentajes de desempeño: 77.3 % de los RSU pueden ser explotados en su recolección, el 27.18 % es susceptible a recuperación o rehúso, 49.95 % es orgánico, y solo el 22.97 % no es susceptible a recuperación, rehúso o reciclado (Gutiérrez Galicia et al., 2019). Estos porcentajes demuestran la necesidad de mejorar la administración y manejo de los RSU de la Ciudad de México, debido al crecimiento poblacional y aumento en la generación de RSU; el inadecuado depósito y recuperación:; y el no cumplimiento con las regulaciones existentes para el cuidado y preservación del ambiente. Por lo que, esta situación debe ser responsabilidad de los servicios públicos y privados.

2.7. Síntesis

La adecuada administración de los residuos sólidos urbanos en la ciudad de México, y sus alcaldía, presenta diferentes retos, como: la correcta administración de los agentes involucrados en la recolección, traslado y tratamiento de los diferentes RSU. Además, la escasa información sobre la composición y tratamiento de los RSU dificulta el correcto tratamiento. Otro de los problemas es la falta de coordinación, unidades, trabajadores, planeación y gestión, que acrecientan el problema del manejo de los RSU. Por lo que, entidades como la Secretaría del Medio Ambiente de la Ciudad de México, a través de sus planes de gestión, contemplan incluir herramientas tecnológicas para la administración, traslado y tratamiento de los RSU en la Ciudad de México.

Naturalmente, una aplicación Web bajo una arquitectura de microservicios podría ofrecer ventajas en su desarrollo y escalabilidad para el usuario final, en este caso tres actores: generador, transportista y reciclador. Esta arquitectura de dominio fuertemente modulada permite escalar la aplicación Web según las necesidades de uso y petición. Puesto que los microservicios están homologados en arquitecturas de servidor de RESTful y APIs web, para cuyo desarrollo se emplea GraphQL, que es una mejora de RESTful, utilizado como lenguaje de consulta y manipulación de diferentes operadores de datos para APIs.

3
Propuesta de solución

Índice

En este capítulo se presenta la definición de requerimientos, el diseño basado en la selección de catálogos, la estructura y los diagramas de entidad-relación (ER) establecidos para la base de datos. En esta base de datos se almacenan los datos de origen o los datos relacionados con los diferentes casos de uso de la aplicación Web, denominado Plataforma SEDEMA. Además, se listan los diferentes operadores CRUD (Create, Read, Update and Delete) que fueron codificados para alimentar las funcionalidades de la aplicación.

3.1. Definición de requerimientos

Los requerimientos para el desarrollo de la aplicación Web se establecieron con base en diversas reuniones de trabajo entre el equipo de desarrollo y los representantes de la SEDEMA, quienes son los usuarios finales de la aplicación. Como resultado de estas reuniones se elaboraron los requerimientos y necesidades de software. La Tabla 3.1 resume la lista de requerimientos de usuario identificados:

Tabla 3.1: Requerimientos de usuarios identificados.

Usuario	Descripción del requerimiento
Administrador general (usuario SEDEMA)	Usuario con los permisos para acceder y administrar cualquier parte del sistema. Dar seguimiento a la información presentada en el manifiesto.
Generador	Persona física o moral que genera un promedio igual o superior a 50 kg diarios de RSU, los cuales puede ofertar o solicitar algún servicio a través de la plataforma.
Prestador de servicios	Persona física o moral que publica sus requerimientos de residuos y servicios. Puede ofertar y/o solicitar servicios o residuos para las actividades del acopio, almacenamiento y transporte. Debe contar con una autorización RAMIR vigente.

Con base en los requerimientos de usuario, se identificaron también cinco tipos de prestadores de servicio: *i*) -B1- para la recolección y transporte de los residuos hacia sitios autorizados; *ii*) -B2- para el acopio y almacenamiento temporal de los RSU en un lugar determinado; *iii*) -B3- para el reciclaje, esto es, la transformación de los materiales contenidos en los RSU a través de diferentes procesos; *iv*) -B4- para la reutilización o reuso de los RSU sin que medie en un proceso de transformación; y *v*) -B5- para el tratamiento, esto es, dedicado a procesos de transformación con posibilidad de reducir su volumen o peligrosidad.

Los requerimientos funcionales (RF) se tomaron con base en el registro, búsqueda, solicitud, confirmación y administración de la plataforma Web:

1. Registro

    a) Se debe almacenar la fecha y hora de creación, modificación o baja de usuarios. El proceso de baja de un usuario se llevará a cabo cuando ya no

cuente con autorización vigente, o por alguna situación de incumplimiento.

b) Se debe contar con un registro de eventos para la atención a fallas.

c) Se hará el registro de los usuarios: Generadores y/o Prestadores de servicio; solicitando el número de registro regulatorio vigente (número LAU-CDMX, Planes de Manejo y/o RAMIR), nombre del representante legal, razón social, ubicación, teléfono, correo electrónico y tipo de usuario.

d) El instrumento regulatorio se validará a través de una API proporcionada por la SEDEMA, en la cual se regresará la vigencia de dicho instrumento regulatorio.

e) La plataforma de encadenamiento productivo, permitirá a los usuarios Generadores y/o Prestadores de servicio realizar las actividades de búsqueda, notificaciones, ofertas, solicitudes o manifiestos dentro de la plataforma, según corresponda al instrumento regulatorio. En el caso de que un usuario, cuente con uno o más permisos, y ninguno de estos esté vigente, la plataforma no le permitirá el acceso hasta que se renueve el permiso en la SEDEMA.

f) Se hará el registro, actualización y cancelación de oferta de material por el Generador y/o Prestador de servicio, donde se especifique la actividad que puede realizar conforme a sus permisos de registros ambientales.

g) Para el registro de 'Oferta de residuos' o 'Servicios' se requiere el tipo de material y cantidad, o tipo de servicio que se oferta. También, la ubicación geográfica de donde se requiere el mismo.

h) Los residuos y servicios que son candidatos son únicamente los validados por la LAU-CDMX (grandes generadores), Plan de Manejo y RAMIR (prestadores de servicios).

i) Una misma empresa registrada en la plataforma puede tener funciones de Solicitante, Ofertante y Transportista.

j) Las empresas que se registren como Prestadores de servicios, deben contar con la autorización RAMIR y/o Plan de Manejo vigente, según aplique.

k) La interfaz gráfica de usuario debe estar apegada al manual de identidad institucional del Gobierno de la Ciudad de México del periodo 2021-2024, previamente proporcionado.

2. Búsqueda

a) Búsqueda de residuos en el registro de ofertas.

b) Solicitud del residuo que se desea adquirir o solicitar.

c) Mostrar tipo y cantidad de residuos ofertados.

d) Ubicación física del residuo ofertado.

e) Tiempo de oferta del RSU.

3. Solicitud

   a) Se realizará la solicitud del pedido y se solicitará la confirmación por parte del Ofertador.

4. Confirmación

   a) Se realizará la confirmación o rechazo de la solicitud. En caso de que sea rechazada, describir la razón. En caso de que se confirme el interés por parte del Ofertante y el Solicitante, establecer el día y la hora en que se requiere el servicio, así como los requisitos de acceso a las instalaciones.

5. Administración

   a) El administrador (SEDEMA) solicita la generación del informe donde se muestre los usuarios que interactúan (empresas), la cantidad y el tipo de residuo o servicio, ya con la confirmación final, la ubicación de procedencia y el destino final, o el lugar donde se presta el servicio.

   b) Para la confirmación y reporte de la interacción, se generará el Manifiesto de Entrega-Recepción con los siguientes datos: datos de la empresa generadora, residuos generados, datos de la empresa transportista, fecha de recolección, empresa solicitante, residuos recibidos, fecha de recepción y destino.

   c) Los administradores tendrán la posibilidad de descargar una base de datos con la información generada por los usuarios.

   d) Para la confirmación de la transacción se requiere:

      – La generación del reporte de Manifiesto de Entrega-Recepción.
      – La evidencia fotográfica de las actividades de Entrega-Recepción.

   e) Alertas de correo electrónico a la SEDEMA:

      – Nuevas ofertas de producto, donde se indique el tipo de residuo que se oferta, la ubicación y la cantidad.
      – Nuevas solicitudes de producto, donde se indique el tipo de residuo que se solicita, la ubicación y la cantidad.
      – Nuevas ofertas de servicio de transporte y aprovechamiento.
      – Recordatorio a los 20 días de realizar una transacción para adjuntar el manifiesto físico.

   f) Generación de estadísticas mediante filtros de los residuos vinculados conforme ciertos periodos para la generación de reportes.

   g) Manual para cada tipo de usuario.

Los requerimientos no funcionales (RNF) que se identificaron en función de las reuniones de trabajo fueron:

1. Capacidad para 5000 usuarios con opción a escalabilidad en el transcurso del tiempo.
2. Seguridad en el intercambio de datos personales en el proceso de interacción de solicitud y confirmación.
3. Manejo de la información personal mediante acuerdos de confidencialidad.
4. El manejo de los RSU será concedido únicamente a personas físicas y morales que se encuentren autorizadas por la SEDEMA.
5. La plataforma debe estar disponible en tiempo y forma según se solicite durante la etapa de desarrollo.
6. Para la gestión de la información, la plataforma debe de incluir un sistema de base de datos.
7. Se emplearán como herramientas de desarrollo: Go, Flutter, Dart y Docker, entre otros.

Con base en la definición de los requerimientos mencionados, se hizo una descomposición de tareas a nivel del equipo de trabajo, mediante la cual se asignaron actividades a cada integrante del proyecto; quedando del lado del equipo de la Facultad de Ingeniería de la UNAM, la implementación de determinadas funcionalidades del lado del servidor (Back-End). Para el caso particular de este trabajo de tesis, se asignó la codificación y pruebas de las funciones CRUD (Create, Read, Update and Delete) para determinados catálogos de la base de datos, como: catálogos generales (cg) y catálogos específicos para procesos (cpi, cpr, p1 y p2).

3.2. Diseño del Back-End

La aplicación Web para la gestión de los RSU está basada en una arquitectura de microservicios. Esta arquitectura proporciona una serie de ventajas, las cuales se describieron en la Sección 2.4, del Capítulo 2. Algunas ventajas son la independencia de la tecnología, fronteras de ejecución por cada microservicio, una base de datos compartida con esquemas exclusivos por cada microservicio, entre otras.

Esta arquitectura permite la fácil escalabilidad, permitiendo reducir el riesgo de errores del sistema en el tiempo de ejecución, y escalar según sea la necesidad de la

aplicación, ya sea de forma vertical u horizontal. Con base en los requerimientos, para el desarrollo de esta aplicación se eligió la escalabilidad horizontal, permitiendo agregar servidores y contenedores de acuerdo al número de usuarios al realizar peticiones por cada unidad de tiempo.

En este sentido, la propuesta para el desarrollo de la aplicación Web fue en dos capas (Figura 3.1), donde la interna hace referencia a los sistemas operativos utilizados en los dispositivos de cómputo, y la externa porta el ambiente del servidor y las tecnologías, conocido como Back-End. El sistema operativo instalado en el ambiente de desarrollo fue GNU/Linux (Ubuntu), en el cual se ejecutan los contenedores que provee Docker para que la aplicación Web no dependa del sistema operativo local. En el mismo sentido, para la gestión y ejecución de la base de datos se utilizó PostgreSQL, aprovechando el costo cero del mismo, y como lenguaje de programación se utilizó Go.

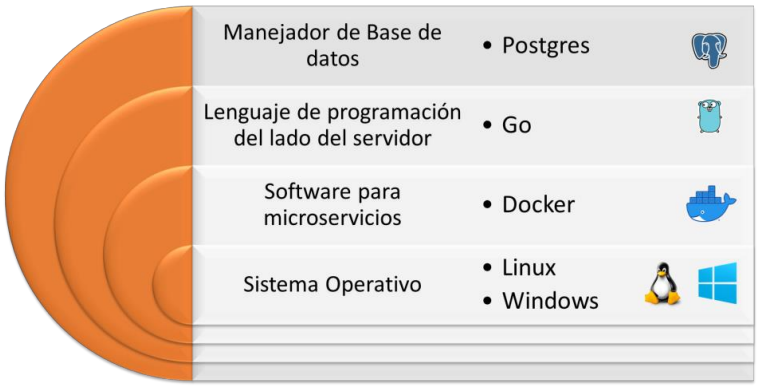

Figura 3.1: Esquema del diseño del Back-End, basado en dos capas. Interna que contiene al sistema operativo, y externa, que es el ambiente en Docker.

La ventaja de utilizar Go como lenguaje de programación se debe a sus características en el desarrollo de software, esto es, es estructurado, seudo-orientado a objetos y funcional. Además, posee características avanzadas que permiten usar una sintaxis limpia y ordenada, debido a que es un lenguaje para su ejecución en múltiples plataformas, con una amplia enfocada documentación de sus bibliotecas y su enfoque de buenos principios en el desarrollo de software (Doxsey, 2016).

En el caso de PostgreSQL, como manejador de bases de datos, este es una de las primeras opciones de dearrollo en la industria y el gobierno, debido a que es completamente orientado a bases de datos relacionales y libre de costo. Además, PostgresSQL soporta diferentes lenguajes de programación, lo que incrementa las características para trabajar con bases de datos. Asimismo, al ser de código abierto, se le ha otorgado una serie de reconocimientos como: elección del editor del Linux Journal como mejor base de datos en tres ocasiones: 2000, 2003, y 2004 (Matthew & Stones,

2005).

Para la administración de los contenedores se eligió Docker (Figura 3.2), el cual es un programa de línea de comandos (en segundo plano), y un conjunto de servicios remotos que, a través de un enfoque logístico, simplifican la instalación, ejecución, lanzamiento y desinstalación de software. Docker usa tecnología nativa de UNIX, denominada 'contenedores' (Nickoloff, 2016). El término contenedor es nativo de UNIX. Históricamente, los sistemas operativos basados en UNIX usan el término *jail* (cárcel, por su traducción al español) para describir un ambiente de ejecución para un determinado programa, este entorno previene que algún programa pueda acceder a recursos protegidos o privilegiados del sistema operativo.

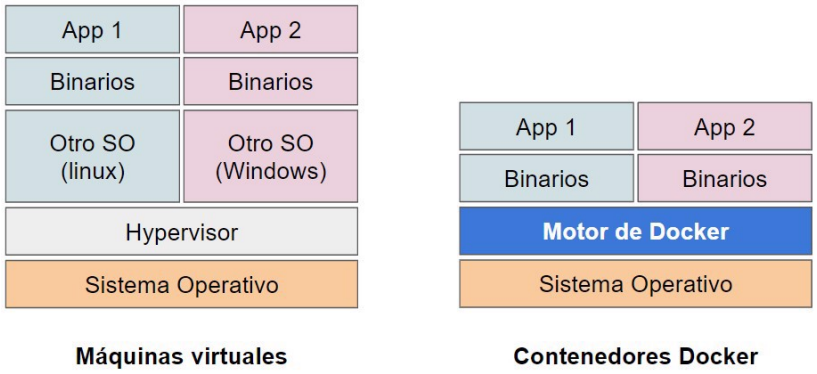

Figura 3.2: Motor de Docker para el aislamiento y ejecución de los programas alojados en los contenedores.

Por otro lado, el equipo de desarrollo propuso trabajar bajo el estándar europeo *Git-Flow o GitFlow*, el cual es una estrategia de ramificaciones que puede ser implementado en el desarrollo de software (Westby, 2015). Inicialmente, para el proyecto se trabajó con una única rama denominada *develop*. Desde esa rama, los desarrolladores crearon ramas divergentes para añadir características de funcionalidad a la aplicación (ver Figura 3.3).

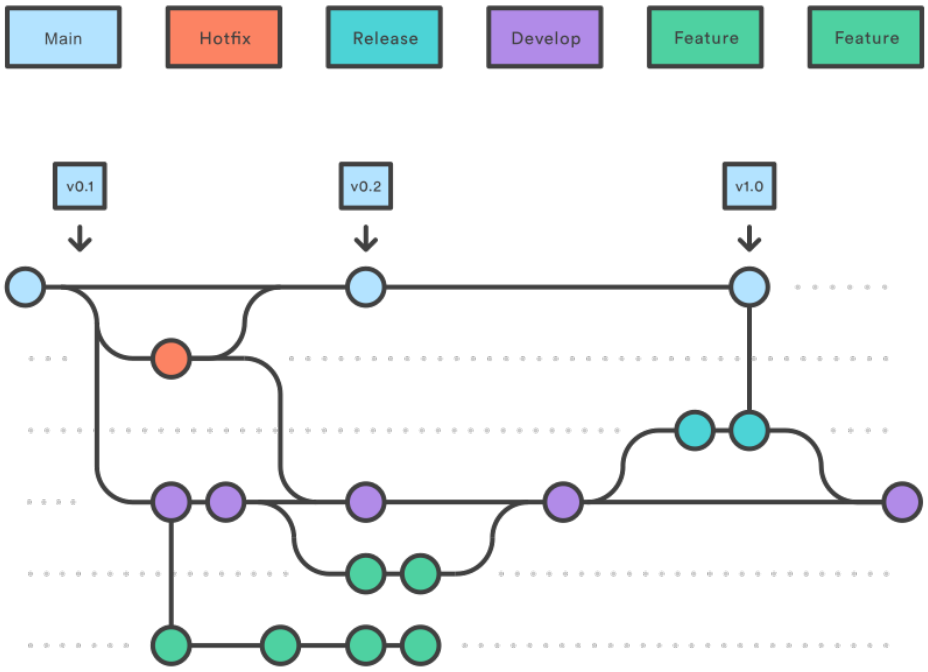

Figura 3.3: Diagrama de GitFlow utilizado para el control de trabajo en el desarrollo del proyecto.

Las ramas *feature* (característica, por su traducción al español) fueron destinadas para una refactorización o una característica completamente nueva. Además, las ramas destinadas a la corrección de errores de código reciben el nombre de *hotfix*. Mientras que para la publicación de un lanzamiento, se creó una rama de nombre *release* a partir de la rama *develop*, donde se concentra el trabajo del equipo *feature*. De modo que, la rama de *release* se fusiona en la rama *main*, la cual está destinada para el ambiente de producción, donde se agregó una etiqueta con la versión del lanzamiento y una descripción que identifique a la nueva versión.

3.3. Diagramas ER de los catálogos

Como parte de la construcción de la aplicación Web, se definieron algunos catálogos como parte del diseño de modelo de bases de datos. Así, como base en el diagrama de entidad-relación (ER), se definieron una serie de catálogos que fueron destinados por el equipo de desarrollo para su codificación. Estos catálogos fueron de dos tipos: catálogos generales (cg) y catálogos específicos para procesos (cpi, cpr, p1 y p2).

Estos diferentes catálogos fueron agrupados de acuerdo con la función para la cual fueron diseñados. Para el caso de catálogos generales se codificaron: cg_usuarios_tipo

(Figura 3.4), cg_estados (Figura 3.5), cg_calles (Figura 3.6), y cg_codigos_postales (Figura 3.7).

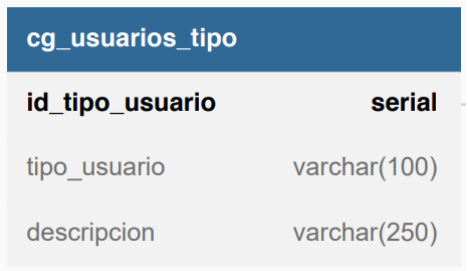

Figura 3.4: Catálogo general para usuarios tipo.

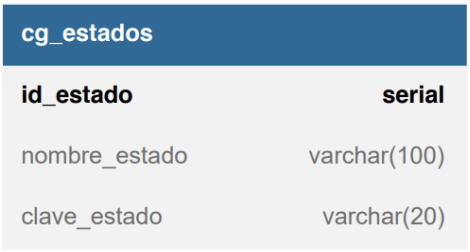

Figura 3.5: Catálogo general para estados.

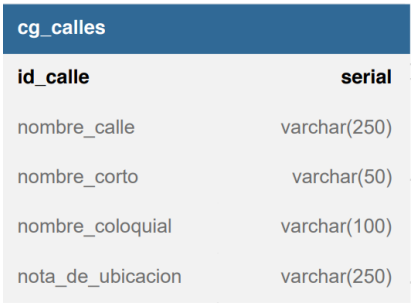

Figura 3.6: Catálogo general para calles.

Figura 3.7: Catálogo general para códigos postales.

Por otro lado, los catálogos de procesos que se requirieron para la interfaz gráfica de usuario, donde se ejecuten acciones de crear, consultar, actualizar o eliminar, forman parte de los operadores CRUD. Estos operadores provienen del acrónimo en inglés Create, Read, Update, and Delete. Por lo que, se codificaron los siguientes catálogos: cpi_roles (Figura 3.8), cpi_empresas (Figura 3.9), cpi_empresas_identificadores (Figura 3.10), cpr_estatus_procesos (Figura 3.11), p1_ofertas (Figura 3.12), y p2_ofertas_solicitantes_candidatos (Figura 3.13).

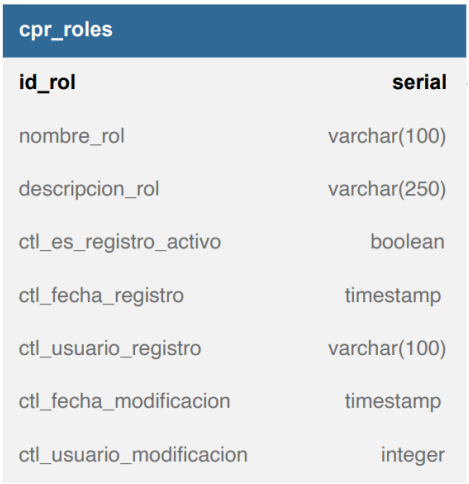

Figura 3.8: Catálogo de procesos para roles.

cpi_empresas	
id_empresa	**serial**
razon_social	varchar(250)
RFC_empresa	varchar(100)
es_matriz_o_sucursal	integer
id_empresa_global	integer
nombre_empresa_sucursal	varchar(250)
representante_gerente_contacto	varchar(250)
horario_atencion	varchar(100)
notas_descripcion	varchar(250)
tipo_empresa_fiscal	varchar(50)
numreg_LAU_CDMX	varchar(100)
numreg_RAMIR	varchar(100)
numreg_plan_manejo	varchar(100)
numreg_ambiental	varchar(100)
numreg_LAU_CDMX_vigencia	date
numreg_ramir_vigencia	date
numreg_plan_manejo_vigencia	date
numreg_ambiental_vigencia	date
ctl_es_registro_activo	boolean
ctl_fecha_registro	timestamp
ctl_usuario_registro	text

Figura 3.9: Catálogo de procesos para empresas.

cpi_empresas_identificadores	
id_empresa_identificador	**serial**
id_empresa	integer
texto_identificador	varchar(250)
tipo_identificador	varchar(100)
otro_tipo_identificador	varchar(100)
ctl_es_registro_activo	boolean
ctl_fecha_registro	timestamp
ctl_usuario_registro	varchar(100)
ctl_fecha_modificacion	timestamp
ctl_usuario_modificacion	integer

Figura 3.10: Catálogo de procesos para identificadores de empresas.

cpr_estatus_procesos	
id_estatus	**varchar(20)**
descripcion	varchar(250)
codigo_estatus	integer

Figura 3.11: Catálogo para los estatus de los procesos.

p1_ofertas	
id_oferta	**serial**
tipo_oferta	integer
id_empresa	integer
id_rol_empresa	integer
titulo_oferta	varchar(250)
descripcion	varchar(250)
nota_descripcion	varchar(250)
id_estatus	varchar(20)
fecha_inicio_oferta	timestamp
fecha_fin_oferta	timestamp
id_domicilio_origen	integer
id_residuo_autorizado	integer
residuo_cantidad	integer
residuo_unidad	integer
id_servicio	integer
ctl_es_registro_activo	boolean
ctl_fecha_registro	timestamp
ctl_usuario_registro	varchar(100)
ctl_fecha_modificacion	timestamp
ctl_usuario_modificacion	integer

Figura 3.12: Catálogo de procesos para las ofertas.

La codificación de los catálogos sufrieron varias iteraciones, en función de los

p2_ofertas_solicitantes_candidatos	
id_ofertas_solicitante_candidato	**serial**
constraint	p2_ofertas_solicitantes_candidatos_pkey
id_oferta	integer
id_empresa_solicitante	integer
id_empresa_ofertante	integer
fecha_oferta_solicitud	timestamp
id_estatus	varchar(20)
notas_observaciones	varchar(250)
ctl_es_registro_activo	boolean
ctl_fecha_registro	timestamp
ctl_usuario_registro	varchar(100)
ctl_fecha_modificacion	timestamp
ctl_usuario_modificacion	integer

Figura 3.13: Catálogo para las ofertas relacionadas a los solicitantes y candidatos.

requerimientos y solicitudes de la SEDEMA. Estos cambios fueron realizados a lo largo de abril a noviembre de 2022. Un ejemplo de estos cambios fue en catálogo de procesos empresas (Figura 3.9), en el cual se añadieron los campos: *rolEmpresaEsSolicitante*, *rolEmpresaEsGenerador*, *rolEmpresaEsPrestadorservicios*, *tipoEmpresaEsPersonafisica* y *tipoEmpresaEsPersonamoral*. En este trabajo se utilizó la notación *camel case* para hacer referencia al contenido del código, mientras que en el código del proyecto se utilizó la notación *snake case*.

Esta diferencia de las dos notaciones se debe a que al iniciar una nueva palabra en 'camel case' se utiliza la letra mayúscula de la siguiente palabra, mientras que en 'snake case' se utiliza un guión bajo como separador de palabras, las cuales todas son minúsculas. Esta diferencia entre ambas notaciones hacen referencia a los diferentes estilos de programación en la industria, y a las diferentes políticas de desarrollo existentes.

3.4. Desarrollo de los operadores CRUD

El desarrollo (codificación) de los operadores, asociados a cada catálogo designado, fue mediante GraphQL, el cual es un lenguaje de consulta y manipulación de datos

para APIs. GraphQL fue desarrollado por Facebook en 2012 y liberado para el público en 2015. En este sentido, la Tabla 3.2 resume la lista de operadores CRUD codificados para catálogos generales (cg). Mientras que la Tabla 3.3 muestra la lista de operadores CRUD codificados para los catálogos específicos para procesos (cpi, cpr y p1 y p2). En ambos casos se indica, el operador de lectura, escritura, actualización o borrado. Además, en el Anexo A se muestra información ampliada sobre las APIs codificadas, las cuales alimentan las funcionalidades que recibe el servidor al ser ejecutadas.

Tabla 3.2: Operadores codificados para los catálogos generales.

Item	**Catálogo**	**Tipo de operador**	**Operador**
1	cg_usuarios_tipo	Lectura	a) GetUsuarioTipo (idTipoUsuario: ID, tipoUsuario: String, descripcion: String) b) GetAllUsuariosTipo()
2	cg_estados	Lectura	a) GetEstado (idEstado: ID, nombreEstado: String, claveEstado: String) b) GetAllEstados()
3	cg_calles	Lectura	a) GetCalle (idCalle: ID, nombreCalle: String, nombreCorto: String, nombreColoquial: String)
4	cg_codigos_postales	Lectura	a) GetCodigoPostal (idCpInterno: ID!) b) GetCodigoPostalPorStr (idCp: String!) c) GetAllCodigosPostales()

Tabla 3.3: Operadores codificados para los catálogos específicos para procesos.

Item	Catálogo	Tipo de operador	Operador
1	cpi_empresas_tipo	Lectura	a) GetEmpresa (idEmpresa: ID, rfcEmpresa: String) b) GetAllEmpresas()
		Escritura	a) CreateEmpresa (input: EmpresaCreateInput!)
		Actualización	a) UpdateEmpresa (idEmpresa: ID!, input: EmpresaUpdateInput!)
		Borrado	a) DeleteEmpresa (idEmpresa: ID!)
2	cpr_roles	Lectura	a) GetRol (idRol: ID, nombre: String) b) GetAllRoles()
3	cpi_identificadores	Lectura	a) GetEmpresaIdentificador (idEmpresaIdentificador: ID) b) GetEmpresaIdentificadoresPorEmpresa (idEmpresa: ID)
		Escritura	a) CreateEmpresaIdentificador (input: EmpresasIdentificadoresInput!)
		Actualización	a) UpdateEmpresaIdentificador (idEmpresaIdentificador: ID!, input: EmpresasIdentificadoresInput!)
		Borrado	a) DeleteEmpresaIdentificador (idEmpresaIdentificador: ID!)
4	cpr_estatus_procesos	Lectura	a) GetEstatusProceso (idEstatus: String)
		Actualización	a) UpdateEstatusProceso (idEstatus: String!, input: EstatusProcesoInput!)
		Borrado	a) DeleteEstatusProceso (idEstatus: String!)
5	p1_ofertas	Lectura	a) GetOfertas (idOferta: ID!) b) GetOfertaPorResiduo (idResiduoAutorizado: ID!) c) GetOfertaPorServicio (idServicio: ID!)
		Escritura	a) CreatePOferta (input: P1OfertasInput!)
		Actualización	a) UpdatePOferta (idOferta: ID!, input: P1OfertasInput!)
		Borrado	a) DeletePOferta(idOferta: ID!)
6	p2_ofertas_solicitantes	Lectura	a) GetOfertaSolicitudesCandidatos (idOfertasSolicitanteCandidato: ID!) b) GetOfertaSolicitudesCandidatosPorOferta(idOferta: ID!)
		Escritura	a) CreateOfertaSolicitantesCandidatos (input: P2OfertasSolicitantesCandidatosInput!)
		Actualización	a) UpdateOfertaSolicitudCandidato (idOfertasSolicitanteCandidato: ID!, input: P2OfertasSolicitantesCandidatosInput!)
		Borrado	a) DeleteOfertaSolicitanteCandidato (idOfertasSolicitanteCandidato: ID!)

Para la codificación de los operadores en GraphQL fue necesario tener en cuenta cuatro alcances: el primero fue *schema.graphqls*, donde se alojan los prototipos para cada uno de los catálogos, usando la sintaxis de GraphQL; el segundo fue *query.graphqls*, el cual está emparejado con *query.resolvers.go*; el tercero fue *mutation.graphqls*, que está emparejado con *mutation.resolvers.go*; y el cuarto fue *resolver.go*, el cual es una estructura general que almacena las dependencias de todos los catálogos con sus respectivas APIs. Estas dependencias se encuentran alojadas en el repositorio de la aplicación Web, en el directorio raíz `~\app\repository\`. En el Anexo C se muestra información ampliada sobre la implementación de un servidor local en Go.

Se desarrolló también el microservicio para el registro del domicilio de una empresa, para esto se utilizó la API de geolocalización de COPOMEX (Código Postal Mexicano), la cual se emplea para la ubicación, búsqueda y manejo de códigos postales, estados, municipios y colonias en México.

3.5. Pruebas de funcionalidad

Para probar el funcionamiento de los operadores desarrollados y posteriormente puedan ser utilizados en forma de APIs, se utilizó como herramienta de pruebas Postman (ver Figura 3.14), la cual es una plataforma que permite gestionar y probar los diferentes operadores en forma de APIs.

Así, se probaron todos los operadores desarrollados, con base en los catálogos designados (Tablas 3.2 y 3.3, mostrados en la sección anterior): a) generales (tipo de usuario, estados, calles y códigos postales); y b) específicos de procesos (tipo de empresas, roles, identificadores, estatus de procesos, ofertas y ofertas solicitantes).

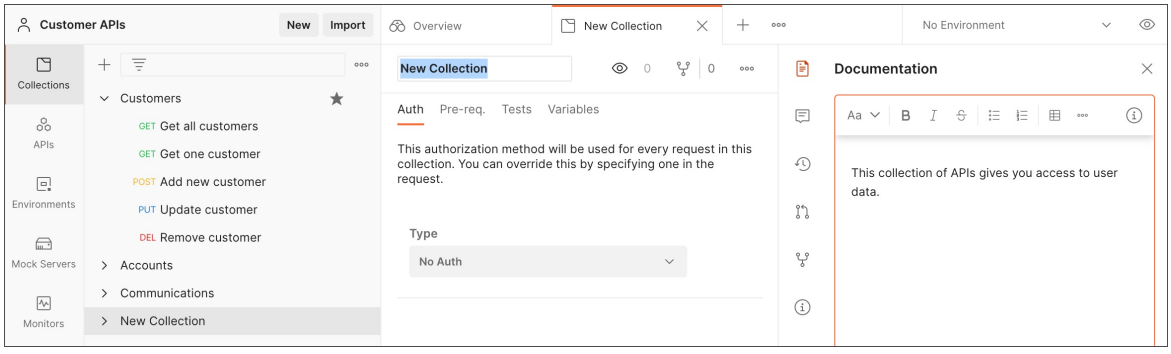

Figura 3.14: Interfaz de Postman.

3.6. Síntesis

La codificación de los operadores de los catálogos seleccionados, generales y específicos de procesos, se basó en estructurar el servidor de GraphQL para la ejecución de las consultas que se realizan a través de la comunicación basada en RESTful. Además, los operadores que se codificaron se fueron actualizando con base en las iteraciones del proyecto, y las definiciones de los requerimientos de los usuarios. Todas codificaciones fueron probadas a través de Postman y quedaron en la rama de *develop*.

Por otro lado, conforme se vayan agregando dependencias o más catálogos, será necesario el realizar la inyección de dependencias y modelos en los ambientes de schema, modelos, repositorio y el manejador del servidor.

4
Resultados

Índice

En este capítulo se presentan los resultados obtenidos con base en la ejecución de las actividades que se desarrollaron como parte de la propuesta de solución, esto es, desde la definición de requerimientos, el diseño del Back-End, la definición de catálogos del diagrama ER, el desarrollo de los operadores CRUD de acuerdo a los catálogos seleccionados, y las pruebas de funcionalidad.

4.1. Resultados obtenidos

Con base en la codificación de los operadores CRUD, basado en la lista de catálogos seleccionados, presentados en la subsección 3.4, estos permitieron la gestión de los datos y de algunas funcionalidades que fueron definidos como parte de los requerimientos funcionales, por ejemplo, iniciando con la verificación de la existencia de la empresa, la cual tendrá alguno de los roles definidos: 'Generador', 'Transportista' o 'Prestador de Servicios'. Posteriormente, se tiene el registro, validación y gestión de las diferentes

necesidades que la empresa pueda presentar, ya sea, desde prestar un servicio de tratamiento, recolección o transporte, hasta la publicación de algún RSU que la empresa requiera procesar, transportar o tratar.

4.1.1. Operadores CRUD

A manera de ejemplo, en esta sección se muestra los resultados de la codificación de dos (2) de los diez (10) catálogos que se desarrollaron. En el Anexo A se muestra la codificación de los demás los operadores desarrollados. En este sentido, los operadores que se muestran son una de catálogo general (cg_usuarios_tipo) y otra del catálogo específico para procesos (cpi_empresas_tipo).

Para el caso del tipo de usuarios (cg_usuarios_tipo) se muestra la estructura que modela al catálogo proveniente de la base de datos (Código 4.1) y la codificación esquemática para su operación en el servidor (Código 4.2). Además, se muestra la API que es consumida del lado del cliente (Código 4.3).

```
1     package models
2
3     type Cg_Usuarios_tipo struct {
4     Id_Tipo_Usuario int  `json:"id_tipo_usuario" gorm:"primary_key"`
5     Tipo_Usuario  string `json:"tipo_usuario"`
6     Descripcion   string `json:"descripcion"`
7     }
```

Código 4.1: Modelo para el catálogo general de usuarios tipo.

```
1     type Cg_usuarios_tipo {
2     id_tipo_usuario: ID!
3     tipo_usuario: String!
4     descripcion: String
5     }
```

Código 4.2: Esquema (schema.graphqls) del catálogo general usuarios tipo.

```
1     # cg_usuarios_tipo
2     GetUsuarioTipo(id_tipo_usuario: ID, tipo_usuario: String,
          descripcion: String): [Cg_usuarios_tipo!]!
3
4     GetAllUsuariosTipo: [Cg_usuarios_tipo!]
```

Código 4.3: API (query.graphqls) para el catálogo general usuarios tipo.

Por otra parte, se definieron también los operadores del catálogo (Código 4.4), tanto las firmas como las definiciones de los operadores, las cuales fueron a través de una interfaz gráfica que son operadas en el servidor de GraphQL. Finalmente, se muestran los resolutores del catálogo (Código 4.5), los cuales son los que ejecutan y envían las respuestas, dependiendo de la API ejecutada.

```
1   package repository
2
3   import (
4   "apiCatalogosGenerales/graph/model"
5   "gorm.io/gorm")
6
7   type Usuarios_TipoRepository interface {
8   //CreateUsuarioTipo(newUsuarioTipo *model.CgUsuariosTipo) (*models.
        Cg_Usuarios_tipo, error)
9   GetUsuarioTipo(id int, tipo_usuario string, descripcion string) ([]*
        model.CgUsuariosTipo, error)
10  GetAllUsuariosTipo() ([]*model.CgUsuariosTipo, error)
11  }
12  type Usuario_TipoService struct {DB *gorm.DB}
13  var _ Usuarios_TipoRepository = &Usuario_TipoService{}
14
15  func NewUsuario_TipoService(db *gorm.DB) *Usuario_TipoService {
16      return &Usuario_TipoService{
17          DB: db,}}
18
19  func (u *Usuario_TipoService) GetUsuarioTipo(id int, tipo_usuario
        string, descripcion string) ([]*model.CgUsuariosTipo, error) {
20  tipos_usuario := []*model.CgUsuariosTipo{}
21  err := u.DB.Table("cg_usuarios_tipo").Where(
22  "id_tipo_usuario = ?", id).Or(
23  "tipo_usuario = ?", tipo_usuario).Or(
24  "descripcion = ?", descripcion,).Find(&tipos_usuario).Error
25  return tipos_usuario, err}
26
27  func (u *Usuario_TipoService) GetAllUsuariosTipo() ([]*model.
        CgUsuariosTipo, error) {
28  tipos_usuario := []*model.CgUsuariosTipo{}
29  err := u.DB.Table("cg_usuarios_tipo").Find(&tipos_usuario).Error
30
31  return tipos_usuario, err
32  }
```

Código 4.4: Cuerpo de las consultas del catálogo general usuarios tipo.

```
1     func (r *queryResolver) GetUsuarioTipo(ctx context.Context,
          idTipoUsuario *int, tipoUsuario *string, descripcion *string)
          ([]*model.CgUsuariosTipo, error) {
2     idTipoUsuario = FillZero(idTipoUsuario)
3     tipoUsuario = FillNA(tipoUsuario)
4     descripcion = FillNA(descripcion)
5       tipo_usuario, err := r.Usuarios_tipo.GetUsuarioTipo(*idTipoUsuario,
            *tipoUsuario, *descripcion)
6         if err != nil {
7             return nil, err
8             }
9
10        return tipo_usuario, nil
11        }
12
13    func (r *queryResolver) GetAllUsuariosTipo(ctx context.Context) ([]*
          model.CgUsuariosTipo, error) {
14        usuarios_tipo, err := r.Usuarios_tipo.GetAllUsuariosTipo()
15            if err != nil {
16                return nil, err
17                }
18        return usuarios_tipo, nil
19    }
```

Código 4.5: Resolutores para los operadores del catálogo general usuarios tipo.

Para el caso del catálogo específico para procesos cpi_empresas_tipo se muestra también la estructura que modela al catálogo proveniente de la base de datos (Código 4.6) y la codificación esquemática del catálogo (Código 4.7) para su operación en el servidor. Además, se muestra la API que es consumida del lado del cliente (Código 4.8).

```
1     package models
2     import (
3     "github.com/guregu/null"
4     "time")
5
6     type Cpi_empresas struct {
7         Id_empresa  int `json:"id_empresa" gorm:"primaryKey"`
8         Razon_social string `json:"razon_social"`
9         Rfc_empresa string `json:"rfc_empresa"`
10        Es_matriz_o_sucursal int `json:"es_matriz_o_sucursal"`
11        Id_empresa_globa null.Int `json:"id_empresa_global"`
12        Nombre_empresa_sucursal null.String `json:"
```

```
            nombre_empresa_sucursal"`
13          Representante_gerente_contacto null.String `json:"
            representante_gerente_contacto"`
14          Horario_atencion null.String `json:"horario_atencion"`
15          Notas_descripcio null.String `json:"notas_descripcion"`
16          Numreg_lau_cdmx null.String `json:"numreg_lau_cdmx"`
17          Numreg_ramir null.String `json:"numreg_ramir"`
18          Numreg_plan_manejo null.String `json:"numreg_plan_manejo"`
19          Numreg_ambiental null.String `json:"numreg_ambiental"`
20          Numreg_lau_cdmx_vigencia null.Time `json:"
            numreg_lau_cdmx_vigencia"`
21          Numreg_ramir_vigencia null.Time `json:"numreg_ramir_vigencia"`
22          Numreg_plan_manejo_vigencia null.Time `json:"
            numreg_plan_manejo_vigencia"`
23          Numreg_ambiental_vigencia null.Time `json:"
            numreg_ambiental_vigencia"`
24          Ctl_es_registro_activo bool `json:"ctl_es_registro_activo"`
25          Ctl_fecha_registro time.Time `json:"ctl_fecha_registro"`
26          Ctl_usuario_registro string `json:"ctl_usuario_registro"`
27          Ctl_fecha_modificacion time.Time `json:"ctl_fecha_modificacion"`
28          Ctl_usuario_modificacion int `json:"ctl_usuario_modificacion"`
29          Rolempresa_es_solicitante bool `json:"rolempresa_es_solicitante"`
            Rolempresa_es_generador bool `json:"rolempresa_es_generador"`
30          Rolempresa_es_prestadorservicios bool `json:"
            rolempresa_es_prestadorservicios"`
31          Tipoempresa_es_personafisica bool `json:"
            tipoempresa_es_personafisica"`
32          Tipoempresa_es_personamoral bool `json:"
            tipoempresa_es_personamoral"`
33          }
```

Código 4.6: Modelo para el catálogo de procesos de empresas.

```
1     type Cpi_empresas{id_empresa: ID!
2     razon_social: String!
3     rfc_empresa: String!
4     es_matriz_o_sucursal: Int
5     id_empresa_global: Int
6     nombre_empresa_sucursal: String
7     representante_gerente_contacto: String
8     horario_atencion: String
9     notas_descripcion: String
10    numreg_lau_cdmx: String
11    numreg_ramir: String
12    numreg_plan_manejo: String
```

```
13      numreg_ambiental: String
14      numreg_lau_cdmx_vigencia: String
15      numreg_ramir_vigencia: String
16      numreg_plan_manejo_vigencia: String
17      numreg_ambiental_vigencia: String
18      ctl_es_registro_activo: Boolean!
19      ctl_fecha_registro: Timestamp!
20      ctl_usuario_registro: String
21      ctl_fecha_modificacion: Timestamp
22      ctl_usuario_modificacion: Int
23      rolempresa_es_solicitante: Boolean
24      rolempresa_es_generador: Boolean
25      rolempresa_es_prestadorservicios: Boolean
26      tipoempresa_es_personafisica: Boolean
27      tipoempresa_es_personamoral: Boolean
28      }
```

Código 4.7: Esquema (schema.graphqls) del catálogo de procesos de empresas

```
1       GetEmpresa(id_empresa: ID, rfc_empresa: String): Cpi_empresas!
2       GetAllEmpresas: [Cpi_empresas!]!
3       CreateEmpresa(input: EmpresaCreateInput!): ID!
4       DeleteEmpresa(id_empresa: ID!): String!
5       UpdateEmpresa(id_empresa: ID!, input: EmpresaUpdateInput!): String!
```

Código 4.8: API (query.graphqls) para el catálogo de procesos de empresas.

Se definieron también los operadores del catálogo (Código 4.9), tanto las firmas como las definiciones de los operadores, las cuales fueron a través de una interfaz gráfica que son operadas en el servidor de GraphQL. Se muestran también los resolutores del catálogo (Código 4.10), los cuales son los que ejecutan y envían las respuestas, dependiendo de la API ejecutada.

```
1       package repository
2
3       import (
4           "apiCatalogosGenerales/app"
5           "apiCatalogosGenerales/app/models"
6           "apiCatalogosGenerales/graph/model"
7           "errors"
8           "github.com/guregu/null"
9           "gorm.io/gorm"
10          "time"
11          )
12
```

```
13          type EmpresasRepository interface {
14              CreateEmpresa(input *model.EmpresaCreateInput, tx *gorm.DB)
                    (*models.Cpi_empresas, error)
15              GetEmpresa(id_empresa int, rfc_empresa string) (*model.
                    CpiEmpresas, error)
16              GetAllEmpresas() ([]*model.CpiEmpresas, error)
17              UpdateEmpresa(empresaInput *model.EmpresaUpdateInput,
                    id_empresa int) error
18              DeleteEmpresa(id_empresa int) error
19          }
20
21          type EmpresaService struct {
22                  DB *gorm.DB
23          }
24
25          var _ EmpresasRepository = &EmpresaService{}
26
27          func NewEmpresaService(db *gorm.DB) *EmpresaService {
28              return &EmpresaService{DB: db}
29              }
30
31          func (e EmpresaService) CreateEmpresa(input *model.
                EmpresaCreateInput, tx *gorm.DB) (*models.Cpi_empresas, error
                ) {
32              empresa := &models.Cpi_empresas{
33              Razon_social:input.RazonSocial,
34              Rfc_empresa: input.RfcEmpresa,
35              Representante_gerente_contacto: null.StringFromPtr(input.
                    RepresentanteGerenteContacto),
36              Horario_atencion:            null.StringFromPtr(input.
                    HorarioAtencion),
37              Notas_descripcion:           null.StringFromPtr(input.
                    NotasDescripcion),
38              Numreg_lau_cdmx:             null.StringFromPtr(input.
                    NumregLauCdmx),
39              Numreg_ramir:                null.StringFromPtr(input.
                    NumregRamir),
40              Numreg_plan_manejo:          null.StringFromPtr(input.
                    NumregPlanManejo),
41              Numreg_ambiental:            null.StringFromPtr(input.
                    NumregAmbiental),
42          }
43
44          if input.NumregAmbiental != nil { // Registro ambiental
45          fecha, _ := time.Parse("2006-01-02", null.StringFromPtr(input.
```

```
            NumregAmbientalVigencia).String)
46          empresa.Numreg_ambiental_vigencia = null.TimeFrom(fecha)
47          }
48
49          if input.EsMatrizOSucursal == 2 { // sucursal
50          empresaGlobal := models.Cpi_empresas{}
51          if err := tx.First(&empresaGlobal, "id_empresa = ?", input.
                RazonSocial).Error; err != nil {
52              return nil, errors.New("No existe la sucursal global. " +
                     err.Error())
53          }
54          empresa.Id_empresa_global = null.IntFrom(int64(empresaGlobal.
                Id_empresa))
55          empresa.Es_matriz_o_sucursal = 2
56              } else { // matriz
57          empresa.Es_matriz_o_sucursal = 1
58              }
59
60          if input.EsFisicaOMoral == 0 { // Fisica
61          empresa.Tipoempresa_es_personafisica = true
62          empresa.Tipoempresa_es_personamoral = false
63              } else if input.EsFisicaOMoral == 1 { // Moral
64          empresa.Tipoempresa_es_personafisica = false
65          empresa.Tipoempresa_es_personamoral = true
66              } else {
67              return empresa, errors.New("Valor no valido para
                    es_fisico_o_moral")
68              }
69
70          if input.RolEmpresa == 1 { // Generador
71              empresa.Rolempresa_es_solicitante = false
72              empresa.Rolempresa_es_generador = true
73              empresa.Rolempresa_es_prestadorservicios = false
74
75              if input.NumregLauCdmx != nil { // LAU
76              fecha, _ := time.Parse("2006-01-02", null.StringFromPtr(
                    input.NumregLauCdmxVigencia).String)
77              empresa.Numreg_lau_cdmx_vigencia = null.TimeFrom(fecha)
78                  }
79
80              //if input.NumregPlanManejoBienes != nil { // Manejo de
                    bienes
81              // fecha, _ := time.Parse("2006-01-02", null.
                    StringFromPtr(input.NumregPlanManejoBienesVigencia).
                    String)
```

```
82              // empresa.Numreg_plan_manejo_bienes_vigencia = null.
                    TimeFrom(fecha)
83              //}
84
85          } else if input.RolEmpresa == 2 { // Prestador de servicios
86              empresa.Rolempresa_es_solicitante = false
87              empresa.Rolempresa_es_generador = false
88              empresa.Rolempresa_es_prestadorservicios = true
89              } else {
90              return empresa, errors.New("Valor no valido para
                    rol_empresa")
91              }
92
93          if input.NumregRamir != nil { // RAMIR
94              fecha, _ := time.Parse("2006-01-02", null.StringFromPtr(
                    input.NumregRamirVigencia).String)
95              empresa.Numreg_ramir_vigencia = null.TimeFrom(fecha)
96              }
97
98              if input.NumregPlanManejo != nil {
99              fecha, _ := time.Parse("2006-01-02", null.StringFromPtr(
                    input.NumregPlanManejoVigencia).String)
100             empresa.Numreg_plan_manejo_vigencia = null.TimeFrom(fecha
                    )
101             }
102
103             err := tx.Create(&empresa).Error
104
105             if err != nil {
106             return nil, errors.New("Error al crear la empresa. " +
                    err.Error())
107             }
108
109             return empresa, err
110         }
111
112     func (e EmpresaService) GetEmpresa(id_empresa int, rfc_empresa
            string) (*model.CpiEmpresas, error) {
113         empresaBusqueda := &model.CpiEmpresas{}
114         //err := e.DB.Where("id_empresa = ?", id_empresa).Or(
115         //  "RFC_empresa = ?", rfc_empresa).First(&empresaBusqueda).
                Error
116         err := e.DB.Where("id_empresa = ?", id_empresa).Or("
                rfc_empresa = ?", rfc_empresa).First(&empresaBusqueda).
                Error
```

```
117             return empresaBusqueda, err
118         }
119
120         func (e EmpresaService) GetAllEmpresas() ([]*model.CpiEmpresas,
                error) {
121             empresas := []*model.CpiEmpresas{}
122             err := e.DB.Find(&empresas).Error
123
124             return empresas, err
125         }
126
127         func (e EmpresaService) UpdateEmpresa(empresaInput *model.
                EmpresaUpdateInput, id_empresa int) error {
128             empresa := &models.Cpi_empresas{}
129
130             empresa := &models.Cpi_empresas{}
131
132             err := e.DB.First(&empresa, id_empresa).Error
133             //Variables de tipo time.Time{} desde cadena de texto desde
                    empresaInput
134             timeParseLauVigencia, _ := time.Parse("2006-01-02", *
                    empresaInput.NumregLauCdmxVigencia)
135             timeParseRamirVigencia, _ := time.Parse("2006-01-02", *
                    empresaInput.NumregRamirVigencia)
136             timeParsePlanManejoVigencia, _ := time.Parse("2006-01-02", *
                    empresaInput.NumregPlanManejoVigencia)
137             timeParseAmbientalVigencia, _ := time.Parse("2006-01-02", *
                    empresaInput.NumregAmbientalVigencia)
138             empresa.Razon_social = app.ChangeString(empresaInput.
                    RazonSocial, empresa.Razon_social)
139             empresa.Rfc_empresa = app.ChangeString(empresaInput.
                    RfcEmpresa, empresa.Rfc_empresa)
140             empresa.Es_matriz_o_sucursal = app.ChangeInt(empresaInput.
                    EsMatrizOSucursal, empresa.Es_matriz_o_sucursal)
141             empresa.Id_empresa_global = null.IntFrom(int64(app.ChangeInt(
                    empresaInput.IDEmpresaGlobal, int(empresa.
                    Id_empresa_global.Int64))))
142             empresa.Nombre_empresa_sucursal = null.StringFrom(app.
                    ChangeString(empresaInput.NombreEmpresaSucursal, empresa.
                    Nombre_empresa_sucursal.String))
143             empresa.Horario_atencion = null.StringFrom(app.ChangeString(
                    empresaInput.HorarioAtencion, empresa.Horario_atencion.
                    String))
144             empresa.Notas_descripcion = null.StringFrom(app.ChangeString(
                    empresaInput.NotasDescripcion, empresa.Notas_descripcion.
```

```
                String))
145             empresa.Numreg_lau_cdmx = null.StringFrom(app.ChangeString(
                    empresaInput.NumregLauCdmx, empresa.Numreg_lau_cdmx.
                    String))
146             empresa.Numreg_ramir = null.StringFrom(app.ChangeString(
                    empresaInput.NumregRamir, empresa.Numreg_ramir.String))
147             empresa.Numreg_plan_manejo = null.StringFrom(app.ChangeString
                    (empresaInput.NumregPlanManejo, empresa.
                    Numreg_plan_manejo.String))
148             empresa.Numreg_ambiental = null.StringFrom(app.ChangeString(
                    empresaInput.NumregAmbiental, empresa.Numreg_ambiental.
                    String))
149             empresa.Numreg_lau_cdmx_vigencia = null.TimeFrom(app.
                    ChangeTime(&timeParseLauVigencia, empresa.
                    Numreg_lau_cdmx_vigencia.Time))
150             empresa.Numreg_ramir_vigencia = null.TimeFrom(app.ChangeTime
                    (&timeParseRamirVigencia, empresa.Numreg_ramir_vigencia.
                    Time))
151             empresa.Numreg_plan_manejo_vigencia = null.TimeFrom(app.
                    ChangeTime(&timeParsePlanManejoVigencia, empresa.
                    Numreg_plan_manejo_vigencia.Time))
152             empresa.Numreg_ambiental_vigencia = null.TimeFrom(app.
                    ChangeTime(&timeParseAmbientalVigencia, empresa.
                    Numreg_ambiental_vigencia.Time))
153
154             err = e.DB.Model(&empresa).Where("id_empresa = ?", id_empresa
                    ).Updates(empresa).Error
155
156                 return err
157         }
158
159         func (e EmpresaService) DeleteEmpresa(id_empresa int) error {
160             empresa := &models.Cpi_empresas{}
161             err := e.DB.Delete(empresa, id_empresa).Error
162                 return err
163         }
164
165         func esVacioCadena(parametro string) string {
166             if &parametro == nil {
167                 parametro = ""
168                 }
169                 return parametro
170         }
171
172         func esVacioEntero(parametro int) int {
```

```
173             if &parametro == nil {
174                 parametro = 0
175                 }
176                 return parametro
177         }
178
179         func esVacioBool(parametro bool) bool {
180             if &parametro == nil {
181                 parametro = false
182                 }
183                 return parametro
184         }
```

Código 4.9: Cuerpo de las consultas del catálogo de procesos de empresas.

```
 1      func (r *queryResolver) GetEmpresa(ctx context.Context, idEmpresa *
            int, rfcEmpresa *string) (*model.CpiEmpresas, error) {
 2          idEmpresa = FillZero(idEmpresa)
 3          rfcEmpresa = FillNA(rfcEmpresa)
 4          fmt.Printf("Gettin incoming ID %x\n", idEmpresa)
 5          empresa, err := r.Empresa.GetEmpresa(*idEmpresa, *rfcEmpresa)
 6
 7          seleccionEmpresa := &model.CpiEmpresas{
 8          IDEmpresa:                  empresa.IDEmpresa,
 9          RazonSocial:                empresa.RazonSocial,
10          RfcEmpresa:                 empresa.RfcEmpresa,
11          EsMatrizOSucursal:          empresa.EsMatrizOSucursal,
12          IDEmpresaGlobal:            empresa.IDEmpresaGlobal,
13          NombreEmpresaSucursal:      empresa.NombreEmpresaSucursal,
14          RepresentanteGerenteContacto: empresa.
               RepresentanteGerenteContacto,
15          HorarioAtencion:            empresa.HorarioAtencion,
16          NotasDescripcion:           empresa.NotasDescripcion,
17          NumregLauCdmx:              empresa.NumregLauCdmx,
18          NumregRamir:                empresa.NumregRamir,
19          NumregPlanManejo:           empresa.NumregPlanManejo,
20          NumregAmbiental:            empresa.NumregAmbiental,
21          NumregLauCdmxVigencia:      empresa.NumregLauCdmxVigencia,
22          NumregRamirVigencia:        empresa.NumregRamirVigencia,
23          NumregPlanManejoVigencia:   empresa.NumregPlanManejoVigencia,
24          NumregAmbientalVigencia:    empresa.NumregAmbientalVigencia,
25          CtlEsRegistroActivo:        empresa.CtlEsRegistroActivo,
26          CtlFechaRegistro:           empresa.CtlFechaRegistro,
27          CtlUsuarioRegistro:         empresa.CtlUsuarioRegistro,
28          CtlFechaModificacion:       empresa.CtlFechaModificacion,
```

```
29          CtlUsuarioModificacion:      empresa.CtlUsuarioModificacion,
30          RolempresaEsSolicitante:     empresa.RolempresaEsSolicitante,
31          RolempresaEsGenerador:       empresa.RolempresaEsGenerador,
32          RolempresaEsPrestadorservicios: empresa.
                RolempresaEsPrestadorservicios,
33          TipoempresaEsPersonafisica:  empresa.TipoempresaEsPersonafisica,
34          TipoempresaEsPersonamoral:   empresa.TipoempresaEsPersonamoral,
35              }
36              if err != nil {
37                  return nil, err
38              }
39              return seleccionEmpresa, nil
40          }
41
42      func (r *queryResolver) GetAllEmpresas(ctx context.Context) ([]*
            model.CpiEmpresas, error) {
43          empresas, err := r.Empresa.GetAllEmpresas()
44              if err != nil {
45                  return nil, err
46              }
47
48          return empresas, nil
49      }
50      func (r *mutationResolver) CreateEmpresa(ctx context.Context, input
            model.EmpresaCreateInput) (int, error) {
51          var idEmpresa int
52          err := r.DB.Transaction(func(tx *gorm.DB) error {
53          empresa, err := r.Empresa.CreateEmpresa(&input, tx)
54              if err != nil {
55              return err
56                  }
57              idEmpresa = empresa.Id_empresa
58                  return nil
59              })
60              if err != nil {
61                  return 0, err
62              }
63              return idEmpresa, nil
64          }
65
66      func (r *mutationResolver) DeleteEmpresa(ctx context.Context,
            idEmpresa int) (string, error) {
67          err := r.Empresa.DeleteEmpresa(idEmpresa)
68              if err != nil {
69                  return "", err
```

```
70          }
71          successMessage := "Borrado exitoso"
72              return successMessage, nil
73      }
74
75      func (r *mutationResolver) UpdateEmpresa(ctx context.Context,
            idEmpresa int, input model.EmpresaUpdateInput) (string, error) {
76          err := r.Empresa.UpdateEmpresa(&input, idEmpresa)
77              if err != nil {
78                  return "nil", err
79          }
80          successMessage := "Actualizacion exitosa"
81
82          return successMessage, nil
83      }
```

Código 4.10: Resolutores para los operadores del catálogo de procesos de empresas.

4.1.2. Microservicio

Como se mencionó en el capítulo anterior, se desarrolló también un microservicio para el registro del domicilio de una empresa. Para lo cual se utilizó la API de geolocalización de COPOMEX. Este microservicio fue integrado en la plataforma SEDEMA (Código 4.11). Además, como resultado se muestra el Código 4.12 de la función 'main' en el que se invoca el despliegue del microservicio. Se presenta también el Código 4.13 del servidor local en Go, que escucha las peticiones del microservicio, utilizando la API de la COPOMEX.

```
1       import 'package:autofill_comp/utils/responsive.dart';
2       import 'package:autofill_comp/models/address_model.dart';
3       import 'package:flutter/material.dart';
4       import 'package:flutter_form_builder/flutter_form_builder.dart';
5       import 'package:http/http.dart' as http;
6       import 'dart:convert' as convert;
7
8       Color primaryColor = const Color(0xff1E5C4F); //Color verde
9       Color secondaryColor = const Color(0xffBC955B);
10      class AddressForm extends StatefulWidget {
11      const AddressForm({Key? key}) : super(key: key);
12      @override
13      _AddressFormState createState() => _AddressFormState();}
14
15      // Front-end del formulario de direccion
```

```
16      class _AddressFormState extends State<AddressForm> {
17      final _formKey = GlobalKey<FormBuilderState>(); // Formulario
18
19      String? estado;
20          String? alcaldia;
21          List<String> colonias = [''];
22
23          @override
24          Widget build(BuildContext context) {
25          final Responsive responsive = Responsive.of(context);
26
27          return FormBuilder( key: _formKey,
28          child: Column(
29          children: [
30          // Buscador de direccion por codigo postal
31          TextField(
32              decoration: const InputDecoration(
33              hintText: 'Codigo Postal',),
34              onSubmitted: (value) => fetchFromServer(value),),
35              FormBuilderTextField(
36              name: 'calle',
37              decoration: InputDecoration(
38              labelText: 'Calle',
39              labelStyle: TextStyle(
40              color: primaryColor,
41              fontWeight: FontWeight.w500,),
42              border: const UnderlineInputBorder(),
43              focusedBorder: UnderlineInputBorder(
44              borderSide: BorderSide(color: primaryColor),),
45              filled: true,
46              fillColor: secondaryColor,), ),
47              FormBuilderTextField(
48              name: 'numero',
49              decoration: InputDecoration(
50              labelText: 'Numero'),),
51              FormBuilderDropdown(
52              name: 'colonia',
53              decoration: InputDecoration(
54              labelText: 'Colonia',),
55              items: colonias.map((colonia) => DropdownMenuItem(
56              value: colonia,
57              child: Text('$colonia'),
58              )).toList(),),
59              FormBuilderTextField(
60              name: 'alcaldia',
```

```
61              enabled: false,
62              decoration: InputDecoration(
63              labelText: 'Alcaldia',),),
64              FormBuilderTextField(
65              name: 'codigo_postal',
66              enabled: false,
67              decoration: InputDecoration(
68              labelText: 'Codigo Postal',),),
69              FormBuilderTextField(
70              name: 'estado',
71              enabled: false,
72              decoration: InputDecoration(
73              labelText: 'Estado'),),
74              ElevatedButton(
75              onPressed: (){
76              //Salvas el estado del formulario_formKey.currentState?.save
                    ();
77              print(_formKey.currentState?.value);
78              },
79              child: Text('Submit')),],),);
80          }
81
82          Future<void> fetchFromServer(String cp) async {
83          //Request al server
84          Uri url = Uri.parse("http://127.0.0.1:5500/cp$cp");
85          //Respuesta del server
86          var serverResponse = await http.get(url);
87          // Si Status code != 200 hubo un error en la llamada al servidor
88          if (serverResponse.statusCode == 200) { print(serverResponse.body
                );
89          // Guardamos en una varible el json
90          Response response = Response.fromJson(convert.jsonDecode(
                serverResponse.body));
91          // Si error es verdadero hay un error para encontrar el codigo
                postal
92          if (response.error == false && response.direccion != null) {
93          // Recuperacion del estado, municipio y colonias
94          _formKey.currentState!.fields['codigo_postal']?.didChange(
                response.direccion?.cp);
95          _formKey.currentState!.fields['estado']?.didChange(response.
                direccion?.estado);
96          _formKey.currentState!.fields['alcaldia']?.didChange(response.
                direccion?.municipio);
97
98          // CAmbio dinamico de dropdown de colonias
```

```
99          setState(() {
100         _formKey.currentState!.fields['colonia']?.didChange('');
101         colonias = [''];
102         List coloniasTemp = response.direccion!.asentamiento;
103         for (var colonia in coloniasTemp) {
104         colonias.add(colonia);}});
105         }}}
106     }
```

Código 4.11: Construcción del microservico para el registro del domicilio de una empresa.

```
1     import 'package:autofill_comp/pages/home_page.dart';
2     import 'package:flutter/material.dart';
3
4     void main(List<String> args) {
5     runApp(MyApp());}
6
7     class MyApp extends StatelessWidget {
8         @override
9         Widget build(BuildContext context) {
10        return MaterialApp(
11            title: 'API Geolocalizacion',
12            theme: ThemeData(
13            primarySwatch: Colors.blue,
14            visualDensity: VisualDensity.adaptivePlatformDensity,),
15        home: HomePage(),
16        );}
17    }
```

Código 4.12: Función main para la ejeución del microservicio.

```
1     import (
2         "fmt"
3         "github.com/gorilla/handlers"
4         "github.com/gorilla/mux"
5         "io/ioutil"
6         "log"
7         "net/http"
8             )
9         const port = ":5500"
10
11        func main() {
12        router := mux.NewRouter()
13        // Manejadores del CORS policy
14        headers := handlers.AllowedHeaders([]string{"X-Request-With", "
              Content-Type", "Authorization"})
```

```
15        methods := handlers.AllowedMethods([]string{"GET", "POST", "PUT",
              "DELETE"})
16        origins := handlers.AllowedOrigins([]string{"*"})
17        //Handlers root
18        router.HandleFunc("/", rootPage)
19        //Handler obtener direccion por codigo postal
20        router.HandleFunc("/cp{cp}", getDireccion)
21        // Mensaje de inicializacion de servidor
22        fmt.Println("Serving @ http://127.0.0.1" + port)
23        log.Fatal(http.ListenAndServe(port, handlers.CORS(headers,
              methods, origins)(router)))
24        }
25
26        func rootPage(w http.ResponseWriter, r *http.Request) {
27        w.Write([]byte("This is root page"))}
28
29        func getDireccion(w http.ResponseWriter, r *http.Request) {
30        // Guardamos la variable que contiene el codigo postal
31        cp := mux.Vars(r)["cp"]
32        // Respuesta del servidor
33        //resp, err := http.Get("https://api.copomex.com/query/info_cp/"
              + cp + "?type=simplified&token=pruebas") // Token de pruebas
34        resp, err := http.Get("https://api.copomex.com/query/info_cp/" +
              cp + "?type=simplified&token=c5b05a80-9d5a-4752-92c8-
              ee85ba8f83be") // Token real limitado a 50 pruebas
35        if err != nil {
36        log.Fatalln(err)
37        } else {
38            body, err := ioutil.ReadAll(resp.Body)
39            if err != nil {
40            log.Fatalln(err)
41            } else {
42                //We Read the response body on the line below.
43                w.Header().Set("Content-Type", "application/json; charset
                      =utf-8")
44                w.Write(body)
45            }}
46        }
```

Código 4.13: Servidor local que escucha las peticiones del microservicio.

4.1.3. Resultados de las pruebas de funcionalidad

Como se mencionó en el capítulo anterior, para probar el funcionamiento de los operadores desarrollados y posteriormente sean utilizados en forma de APIs en la plataforma SEDEMA, se utilizó como herramienta de pruebas Postman, la cual permitió probar los diferentes operadores en forma de APIs. En este sentido, se probaron todos los operadores desarrollados, con base en los catálogos seleccionados: a) generales (tipo de usuario, estados, calles y códigos postales); y b) específicos de procesos (tipo de empresas, roles, identificadores, estatus de procesos, ofertas y ofertas solicitantes). Por lo tanto, a manera de ejemplo, en este apartado se muestra los resultados de la funcionalidad del catálogo general `cg_usuarios_tipo` y del catálogo específico para procesos `cpi_empresas_tipo`. En el Anexo B se muestran las pruebas de funcionalidad realizadas para los demás operadores desarrollados.

Así, para el caso del catálogo `cg_usuarios_tipo`, se probaron en forma de APIs: GetUsuarioTipo y GetAllUsuariosTipo. La primera (GetUsuarioTipo) es utilizada para la consulta de los tipos de usuario (ver Figura 4.1), donde con base en su ejecución devuelve el registro del catálogo, filtrado por el campo id_tipo_usuario, el cual corresponde al administrador de la plataforma SEDEMA. Mientras que con la segunda API (GetAllUsuariosTipo) se obtiene todos los registros existentes en dicho catálogo (tal como se muestra en la Figura 4.2).

Figura 4.1: API para la consulta de un tipo específico de usuario.

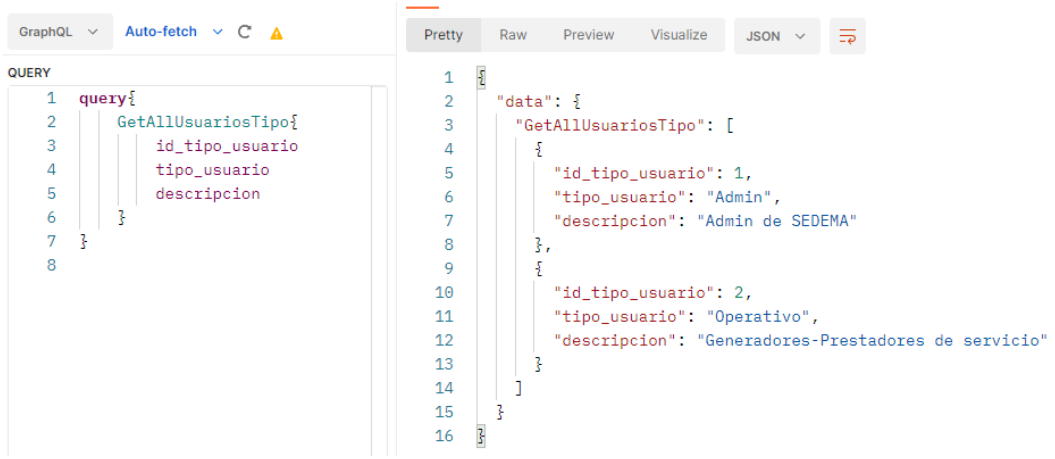

Figura 4.2: API para la consulta de los tipos de usuario existentes en el catálogo.

Para el caso del catálogo `cpi_empresas`, se probaron en forma de APIs: GetAllEmpresas, GetEmpresa, CreateEmpresa, UpdateEmpresa y DeleteEmpresa. Mediante GetAllEmpresas se obtienen todas las empresas registradas en dicho catálogo (Figura 4.3); a través de GetEmpresa se obtiene la información de una empresa en particular, ya sea mediante el identificador o rfc, como se muestra en la Figura 4.4, para el caso del identificador igual a '2'.

Figura 4.3: API para la consulta de todas las empresas registradas.

```
query(
    $id_empresa: ID
    $rfc_empresa: String
){
    GetEmpresa(
        id_empresa: $id_empresa
        rfc_empresa: $rfc_empresa
        ){
            id_empresa
            razon_social
            rfc_empresa
            es_matriz_o_sucursal
            id_empresa_global
            nombre_empresa_sucursal
            representante_gerente_contacto
            horario_atencion
            notas_descripcion
            tipo_empresa_fiscal
            numreg_lau_cdmx
            numreg_ramir
            numreg_plan_manejo
            numreg_ambiental
            numreg_lau_cdmx_vigencia
            numreg_ramir_vigencia
            numreg_plan_manejo_vigencia
            numreg_ambiental_vigencia
            ctl_es_registro_activo
            ctl_fecha_modificacion
            ctl_fecha_registro
            ctl_usuario_modificacion
            ctl_usuario_registro
        }
}
```


```
{
    "data": {
        "GetEmpresa": {
            "id_empresa": 2,
            "razon_social": "Cartones S.A.",
            "rfc_empresa": "RFC1234",
            "es_matriz_o_sucursal": 1,
            "id_empresa_global": null,
            "nombre_empresa_sucursal": null,
            "representante_gerente_contacto": null,
            "horario_atencion": null,
            "notas_descripcion": null,
            "tipo_empresa_fiscal": null,
            "numreg_lau_cdmx": null,
            "numreg_ramir": "RAMIR1234",
            "numreg_plan_manejo": null,
            "numreg_ambiental": null,
            "numreg_lau_cdmx_vigencia": null,
            "numreg_ramir_vigencia": "0001-01-01T00:00:00Z",
            "numreg_plan_manejo_vigencia": null,
            "numreg_ambiental_vigencia": null,
            "ctl_es_registro_activo": false,
            "ctl_fecha_modificacion": "0001-01-01T00:00:00Z",
            "ctl_fecha_registro": "0001-01-01T00:00:00Z",
            "ctl_usuario_modificacion": 0,
            "ctl_usuario_registro": ""
        }
    }
}
```


Figura 4.4: API para la consulta de una determinada empresa registrada.

Se probaron también UpdateEmpresa para actualizar la información de los campos de una determinada empresa (ver Figura 4.5), y DeleteEmpresa para eliminar un registro específico de una empresa mediante su identificador (ver Figura 4.6).

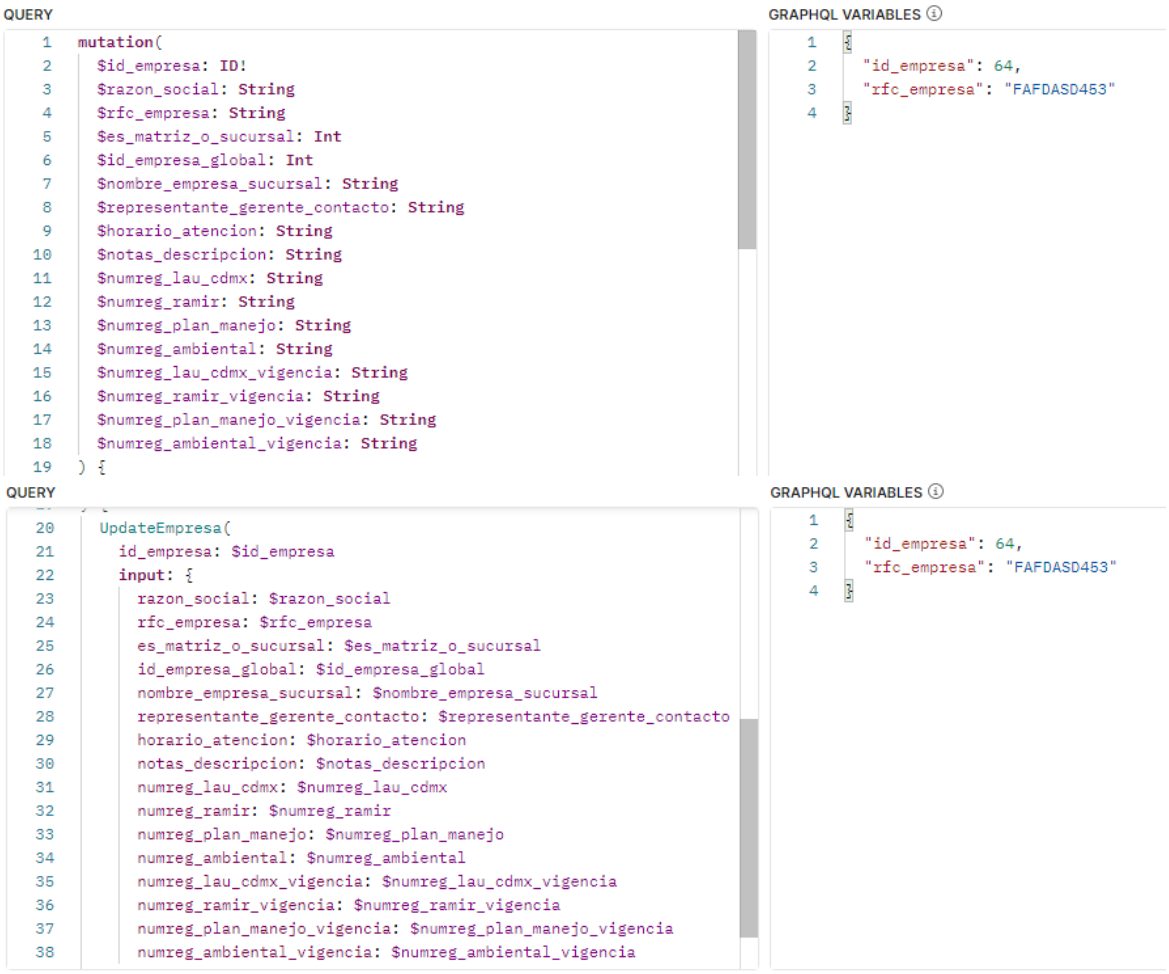

Figura 4.5: API utilizada para la actualización del registro de una determinada empresa.

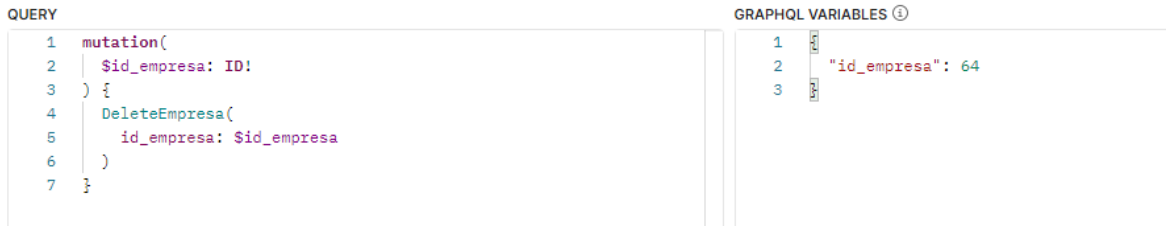

Figura 4.6: API utilizada para eliminar el registro de una determinada empresa.

Como se mencionó, en el Anexo B se muestran las pruebas realizadas a los operadores de los demás catálogos. Estas pruebas de funcionalidad, para cada catálogo, forman parte de las pruebas unitarias de los catálogos codificados, donde se devuelven estructuras de datos en lugar de campos a través del uso de operadores anidados.

Por otra parte, se probó también la funcionalidad del microservicio para el registro del domicilio de una empresa. La Figura 4.7 muestra la ejecución de este microservicio, donde se pide el "Código Postal" para registrar el domicilio y de manera interna se envía la petición para el autocompletado de la alcaldía, el código postal y el estado. A modo de prueba, a través de la Figura 4.8 se muestra la ejecución con el código postal 14100, el cual se encuentra en la alcaldía Tlalpan, de la Ciudad de México; siendo estos datos autocompletados por el microservicio, para así facilitar el registro.

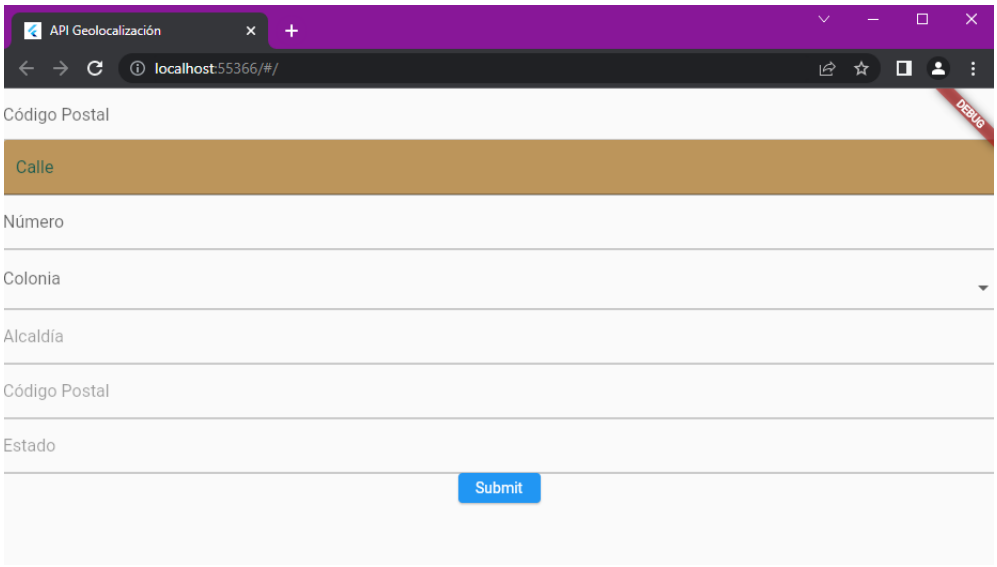

Figura 4.7: Estado inicial del microservicio para el autocompletado del formulario.

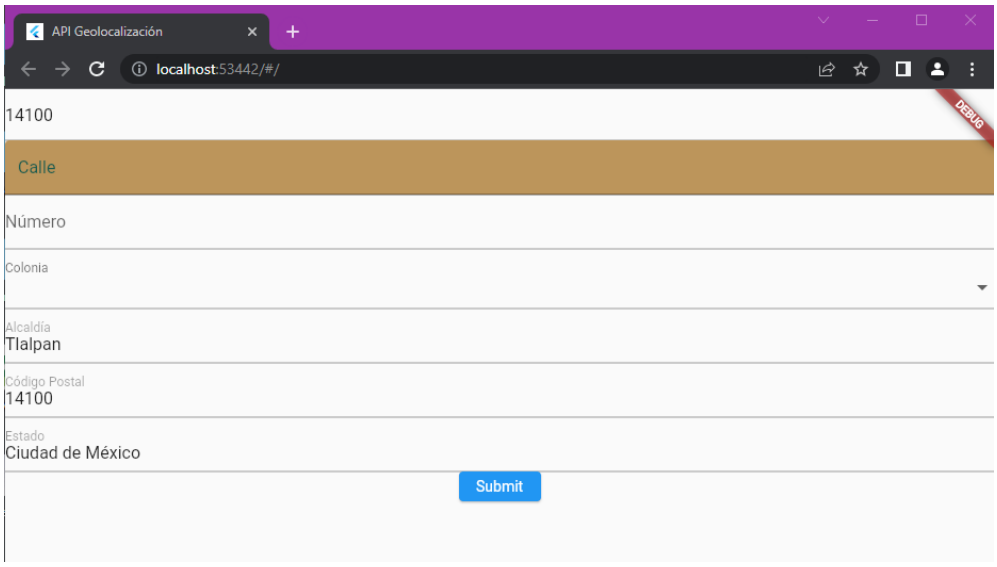

Figura 4.8: Respuesta del microservicio para el autocompletado del formulario.

4.2. Síntesis

Como se observó en las pruebas de los diferentes operadores, cada catálogo posee los operadores definidos en la sección 3.4 de manera funcional, permitiendo una comunicación entre el Back-End y el Front-End para el manejo adecuado de la plataforma SEDEMA. Además, las pruebas realizadas fueron de manera exitosa, verificándose así el comportamiento de cada tipo de operador CRUD. Esto garantiza que la funcionalidad de la aplicación.

Por otro lado, el haber elegido GraphQL para el Back-End permite escalar, mediante parámetros de ancho de banda y número de solicitudes, los cambios requeridos por los usuarios, tomando como base el acceso simultáneo al servidor a través de las diferentes peticiones de los operadores que se desarrollaron.

5
Conclusiones y trabajo futuro

Índice

En este capítulo se presentan las conclusiones del trabajo realizado y se establecen las futuras líneas de investigación como trabajo posterior, cuyas bases se sustentan de acuerdo a los resultados obtenidos.

5.1. Conclusiones generales

La conectividad que genera Internet presenta una nueva manera de comunicación para compartir la información, donde cada vez hay mayor demanda en la industria del desarrollo de software para cubrir diferentes necesidades, como es el caso de las aplicaciones Web, las cuales presentan una serie de ventajas para gestionar la información, con el propósito de ser utilizadas en diversos dispositivos con una alta calidad gráfica, personalización, adaptación dinámica del contenido y estilos de presentación.

La plataforma SEDEMA, en la que se participó en su desarrollo del lado del Back-End, busca servir como herramienta de gestión de los residuos sólidos urbanos de la

Ciudad de México a través de diferentes dispositivos, como navegadores Web en una computadora y celulares. El Back-End emplea una arquitectura de servidor multicapa, donde se implementó GraphQL como lenguaje de consulta y manipulación de datos para las APIs codificadas en forma de operadores CRUD.

Una comunicación eficiente y asertiva en la resolución de problemas que surgen en el desarrollo de software de gran magnitud es esencial. Esta comunicación asertiva entre los miembros del equipo de desarrollo y los usuarios permitieron un desarrollo ágil de la plataforma SEDEMA. Esto se vio reflejado en los diferentes cambios que se realizaron debido a las iteraciones en el desarrollo, y en la toma de decisiones entre los miembros del equipo de desarrollo del Back-End, asegurando así la calidad de operación de las diferentes funcionalidades codificadas.

Por otro lado, a través de la plataforma SEDEMA se busca encadenar una economía circular, la cual basa su comportamiento en la producción de bienes materiales en un mercado donde la oferta y demanda toma como referencia la sustentabilidad y responsabilidad con el ambiente. Esta responsabilidad tiene como objetivo gestionar el manejo de los RSU, para así reciclar, de manera eficiente y rápida, estos residuos en el ciclo productivo y de consumo.

Los diferentes RSU que se concentran en los catálogos de procesos de la plataforma web, son un punto de referencia para el tratamiento y control de la mayor cantidad de RSU que son generados en la Ciudad de México. Los catálogos anteriores están sujetos a depuración y crecimiento programado, tanto como sea requerido por los usuarios o administradores. Esto permite tener un mayor alcance en los diferentes RSU que son tratados.

5.2. Conclusiones particulares

La contribución en el desarrollo del Back-End de la plataforma SEDEMA generó diferentes retos. Uno de estos retos fue el diseño de la base de datos, esto es, se solicitó que diferentes operadores de los catálogos de procesos devolvieran estructuras de datos, en lugar de campos correspondientes a cada una de las llaves foráneas. Otro de los retos fueron los cambios solicitados para que la respuesta del servidor mantuviera la rapidez en la ejecución de cada uno de los operadores CRUD. Esto representó uno de los principios de las aplicaciones Web, la velocidad en el tiempo de respuesta, preservando la inmediatez en el acceso a la información. Por lo que, como conclusiones particulares, destacan:

- En la validación de los requerimientos funcionales se logró identificar la operación de la plataforma SEDEMA. Iniciando con el registro de una empresa mediante alguno de los roles: Generador o Prestador de Servicios, donde en ciertos casos de uso, una empresa puede estar registrada en ambos roles e interactuar de manera simultánea.

- Se diseñó la base de datos con el fin de reducir la pérdida de información en el flujo de datos de la plataforma. Se propusieron cambios en algunos catálogos para reducir la redundancia de los datos y así mitigar el riesgo de overloading en el tiempo de ejecución de los operadores CRUD.

- Se logró implementar diferentes operadores CRUD para los diferentes catálogos seleccionados, manteniendo la integridad de los datos de origen, contemplando los diferentes escenarios donde los datos de procesos son nulos, y asegurando el tipado de los datos utilizados en el Front-End.

- Se preservó la naturalidad del lenguaje español mediante la sintaxis del framework gqlgen, garantizando la permanencia de los acentos de los diferentes datos de origen o generados en el flujo de datos en la aplicación Web. Esto debido a que la sintaxis inicial en el Back-End con graphql-go carecía de la capacidad de parsear los acentos en la operación de los diferentes catálogos al ser ejecutados para alguna operación CRUD específica.

- Como parte de las mejoras, se realizó un refactoring al Back-End para eliminar el catálogo de `cpr_roles`, donde esta eliminación generó una dependencia en cascada con los catálogos de procesos involucrados. Se eliminó todas las dependencias por campos, llaves foráneas y de los servicios de escucha del servidor para los operadores codificados.

- El impacto de renombrar un catálogo es una tarea que implica un refactoring complejo, ya que al renombrar un catálogo, la dependencia que tienen otros catálogos con este, genera un error en cascada, que permite observar los detalles que se omitieron en el momento de realizar el refactoring. El catálogo que se renombró fue `cpr_roles_empresas` por `cpr_servicios_empresas`.

5.3. Trabajo futuro

Si bien los resultados obtenidos fueron favorables, el interés de extender más funcionalidades en la plataforma SEDEMA, deja abierta la posibilidad de añadir más microservicios que se puedan utilizar en el Front-End, los cuales permitirán la escalabilidad de la aplicación en función de las funcionalidades que sean requeridas por los usuarios y administradores. Por lo que, entre los trabajos futuros destacan:

- Implementación de un microservicio para generar reportes de las actividades de un usuario en la plataforma. Este reporte podrá satisfacer el requerimiento funcional del control y administración de la plataforma.
- Implementación de microservicios para los diferentes tipos de usuarios en la plataforma y validar las operaciones definidas para cada uno de estos usuarios.
- Implementación de un microservicio de respuesta para el envío de correos electrónicos al registrarse en la plataforma.

Anexos

A

Código de los operadores CRUD

Índice

A.1. Códigos para los catálogos generales

```
1  package models
2
3  type Cg_Estados struct {
4      Id_Estado    int    `json:"id_estado" gorm:"primary_key"`
5      Nombre_Estado string `json:"nombre_estado"`
6      Clave_Estado string `json:"clave_estado"`
7  }
```

Código A.1: Modelo para el catálogo general de estados.

```
1  type Cg_estados {
2      id_estado: ID!
3      nombre_estado: String!
4      clave_estado: String!
5  }
```

Código A.2: Prototipo del schema.graphqls del catálogo general estados.

```
1  GetEstado(id_estado: ID, nombre_estado: String, clave_estado:
       String): [Cg_estados!]!
2  GetAllEstados: [Cg_estados!]!
```

Código A.3: Firmas de tipo query del query.graphqls del catálogo general estados.

```
1  package repository
2
3  import (
4      "apiCatalogosGenerales/graph/model"
5      "gorm.io/gorm"
6  )
7
8  type EstadosRepository interface {
9      GetEstado(id int, nombre_estado string, clave_estado
           string) ([]*model.CgEstados, error)
10     GetEstadoById(id int) (model.CgEstados, error)
11     GetAllEstados() ([]*model.CgEstados, error)
12 }
13
14 type EstadoService struct {
15     DB *gorm.DB
16 }
17
18 var _ EstadosRepository = &EstadoService{}
19
20 func NewEstadoService(db *gorm.DB) *EstadoService {
21     return &EstadoService{
22         DB: db,
23     }
24 }
25
26 func (e *EstadoService) GetEstado(id int, nombre_estado
       string, clave_estado string) ([]*model.CgEstados, error) {

27     estado := []*model.CgEstados{}
28     err := e.DB.Table("cg_estados").Where(
29         "id_estado = ?", id).Or(
30         "nombre_estado = ?", nombre_estado).Or(
31         "clave_estado = ?", clave_estado,
32     ).Find(&estado).Error
33
34     return estado, err
35 }
36
```

```
37          func (e *EstadoService) GetEstadoById(id int) (model.
                CgEstados, error) {
38              estado := model.CgEstados{}
39              err := e.DB.Table("cg_estados").Where(
40                  "id_estado = ?", id).First(&estado).Error
41
42              return estado, err
43          }
44
45          func (e *EstadoService) GetAllEstados() ([]*model.CgEstados,
                error) {
46              estados := []*model.CgEstados{}
47              err := e.DB.Table("cg_estados").Find(&estados).Error
48
49              return estados, err
50          }
```

Código A.4: Cuerpo de las consultas del repositorio del catálogo general estados.

```
 1          func (r *queryResolver) GetEstado(ctx context.Context,
                idEstado *int, nombreEstado *string, claveEstado *string)
                 ([]*model.CgEstados, error) {
 2              idEstado = FillZero(idEstado)
 3              nombreEstado = FillNA(nombreEstado)
 4              claveEstado = FillNA(claveEstado)
 5
 6              estado, err := r.Estado.GetEstado(*idEstado, *
                    nombreEstado, *claveEstado)
 7
 8              if err != nil {
 9                  return nil, err
10              }
11
12              return estado, nil
13          }
14
15          func (r *queryResolver) GetAllEstados(ctx context.Context)
                ([]*model.CgEstados, error) {
16              estados, err := r.Estado.GetAllEstados()
17              if err != nil {
18                  return nil, err
19              }
20              return estados, nil
21          }
```

Código A.5: Resolutores del query.resolvers.go para los operadores del catálogo general estados.

```
package models

import "github.com/guregu/null"

type Cg_Calles struct {
    Id_Calle       int        `json:"id_calle" gorm:"
       primary_key"`
    Nombre_Calle   string     `json:"nombre_calle"`
    Nombre_Corto   null.String `json:"nombre_corto"`
    Nombre_Coloquial null.String `json:"nombre_coloquial"`
    Nota_Ubicacion null.String `json:"nota_ubicacion"`
}
```

Código A.6: Modelo para el catálogo general de calles.

```
# cg_calles
type Cg_calles {
id_calle: ID!
nombre_calle: String!
nombre_corto: String
nombre_coloquial: String
nota_ubicacion: String
}
```

Código A.7: Prototipo del schema.graphqls del catálogo general calles

```
GetCalle(id_calle: ID, nombre_calle: String, nombre_corto:
   String, nombre_coloquial: String): [Cg_calles!]!
```

Código A.8: Firma de tipo query del query.graphqls del catálogo general calles.

```
package repository

import (
    "apiCatalogosGenerales/app/models"
    "apiCatalogosGenerales/graph/model"
    "errors"
    "github.com/guregu/null"
    "gorm.io/gorm"
)

type CalleRepository interface {
```

```
12              CreateCalle(input *model.CalleCreateInput, tx *gorm.DB)
                    (*models.Cg_Calles, error)
13              GetCalle(id_calle int, nombre_calle string, nombre_corto
                    string, nombre_coloquial string) ([]*model.CgCalles,
                    error)
14              UpdateCalle(input *model.CalleUpdateInput, id int) error
15              DeleteCalle(id int) error
16
17              //Crear(newCalle *model.CalleInput) (*models.Cg_Calles,
                    error)
18          }
19
20          type CalleService struct {
21              DB *gorm.DB
22          }
23
24          var _ CalleRepository = &CalleService{}
25
26          func NewCalleService(db *gorm.DB) *CalleService {
27              return &CalleService{DB: db}
28          }
29
30          func (c *CalleService) CreateCalle(input *model.
                CalleCreateInput, tx *gorm.DB) (*models.Cg_Calles, error)
                 {
31              calle := &models.Cg_Calles{
32                  Nombre_Calle: input.NombreCalle,
33                  Nota_Ubicacion: null.StringFromPtr(input.NotaUbicacion
                        ),
34              }
35
36              err := tx.Create(&calle).Error
37
38              if err != nil {
39                  return nil, errors.New("Error en la creacion de la
                        calle. " + err.Error())
40              }
41
42              return calle, err
43          }
44
45          func (c *CalleService) GetCalle(id_calle int, nombre_calle
                string, nombre_corto string, nombre_coloquial string) ([]*
                model.CgCalles, error) {
46              calle := []*model.CgCalles{}
```

```
47
48              err := c.DB.Where(
49                  "id_calle = ?", id_calle).Or(
50                  "nombre_calle = ?", nombre_calle).Or(
51                  "nombre_corto = ?", nombre_corto).Or(
52                  "nombre_coloquial = ?", nombre_coloquial,
53              ).Find(&calle).Error
54
55              return calle, err
56          }
57
58          func (c *CalleService) UpdateCalle(input *model.
                CalleUpdateInput, id int) error {
59              // Se crea una variable para guardar la informacion del
                    registro anterior
60              calle := models.Cg_Calles{}
61              err := c.DB.First(&calle, id).Error
62
63              if err != nil {
64                  return err
65              }
66
67              if input.NombreCalle != nil {
68                  calle.Nombre_Calle = *input.NombreCalle
69              }
70              if input.NombreCorto != nil {
71                  calle.Nombre_Corto = null.StringFromPtr(input.
                        NombreCorto)
72              }
73              if input.NombreColoquial != nil {
74                  calle.Nombre_Coloquial = null.StringFromPtr(input.
                        NombreColoquial)
75              }
76              if input.NotaUbicacion != nil {
77                  calle.Nota_Ubicacion = null.StringFromPtr(input.
                        NotaUbicacion)
78              }
79
80              err = c.DB.Model(&calle).Where("id_calle = ?", id).
                    Updates(calle).Error
81              return err
82          }
83
84          func (c *CalleService) DeleteCalle(id int) error {
85              calle := &models.Cg_Calles{}
```

```
86              err := c.DB.Delete(calle, id).Error
87              return err
88          }
```

Código A.9: Cuerpo de las consultas del repositorio del catálogo general calles.

```
1           func (r *queryResolver) GetCalle(ctx context.Context, idCalle
                *int, nombreCalle *string, nombreCorto *string,
               nombreColoquial *string) ([]*model.CgCalles, error) {
2               idCalle = FillZero(idCalle)
3               nombreCalle = FillNA(nombreCalle)
4               nombreCorto = FillNA(nombreCorto)
5               nombreColoquial = FillNA(nombreColoquial)
6
7               calle, err := r.Calle.GetCalle(*idCalle, *nombreCalle, *
                    nombreCorto, *nombreColoquial)
8               if err != nil {
9                   return nil, err
10              }
11              return calle, nil
12          }
13          func (r *mutationResolver) CreateCalle(ctx context.Context,
                input model.CalleCreateInput) (int, error) {
14              var idCalle int
15              err := r.DB.Transaction(func(tx *gorm.DB) error {
16                  calle, err := r.Calle.CreateCalle(&input, tx)
17                  if err != nil {
18                      return err
19                  }
20                  idCalle = calle.Id_Calle
21                  return nil
22              })
23              if err != nil {
24                  return 0, err
25              }
26              return idCalle, nil
27          }
28
29          func (r *mutationResolver) UpdateCalle(ctx context.Context,
                idCalle int, input model.CalleUpdateInput) (string, error)
                 {
30              err := r.Calle.UpdateCalle(&input, idCalle)
31              if err != nil {
32                  return "nil", err
33              }
```

```
34              successMessage := "successfully updated"
35
36              return successMessage, nil
37          }
38
39          func (r *mutationResolver) DeleteCalle(ctx context.Context,
                idCalle int) (string, error) {
40              err := r.Calle.DeleteCalle(idCalle)
41              if err != nil {
42                  return "", err
43              }
44              successMessage := "successfully deleted"
45              return successMessage, nil
46          }
```

Código A.10: Resolutores del query.resolvers.go y mutation.resolvers.go para el operador del catálogo general calles.

```
1           package models
2
3           type Cg_Codigos_Postales struct {
4               Id_Estado           int    `json:"id_estado"`
5               Id_Alcaldia_Municipio int `json:"id_alcaldia_municipio"`
6               Id_Cp_Interno       int    `json:"id_cp_interno" gorm:"
                    primary_key"`
7               Id_Cp               string `json:"id_cp"`
8           }
```

Código A.11: Modelo para el catálogo general de códigos postales.

```
1           # cg_codigos_postales
2           type Cg_codigos_postales {
3           estado: Cg_estados!
4           alcaldia_municipio: Cg_alcaldia_municipios!
5           id_cp_interno: ID!
6           id_cp: String!
7           }
```

Código A.12: Prototipo del schema.graphqls del catálogo general códigos postales.

```
1           # cg_codigos_postales
2           GetCodigoPostal(id_cp_interno: ID!): Cg_codigos_postales!
3           GetCodigoPostalPorStr(id_cp: String!): [Cg_codigos_postales
                !]!
4           GetAllCodigosPostales: [Cg_codigos_postales!]!
```

Código A.13: Firmas de tipo query del query.graphqls del catálogo general códigos postales.

```
1  package repository
2
3  import (
4      "apiCatalogosGenerales/app/models"
5      "apiCatalogosGenerales/graph/model"
6      "errors"
7      "gorm.io/gorm"
8  )
9
10 type Codigos_PostalesRepository interface {
11     // Funcion para obtener el codido postal por
           identificador unico
12     GetCodigoPostal(id_codigo_postal int) (*model.
           CgCodigosPostales, error)
13     // Funcion para obtener el codigo postal por la cadena de
            5 caracteres
14     GetCodigoPostalPorStr(id_codigo_postal string) ([]*model.
           CgCodigosPostales, error)
15     // Funcion para obtener todos los codigos postales
16     GetAllCodigosPostales() ([]*model.CgCodigosPostales,
           error)
17 }
18
19 type Codigos_PostalesService struct {
20     DB *gorm.DB
21 }
22
23 var _ Codigos_PostalesRepository = &Codigos_PostalesService{}
24
25 func NewCodigo_PostalService(db *gorm.DB) *
       Codigos_PostalesService {
26     return &Codigos_PostalesService{
27         DB: db,
28     }
29 }
30
31 func (c Codigos_PostalesService) GetCodigoPostal(
       id_codigo_postal int) (*model.CgCodigosPostales, error) {
32     cpSearch := models.Cg_Codigos_Postales{}
33     err := c.DB.Where("id_cp_interno = ?", id_codigo_postal).
           First(&cpSearch).Error
34     if err != nil {
35         return &model.CgCodigosPostales{}, errors.New("
               Identificador unico no valido. " + err.Error())
```

```
36              }
37
38              // Creacion de un "objeto" estadoService para poder
                    utilizar sus metodos
39              estadoService := EstadoService{DB: c.DB}
40              estado, err := estadoService.GetEstadoById(cpSearch.
                    Id_Estado)
41              if err != nil {
42                  return &model.CgCodigosPostales{}, err
43              }
44
45              // Creacion de un "objeto" alcaldiaService para poder
                    utilizar sus metodos
46              alcaldiaService := Alcaldias_MunicipiosService{DB: c.DB}
47              alcaldia, err := alcaldiaService.GetAlcaldiaMunicipioById
                    (cpSearch.Id_Alcaldia_Municipio)
48              if err != nil {
49                  return &model.CgCodigosPostales{}, err
50              }
51
52              codigo_postal := &model.CgCodigosPostales{
53                  IDCpInterno:     id_codigo_postal,
54                  IDCp:            cpSearch.Id_Cp,
55                  Estado:          &estado,
56                  AlcaldiaMunicipio: &alcaldia,
57              }
58              return codigo_postal, err
59          }
60
61          func (c Codigos_PostalesService) GetCodigoPostalPorStr(
                id_codigo_postal string) ([]*model.CgCodigosPostales,
                error) {
62              cpSearch := []models.Cg_Codigos_Postales{}
63              err := c.DB.Where(
64                  "id_cp = ?", id_codigo_postal,
65              ).Find(&cpSearch).Error
66
67              if err != nil {
68                  return nil, err
69              }
70
71              // Creacion de un "objeto" estadoService para poder
                    utilizar sus metodos
72              estadoService := EstadoService{DB: c.DB}
73              // Creacion de un "objeto" alcaldiaService para poder
```

```
                utilizar sus metodos
74              alcaldiaService := Alcaldias_MunicipiosService{DB: c.DB}
75
76              codigos_postales := []*model.CgCodigosPostales{}
77
78              for _, search := range cpSearch {
79                  // Se obtiene la informacion del estado
80                  estado, _ := estadoService.GetEstadoById(search.
                        Id_Estado)
81                  alcaldia, _ := alcaldiaService.
                        GetAlcaldiaMunicipioById(search.
                        Id_Alcaldia_Municipio)
82                  codigo_postal := &model.CgCodigosPostales{
83                      IDCpInterno:     search.Id_Cp_Interno,
84                      IDCp:            search.Id_Cp,
85                      Estado:          &estado,
86                      AlcaldiaMunicipio: &alcaldia,
87                  }
88                  codigos_postales = append(codigos_postales,
                        codigo_postal)
89              }
90
91              return codigos_postales, err
92          }
93
94          func (c Codigos_PostalesService) GetAllCodigosPostales() ([]*
                model.CgCodigosPostales, error) {
95              cpSearch := []models.Cg_Codigos_Postales{}
96              err := c.DB.Find(&cpSearch).Error
97
98              if err != nil {
99                  return nil, err
100             }
101
102             // Creacion de un "objeto" estadoService para poder
                    utilizar sus metodos
103             estadoService := EstadoService{DB: c.DB}
104             // Creacion de un "objeto" alcaldiaService para poder
                    utilizar sus metodos
105             alcaldiaService := Alcaldias_MunicipiosService{DB: c.DB}
106
107             codigos_postales := []*model.CgCodigosPostales{}
108
109             for _, search := range cpSearch {
110                 // Se obtiene la informacion del estado
```

```
111                 estado, _ := estadoService.GetEstadoById(search.
                        Id_Estado)
112                 alcaldia, _ := alcaldiaService.
                        GetAlcaldiaMunicipioById(search.
                        Id_Alcaldia_Municipio)
113                 codigo_postal := &model.CgCodigosPostales{
114                     IDCp:            search.Id_Cp,
115                     Estado:          &estado,
116                     AlcaldiaMunicipio: &alcaldia,
117                 }
118                 codigos_postales = append(codigos_postales,
                        codigo_postal)
119             }
120
121             return codigos_postales, err
122         }
```

Código A.14: Cuerpo de las consultas del repositorio del catálogo general usuarios tipo.

```
 1          func (r *queryResolver) GetCodigoPostal(ctx context.Context,
                idCpInterno int) (*model.CgCodigosPostales, error) {
 2              codigo_postal, err := r.Codigos_Postales.GetCodigoPostal(
                    idCpInterno)
 3              if err != nil {
 4                  return nil, err
 5              }
 6              return codigo_postal, nil
 7          }
 8
 9          func (r *queryResolver) GetCodigoPostalPorStr(ctx context.
                Context, idCp string) ([]*model.CgCodigosPostales, error)
                 {
10              codigo_postal, err := r.Codigos_Postales.
                    GetCodigoPostalPorStr(idCp)
11              if err != nil {
12                  return nil, err
13              }
14              return codigo_postal, nil
15          }
16
17          func (r *queryResolver) GetAllCodigosPostales(ctx context.
                Context) ([]*model.CgCodigosPostales, error) {
18              codigosPostales, err := r.Codigos_Postales.
                    GetAllCodigosPostales()
19              if err != nil {
```

```
20                  return nil, err
21              }
22              return codigosPostales, nil
23          }
```

Código A.15: Resolutores del query.resolvers.go para los operadores del catálogo general códigos postales.

A.2. Códigos para los catálogos de procesos

```
1       package models
2
3       import (
4           "time"
5       )
6
7       type Cpr_Roles struct {
8           Id_Rol                int      `json:"id_rol" gorm:"
                primaryKey"`
9           Nombre_Rol            string   `json:"nombre_rol"`
10          Descripcion_Rol       string   `json:"descripcion_rol"`
11          Ctl_Es_Resgistro_Activo bool   `json:"ctl_es_registro_activo
                "`
12          Ctl_Fecha_Registro    time.Time `json:"ctl_fecha_registro"`
13          Ctl_Usuario_Registro  string   `json:"ctl_usuario_registro"`
14          Ctl_Fecha_Modificacion time.Time `json:"
                ctl_fecha_modificacion"`
15          Ctl_Usuario_Modificacion int   `json:"
                ctl_usuario_modificacion"`
16      }
```

Código A.16: Modelo para el catálogo de procesos roles.

```
1       # cpr_roles
2       type Cpr_roles {
3       id_rol: ID!
4       nombre_rol: String!
5       descripcion_rol: String!
6       ctl_es_registro_activo: Boolean!
7       ctl_fecha_registro: Timestamp!
8       ctl_usuario_registro: String
9       ctl_fecha_modificacion: Timestamp
10      ctl_usuario_modificacion: Int
11      }
```

Código A.17: Prototipo del schema.graphqls del catálogo de procesos roles

```
# cpr_roles
GetRol(id_rol: ID, nombre: String): Cpr_roles!
GetAllRoles: [Cpr_roles!]!
```

Código A.18: Firmas de tipo query del query.graphqls del catálogo de procesos roles.

```
package repository

import (
    "apiCatalogosGenerales/graph/model"
    "gorm.io/gorm"
)

type RolRepository interface {
    GetRol(id_rol int, nombre_rol string) (*model.CprRoles, error)
    GetAllRoles() ([]*model.CprRoles, error)
}

type RolService struct {
    DB *gorm.DB
}

var _ RolRepository = &RolService{}

func NewRolService(db *gorm.DB) *RolService {
    return &RolService{DB: db}
}

func (r RolService) GetRol(id_rol int, nombre_rol string) (*model.CprRoles, error) {
    rolSearch := &model.CprRoles{}
    err := r.DB.Where(
        "id_rol = ?", id_rol).Or(
        "nombre_rol = ?", nombre_rol,
    ).First(&rolSearch).Error
    return rolSearch, err
}

func (r RolService) GetAllRoles() ([]*model.CprRoles, error) {
    roles := []*model.CprRoles{}
    err := r.DB.Table("cpr_roles").Find(&roles).Error
```

```
35            return roles, err
36        }
```

Código A.19: Cuerpo de las consultas del repositorio del catálogo de procesos roles.

```
1         func (r *queryResolver) GetRol(ctx context.Context, idRol *int,
              nombre *string) (*model.CprRoles, error) {
2             idRol = FillZero(idRol)
3             nombre = FillNA(nombre)
4
5             rol, err := r.Rol.GetRol(*idRol, *nombre)
6             selectedRol := &model.CprRoles{
7                 IDRol:                rol.IDRol,
8                 NombreRol:            rol.NombreRol,
9                 DescripcionRol:       rol.DescripcionRol,
10                CtlEsRegistroActivo:  rol.CtlEsRegistroActivo,
11                CtlFechaRegistro:     rol.CtlFechaRegistro,
12                CtlUsuarioRegistro:   rol.CtlUsuarioRegistro,
13                CtlFechaModificacion: rol.CtlFechaModificacion,
14                CtlUsuarioModificacion: rol.CtlUsuarioModificacion,
15            }
16            if err != nil {
17                return nil, err
18            }
19            return selectedRol, nil
20        }
21
22        func (r *queryResolver) GetAllRoles(ctx context.Context) ([]*
              model.CprRoles, error) {
23            roles, err := r.Rol.GetAllRoles()
24            if err != nil {
25                return nil, err
26            }
27            return roles, nil
28        }
```

Código A.20: Resolutores del query.resolvers.go para los operadores del catálogo de procesos roles.

```
1         package models
2
3         import "time"
4
5         type Cpi_Empresas_identificadores struct {
6             IdEmpresaIdentificador int    `json:"id_empresa_identificador
                  " gorm:"primaryKey"`
```

```
7           IdEmpresa            int      `json:"id_empresa"`
8           TextoIdentificador   string   `json:"texto_identificador"`
9           TipoIdentificador    string   `json:"tipo_identificador"`
10          OtroTipoIdentificador string  `json:"otro_tipo_identificador"`

11          CtlEsRegistroActivo  bool      `json:"ctl_es_registro_activo"`
12          CtlFechaRegistro     time.Time `json:"ctl_fecha_registro"`
13          CtlUsuarioRegistro   string   `json:"ctl_usuario_registro"`
14          CtlFechaModificacion time.Time `json:"ctl_fecha_modificacion"
                `
15          CtlUsuarioModificacion int    `json:"ctl_usuario_modificacion
                "`
16      }
```

Código A.21: Modelo para el catálogo de procesos empresas identificadores.

```
1       # cpi_empresas_identificadores
2       type Cpi_empresas_identificadores {
3       id_empresa_identificador: ID!
4       id_empresa: Int!
5       texto_identificador: String!
6       tipo_identificador: String!
7       otro_tipo_identificador: String
8       ctl_es_registro_activo: Boolean!
9       ctl_fecha_registro: Timestamp!
10      ctl_usuario_registro: String
11      ctl_fecha_modificacion: Timestamp
12      ctl_usuario_modificacion: Int
13      }
```

Código A.22: Prototipo del schema.graphqls del catálogo de procesos empresas identificadores.

```
1       GetEmpresaIdentificador(id_empresa_identificador: ID):
            Cpi_empresas_identificadores!
2       GetEmpresaIdentificadoresPorEmpresa(id_empresa: ID): [
            Cpi_empresas_identificadores!]!
3       CreateEmpresaIdentificador(input: EmpresasIdentificadoresInput!):
             String!
4       UpdateEmpresaIdentificador(id_empresa_identificador: ID!, input:
             EmpresasIdentificadoresInput!): String!
5       DeleteEmpresaIdentificador(id_empresa_identificador: ID!):
            String!
```

Código A.23: Firmas de tipo query del query.graphqls y mutation.graphqls del catálogo de procesos empresas identificadores.

```
1   package repository
2
3   import (
4       "apiCatalogosGenerales/app"
5       "apiCatalogosGenerales/app/models"
6       "apiCatalogosGenerales/graph/model"
7       "errors"
8       "net/mail"
9       "regexp"
10      "strings"
11      "time"
12
13      "gorm.io/gorm"
14  )
15
16  type EmpresasIdentificadoresRepository interface {
17      CreateEmpresaIdentificador(EmpresaIdentificadorInput *model.
            EmpresasIdentificadoresInput, tx *gorm.DB) (*models.
            Cpi_Empresas_identificadores, error)
18      GetEmpresaIdentificador(id_empresa_identificador int) (*model.
            CpiEmpresasIdentificadores, error)
19      GetEmpresaIdentificadoresPorEmpresa(id_empresa int) ([]*model.
            CpiEmpresasIdentificadores, error)
20      UpdateEmpresaIdentificador(EmpresaIdentificadorInput *model.
            EmpresasIdentificadoresInput, id_empresa_identificador
            int) error
21      DeleteEmpresaIdentificador(id_empresa_identificador int)
            error
22  }
23
24  type EmpresaIndentificadoresService struct {
25      DB *gorm.DB
26  }
27
28  var _ EmpresasIdentificadoresRepository = &
        EmpresaIndentificadoresService{}
29
30  func NewEmpresaIdentificadoresService(db *gorm.DB) *
        EmpresaIndentificadoresService {
31      return &EmpresaIndentificadoresService{DB: db}
32  }
33
34  func (e EmpresaIndentificadoresService)
        CreateEmpresaIdentificador(EmpresaIdentificadorInput *model.
```

```
        EmpresasIdentificadoresInput, tx *gorm.DB) (*models.
        Cpi_Empresas_identificadores, error) {
35      EmpresaIdentificadorInput.IDEmpresa = esVacioEntero(
            EmpresaIdentificadorInput.IDEmpresa)
36      //EmpresaIdentificadorInput.TextoIdentificador = *app.
            EmptyString(EmpresaIdentificadorInput.TextoIdentificador)
37      //EmpresaIdentificadorInput.TipoIdentificador = *app.
            EmptyString(EmpresaIdentificadorInput.TipoIdentificador)
38      *EmpresaIdentificadorInput.OtroTipoIdentificador = *app.
            EmptyString(*EmpresaIdentificadorInput.
            OtroTipoIdentificador) //este si puede ser null
39
40      textoID := EmpresaIdentificadorInput.TextoIdentificador
41      /*
42          Email, celular, whatsapp, url-homepage, twitter,
                instagram, facebook
43      */
44      //Opcion 1: Checar tipo de identificador y despues checar si
            es valido
45      tipoID := strings.ToLower(EmpresaIdentificadorInput.
            TipoIdentificador)
46
47      switch {
48      case tipoID == "email":
49          _, errC := mail.ParseAddress(textoID)
50          if errC != nil {
51              return nil, errors.New("El correo no es valido.")
52          }
53      case tipoID == "instagram":
54          i, _ := regexp.Match(`([a-z0-9_\.]([a-z0-9_\.]{0,28}(a-z0
                -9_\.)))`, []byte(textoID))
55          if i != true {
56              return nil, errors.New("El nombre de usuario no es
                    valido.")
57          }
58      case tipoID == "twitter":
59          i, _ := regexp.Match(`(@([A-Za-z0-9]{0,15}))`, []byte(
                textoID))
60          if i != true {
61              return nil, errors.New("El nombre de usuario no es
                    valido.")
62          }
63      case tipoID == "url":
64          i, _ := regexp.Match(`^(https:|http:|www\.)\S*`, []byte(
                textoID))
```

```
65              if i != true {
66                  return nil, errors.New("La URL no es valida.")
67              }
68          case tipoID == "celular":
69              i, _ := regexp.Match(`([0-9]{10})`, []byte(textoID))
70
71              if i != true { //&& len(textoID) != 10{
72                  return nil, errors.New("El numero proporcionado no es
                        valido.")
73              }
74          case tipoID == "whatsapp":
75              i, _ := regexp.Match(`(\+52([0-9]{12}))`, []byte(textoID)
                    )
76              if i != true {
77                  return nil, errors.New("El Whatsapp proporcionado no
                        es valido (+52).")
78              }
79          case tipoID == "facebook":
80              i, _ := regexp.Match(`((https:|http)//){0,1}(www.){0,1}(
                    facebook.com/)\S*`, []byte(textoID))
81              if i != true {
82                  return nil, errors.New("El url de la pagina de
                        Facebook proporcionada no es valida.")
83              }
84          }
85
86          //Opcion 2: Utilizar el texto identificador para asignar el
                tipo de identificador
87
88          empresaIdentificador := &models.Cpi_Empresas_identificadores{
89              IdEmpresa:           EmpresaIdentificadorInput.IDEmpresa,
90              TextoIdentificador:  EmpresaIdentificadorInput.
                    TextoIdentificador,
91              TipoIdentificador:   EmpresaIdentificadorInput.
                    TipoIdentificador,
92              OtroTipoIdentificador: *EmpresaIdentificadorInput.
                    OtroTipoIdentificador,
93          }
94
95          err := tx.Select("id_empresa", "texto_identificador", "
                tipo_identificador", "otro_tipo_identificador").Create(&
                empresaIdentificador).Error
96
97          return empresaIdentificador, err
98      }
```

```
99
100         func (e EmpresaIndentificadoresService) GetEmpresaIdentificador(
                id_empresa_identificador int) (*model.
                CpiEmpresasIdentificadores, error) {
101             empresaIdentificadorBusqueda := &model.
                    CpiEmpresasIdentificadores{}
102             err := e.DB.Where("id_empresa_identificador = ?",
                    id_empresa_identificador).First(&
                    empresaIdentificadorBusqueda).Error
103
104             return empresaIdentificadorBusqueda, err
105         }
106
107         func (e EmpresaIndentificadoresService)
                GetEmpresaIdentificadoresPorEmpresa(id_empresa int) ([]*model.
                CpiEmpresasIdentificadores, error) {
108             empresaIdentificadoresBusqueda := []*model.
                    CpiEmpresasIdentificadores{}
109             err := e.DB.Table("cpi_empresas_identificadores").Where("
                    id_empresa = ?", id_empresa).Find(&
                    empresaIdentificadoresBusqueda).Error
110
111             return empresaIdentificadoresBusqueda, err
112
113         }
114
115         func (e EmpresaIndentificadoresService)
                UpdateEmpresaIdentificador(EmpresaIdentificadorInput *model.
                EmpresasIdentificadoresInput, id_empresa_identificador int)
                error {
116             empresaIdentificador := &models.Cpi_Empresas_identificadores{
117                 IdEmpresa:           EmpresaIdentificadorInput.IDEmpresa,
118                 TextoIdentificador:  EmpresaIdentificadorInput.
                        TextoIdentificador,
119                 TipoIdentificador:   EmpresaIdentificadorInput.
                        TipoIdentificador,
120                 OtroTipoIdentificador: *EmpresaIdentificadorInput.
                        OtroTipoIdentificador,
121                 CtlEsRegistroActivo:  false,
122                 CtlFechaRegistro:     time.Time{},
123                 CtlUsuarioRegistro:   "",
124                 CtlFechaModificacion: time.Time{},
125                 CtlUsuarioModificacion: 0,
126             }
127             err := e.DB.Model(&empresaIdentificador).Where("
```

```
                id_empresa_identificador = ?", id_empresa_identificador).
                Updates(empresaIdentificador).Error
128         return err
129     }
130
131     func (e EmpresaIndentificadoresService)
            DeleteEmpresaIdentificador(id_empresa_identificador int)
            error {
132         empresaIdentificador := &models.Cpi_Empresas_identificadores
                {}
133         err := e.DB.Delete(&empresaIdentificador,
                id_empresa_identificador).Error
134         return err
135     }
```

Código A.24: Cuerpo de las consultas del repositorio del catálogo de procesos empresas identificadores.

```
 1      func (r *queryResolver) GetEmpresaIdentificador(ctx context.
            Context, idEmpresaIdentificador *int) (*model.
            CpiEmpresasIdentificadores, error) {
 2          idEmpresaIdentificador = FillZero(idEmpresaIdentificador)
 3          empresaIdentificador, err := r.EmpresaIdentificador.
                GetEmpresaIdentificador(*idEmpresaIdentificador)
 4
 5          seleccionEmpresaIdentificador := &model.
                CpiEmpresasIdentificadores{
 6              IDEmpresaIdentificador: empresaIdentificador.
                    IDEmpresaIdentificador,
 7              IDEmpresa:            empresaIdentificador.IDEmpresa,
 8              TextoIdentificador:   empresaIdentificador.
                    TextoIdentificador,
 9              TipoIdentificador:    empresaIdentificador.
                    TipoIdentificador,
10              OtroTipoIdentificador: empresaIdentificador.
                    OtroTipoIdentificador,
11              CtlEsRegistroActivo:  empresaIdentificador.
                    CtlEsRegistroActivo,
12              CtlFechaRegistro:     empresaIdentificador.
                    CtlFechaRegistro,
13              CtlUsuarioRegistro:   empresaIdentificador.
                    CtlUsuarioRegistro,
14              CtlFechaModificacion: empresaIdentificador.
                    CtlFechaModificacion,
15              CtlUsuarioModificacion: empresaIdentificador.
```

```
                CtlUsuarioModificacion,
16          }
17          if err != nil {
18              return nil, err
19          }
20          return seleccionEmpresaIdentificador, nil
21      }
22
23      func (r *queryResolver) GetEmpresaIdentificadoresPorEmpresa(ctx
            context.Context, idEmpresa *int) ([]*model.
            CpiEmpresasIdentificadores, error) {
24          empresasIdentificadores, err := r.EmpresaIdentificador.
                GetEmpresaIdentificadoresPorEmpresa(*idEmpresa)
25          if err != nil {
26              return nil, err
27          }
28
29          return empresasIdentificadores, nil
30      }
31      func (r *mutationResolver) CreateEmpresaIdentificador(ctx
            context.Context, input model.EmpresasIdentificadoresInput) (
            string, error) {
32          empresaIndentificador, err := r.EmpresaIdentificador.
                CreateEmpresaIdentificador(&input)
33          ctlFechaModificacion := empresaIndentificador.
                CtlFechaModificacion.String()
34
35          empresaIdentificadorCreada := &model.
                CpiEmpresasIdentificadores{
36              IDEmpresaIdentificador: empresaIndentificador.
                    IdEmpresaIdentificador,
37              IDEmpresa:           empresaIndentificador.IdEmpresa,
38              TextoIdentificador:  empresaIndentificador.
                    TextoIdentificador,
39              TipoIdentificador:   empresaIndentificador.
                    TipoIdentificador,
40              OtroTipoIdentificador: &empresaIndentificador.
                    OtroTipoIdentificador,
41              CtlEsRegistroActivo: empresaIndentificador.
                    CtlEsRegistroActivo,
42              CtlFechaRegistro:    empresaIndentificador.
                    CtlFechaRegistro.String(),
43              CtlUsuarioRegistro:  &empresaIndentificador.
                    CtlUsuarioRegistro,
44              CtlFechaModificacion: &ctlFechaModificacion,
```

```
45              CtlUsuarioModificacion: &empresaIndentificador.
                    CtlUsuarioModificacion,
46          }
47          if err != nil {
48              return nil, err
49          }
50
51          return "empresaIdentificadorCreada", nil
52      }
53
54      func (r *mutationResolver) UpdateEmpresaIdentificador(ctx
            context.Context, idEmpresaIdentificador int, input model.
            EmpresasIdentificadoresInput) (string, error) {
55          //panic(fmt.Errorf("not implemented"))
56          err := r.EmpresaIdentificador.UpdateEmpresaIdentificador(&
                input, idEmpresaIdentificador)
57          if err != nil {
58              return "nil", err
59          }
60
61          actualizacionExitosa := "Actualizacion exitosa"
62
63          return actualizacionExitosa, nil
64      }
65
66      func (r *mutationResolver) DeleteEmpresaIdentificador(ctx
            context.Context, idEmpresaIdentificador int) (string, error)
            {
67          err := r.EmpresaIdentificador.DeleteEmpresaIdentificador(
                idEmpresaIdentificador)
68          if err != nil {
69              return "", err
70          }
71          borradoExitoso := "Borrado exitoso"
72          return borradoExitoso, nil
73      }
```

Código A.25: Resolutores del query.resolvers.go y mutation.resolvers.go para los operadores del catálogo de procesos de empresas identificadores.

```
1       package models
2
3       type Cpr_estatus_proceso struct {
4           Id_estatus    string `json:"id_estatus" gorm:"primaryKey"`
5           Descripcion   string `json:"descripcion"`
```

```
6           Codigo_estatus int  `json:"codigo_estatus"`
7       }
```

Código A.26: Modelo para el catálogo de procesos estatus procesos.

```
1       # cpr_estatus_procesos
2       type Cpr_estatus_proceso{
3           id_estatus: String!
4           descripcion: String!
5           codigo_estatus: Int
6       }
```

Código A.27: Prototipo del schema.graphqls del catálogo de procesos estatus procesos.

```
1       GetEstatusProceso(id_estatus: String): Cpr_estatus_proceso!
2       UpdateEstatusProceso(id_estatus: String!, input:
            EstatusProcesoInput!): String!
3       DeleteEstatusProceso(id_estatus: String!): String!
```

Código A.28: Firmas de tipo query del query.graphqls del catálogo de procesos estatus procesos.

```
1       package repository
2
3       import (
4           "apiCatalogosGenerales/app/models"
5           "apiCatalogosGenerales/graph/model"
6           "gorm.io/gorm"
7       )
8
9       type EstatusProcesoRepository interface {
10          GetEstatusProceso(id_estatus_proceso string) (*model.
                CprEstatusProceso, error)
11          UpdateEstatusProceso(id_estatus_proceso string,
                estatusProcesoInput *model.EstatusProcesoInput) error
12          DeleteEstatusProceso(id_estatus_proceso string) error
13      }
14
15      type EstatusProcesoService struct {
16          DB *gorm.DB
17      }
18
19      func NewEstatusProcesoService(db *gorm.DB) *
            EstatusProcesoService {
20          return &EstatusProcesoService{DB: db}
21      }
```

```
22
23          var _ EstatusProcesoRepository = &EstatusProcesoService{}
24
25          func (e EstatusProcesoService) GetEstatusProceso(
                id_estatus_proceso string) (*model.CprEstatusProceso, error)
                {
26              estatusProcesoBusqueda := &model.CprEstatusProceso{}
27              err := e.DB.Where("id_estatus = ?", id_estatus_proceso).First
                    (&estatusProcesoBusqueda).Error
28
29              return estatusProcesoBusqueda, err
30          }
31
32          func (e EstatusProcesoService) UpdateEstatusProceso(
                id_estatus_proceso string, estatusProcesoInput *model.
                EstatusProcesoInput) error {
33              estatusProceso := &models.Cpr_estatus_proceso{
34                  Id_estatus:    estatusProcesoInput.IDEstatus,
35                  Descripcion:   estatusProcesoInput.Descripcion,
36                  Codigo_estatus: *estatusProcesoInput.CodigoEstatus,
37              }
38              err := e.DB.Model(&estatusProceso).Where("id_estatus = ?",
                    id_estatus_proceso).Updates(estatusProceso).Error
39
40              return err
41          }
42
43          func (e EstatusProcesoService) DeleteEstatusProceso(
                id_estatus_proceso string) error {
44              estatusProceso := &models.Cpr_estatus_proceso{}
45              err := e.DB.Delete(&estatusProceso, id_estatus_proceso).Error
46
47              return err
48          }
```

Código A.29: Cuerpo de las consultas del repositorio del catálogo de procesos estatus procesos.

```
1           func (r *queryResolver) GetEstatusProceso(ctx context.Context,
                idEstatus *string) (*model.CprEstatusProceso, error) {
2               estatusProceso, err := r.EstatusProceso.GetEstatusProceso(*
                    idEstatus)
3               if err != nil {
4                   return nil, err
5               }
```

```
6           return estatusProceso, nil
7       }
8       func (r *mutationResolver) UpdateEstatusProceso(ctx context.
            Context, idEstatus string, input model.EstatusProcesoInput) (
            string, error) {
9           err := r.EstatusProceso.UpdateEstatusProceso(idEstatus, &
                input)
10          if err != nil {
11              return "nil", err
12          }
13
14          mensajeExito := "Actualizacion exitosa"
15
16          return mensajeExito, nil
17      }
18
19      func (r *mutationResolver) DeleteEstatusProceso(ctx context.
            Context, idEstatus string) (string, error) {
20          err := r.EstatusProceso.DeleteEstatusProceso(idEstatus)
21          if err != nil {
22              return "nil", err
23          }
24          mensajeExito := "Borrado exitoso"
25          return mensajeExito, nil
26      }
```

Código A.30: Resolutores del query.resolvers.go y mutation.resolvers.go para los operadores del catálogo de procesos estatus procesos.

```
1       package models
2
3       import (
4           "github.com/guregu/null"
5           "time"
6       )
7
8       type P1_ofertas struct {
9           Id_oferta              int        `json:"id_oferta" gorm:"
                primaryKey"`
10          Tipo_oferta            null.Int   `json:"tipo_oferta"`
11          Id_empresa             null.Int   `json:"id_empresa"`
12          Id_servicio_empresa    null.Int   `json:"id_servicio_empresa"`
13          Titulo_oferta          string     `json:"titulo_oferta"`
14          Descripcion            null.String `json:"descripcion"`
15          Nota_descripcion       null.String `json:"nota_descripcion"`
```

```
16          Id_estatus            null.String `json:"id_estatus"`
17          Fecha_inicio_oferta   time.Time  `json:"fecha_inicio_oferta"`
18          Fecha_fin_oferta      time.Time  `json:"fecha_fin_oferta"`
19          Id_domicilio_origen   null.Int   `json:"id_domicilio_origen"`
20          Id_residuo_autorizado null.Int   `json:"id_residuo_autorizado
                "`
21          Residuo_cantidad      null.Int   `json:"residuo_cantidad"`
22          Residuo_unidad        null.Int   `json:"residuo_unidad"`
23          Id_servicio           null.Int   `json:"id_servicio"`
24          Ctl_es_registro_activo null.Bool `json:"
                ctl_es_registro_activo"`
25          Ctl_fecha_registro    time.Time  `json:"ctl_fecha_registro"`
26          Ctl_usuario_registro  null.String `json:"
                ctl_usuario_registro"`
27          Ctl_fecha_modificacion time.Time `json:"
                ctl_fecha_modificacion"`
28          Ctl_usuario_modificacion int     `json:"
                ctl_usuario_modificacion"`
29          Ruta_foto01_residuos  null.String `json:"
                ruta_foto01_residuos"`
30          Ruta_foto02_residuos  null.String `json:"
                ruta_foto02_residuos"`
31      }
```

Código A.31: Modelo para el catálogo de procesos P1 ofertas.

```
1       # p1_ofertas
2       type P1_ofertas{
3           id_oferta: ID!
4           tipo_oferta: Int
5           empresa: Cpi_empresas
6           servicio_empresa: Cpi_servicios_empresas
7           titulo_oferta: String!
8           descripcion: String
9           nota_descripcion: String
10          estatus: Cpr_estatus_proceso
11          fecha_inicio_oferta: Timestamp
12          fecha_fin_oferta: Timestamp
13          domicilio_origen: Cpi_domicilios
14          residuo_autorizado: Cpr_residuos_autorizados
15          residuo_cantidad: Int
16          residuo_unidad: Int
17          servicio: Cpr_servicios
18          ctl_es_registro_activo: Boolean!
19          ctl_fecha_registro: Timestamp!
```

```
20          ctl_usuario_registro: String
21          ctl_fecha_modificacion: Timestamp!
22          ctl_usuario_modificacion: Int
23          ruta_foto01_residuos: String
24          ruta_foto02_residuos: String
25      }
```

Código A.32: Prototipo del schema.graphqls del catálogo de procesos P1 ofertas.

```
1       GetOfertas(id_oferta: ID!): P1_ofertas!
2       GetOfertaPorResiduo(id_residuo_autorizado: ID!): [P1_ofertas!]!
3       GetOfertaPorServicio(id_servicio: ID!): [P1_ofertas!]!
4       CreatePOferta(input: P1OfertasInput!): ID!
5       UpdatePOferta(id_oferta: ID!, input: P1OfertasInput!): String!
6       DeletePOferta(id_oferta: ID!): String!
```

Código A.33: Firmas de tipo query del query.graphqls del catálogo de procesos P1 ofertas.

```
1       package repository
2
3       import (
4           "apiCatalogosGenerales/app"
5           "apiCatalogosGenerales/app/models"
6           "apiCatalogosGenerales/graph/model"
7           "errors"
8           "github.com/guregu/null"
9           "gorm.io/gorm"
10          "time"
11      )
12
13      type P1OfertasRepository interface {
14          GetOferta(id_oferta int) (*model.P1Ofertas, error)
15          GetOfertaPorResiduo(id_residuo_autorizado int) ([]*model.
                P1Ofertas, error)
16          GetOfertaPorServicio(id_servicio int) ([]*model.P1Ofertas,
                error)
17          CreatePOferta(input *model.P1OfertasInput, tx *gorm.DB) (*
                models.P1_ofertas, error)
18          UpdatePOferta(id_oferta int, input *model.P1OfertasInput)
                error
19          DeletePOferta(id_oferta int) error
20      }
21
22      type P1OfertaService struct {
23          DB *gorm.DB
24      }
```

```
25
26          var _ P1OfertasRepository = &P1OfertaService{}
27
28          func NewP1OfertasService(db *gorm.DB) *P1OfertaService {
29              return &P1OfertaService{
30                  DB: db,
31              }
32          }
33
34          func (p P1OfertaService) GetOferta(id_oferta int) (*model.
                P1Ofertas, error) {
35              ofertaBusqueda := &models.P1_ofertas{}
36              err := p.DB.Where("id_oferta = ?", id_oferta).First(&
                    ofertaBusqueda).Error
37              if err != nil {
38                  return nil, err
39              }
40              //Busqueda para la estructura de empresa
41              empresaService := EmpresaService{DB: p.DB}
42              empresa, _ := empresaService.GetEmpresa(int(ofertaBusqueda.
                    Id_empresa.Int64), "")
43              //Busqueda para la estructura de roles empresa
44              servicioEmpresaServise := Servicios_EmpresasService{DB: p.DB}
45              servicioEmpresa, _ := servicioEmpresaServise.
                    GetServicioEmpresaById(int(ofertaBusqueda.
                    Id_servicio_empresa.Int64))
46              //Busqueda para la estructura de estatus procesos
47              estatusProcesosService := EstatusProcesoService{DB: p.DB}
48              estatusProcesos, _ := estatusProcesosService.
                    GetEstatusProceso(ofertaBusqueda.Id_estatus.String)
49              //Manejo de fechas
50              fechaFin := ofertaBusqueda.Fecha_fin_oferta.String()
51              fechaInicio := ofertaBusqueda.Fecha_inicio_oferta.String()
52              //Conversion para el tipo de solicitud
53              tipoOferta := int(ofertaBusqueda.Tipo_oferta.Int64)
54              //Busqueda para la estructura de domicilios
55              domicilioService := DomicilioService{DB: p.DB}
56              domiclio, _ := domicilioService.GetDomicilio(int(
                    ofertaBusqueda.Id_domicilio_origen.Int64))
57
58              var residuoAutorizado *model.CprResiduosAutorizados
59              var residuoCantidad, residuoUnidad int
60              var servicio *model.CprServicios
61
62              if tipoOferta == 1 { /*residuo*/
```

```
63              //Busqueda para la estructura de residuo
64              residuoAutorizadoService := Residuos_AutorizadosService{
                    DB: p.DB}
65              residuoAutorizado, _ = residuoAutorizadoService.
                    GetResiduoAutorizado(int(ofertaBusqueda.
                    Id_residuo_autorizado.Int64))
66              //Conversion de la unidad
67              residuoCantidad = int(ofertaBusqueda.Residuo_cantidad.
                    Int64)
68              residuoUnidad = int(ofertaBusqueda.Residuo_unidad.Int64)
69          } else { /*servicio*/
70              servicioService := ServicioService{DB: p.DB}
71              servicio, _ = servicioService.GetServicio(int(
                    ofertaBusqueda.Id_servicio.Int64), "")
72          }
73
74          oferta := &model.P1Ofertas{
75              IDOferta:             ofertaBusqueda.Id_oferta,
76              TipoOferta:           &tipoOferta,
77              Empresa:              empresa,
78              ServicioEmpresa:      servicioEmpresa,
79              TituloOferta:         ofertaBusqueda.Titulo_oferta,
80              Descripcion:          &ofertaBusqueda.Descripcion.String,
81              NotaDescripcion:      &ofertaBusqueda.Nota_descripcion.
                    String,
82              Estatus:              estatusProcesos,
83              FechaInicioOferta:    &fechaInicio,
84              FechaFinOferta:       &fechaFin,
85              DomicilioOrigen:      domiclio,
86              ResiduoAutorizado:    residuoAutorizado,
87              ResiduoCantidad:      &residuoCantidad,
88              ResiduoUnidad:        &residuoUnidad,
89              Servicio:             servicio,
90              CtlEsRegistroActivo:  ofertaBusqueda.
                    Ctl_es_registro_activo.Bool,
91              CtlFechaRegistro:     ofertaBusqueda.Ctl_fecha_registro.
                    String(),
92              CtlUsuarioRegistro:   &ofertaBusqueda.Ctl_usuario_registro
                    .String,
93              CtlFechaModificacion: ofertaBusqueda.
                    Ctl_fecha_modificacion.String(),
94              CtlUsuarioModificacion: &ofertaBusqueda.
                    Ctl_usuario_modificacion,
95          }
96
```

```
97              return oferta, err
98
99          }
100
101         func (p P1OfertaService) GetOfertaPorResiduo(
                id_residuo_autorizado int) ([]*model.P1Ofertas, error) {
102             ofertaBusqueda := []*models.P1_ofertas{}
103             err := p.DB.Where("id_residuo_autorizado = ?",
                    id_residuo_autorizado).Find(&ofertaBusqueda).Error
104             if err != nil {
105                 return nil, err
106             }
107             empresaService := EmpresaService{DB: p.DB}
108             rolesEmpresasService := Servicios_EmpresasService{DB: p.DB}
109             estatusProcesosService := EstatusProcesoService{DB: p.DB}
110             domicilioService := DomicilioService{DB: p.DB}
111             residuoAutorizadoService := Residuos_AutorizadosService{DB: p.
                    DB}
112
113             ofertas := []*model.P1Ofertas{}
114
115             for i := 0; i < len(ofertaBusqueda); i++ {
116                 empresa, _ := empresaService.GetEmpresa(int(
                        ofertaBusqueda[i].Id_empresa.Int64), "")
117                 servicioEmpresa, _ := rolesEmpresasService.
                        GetServicioEmpresaById(int(ofertaBusqueda[i].
                        Id_servicio_empresa.Int64))
118                 estatusProceso, _ := estatusProcesosService.
                        GetEstatusProceso(ofertaBusqueda[i].Id_estatus.String)
119                 fechaInicio := ofertaBusqueda[i].Fecha_inicio_oferta.
                        String()
120                 fechaFin := ofertaBusqueda[i].Fecha_fin_oferta.String()
121                 tipoOferta := int(ofertaBusqueda[i].Tipo_oferta.Int64)
122                 domiclioOrigen, _ := domicilioService.GetDomicilio(int(
                        ofertaBusqueda[i].Id_domicilio_origen.Int64))
123                 residuoAutorizado, _ := residuoAutorizadoService.
                        GetResiduoAutorizado(int(ofertaBusqueda[i].
                        Id_residuo_autorizado.Int64))
124                 residuoCantidad := int(ofertaBusqueda[i].Residuo_cantidad.
                        Int64)
125                 residuoUnidad := int(ofertaBusqueda[i].Residuo_unidad.
                        Int64)
126
127                 oferta := &model.P1Ofertas{
128                     IDOferta:             ofertaBusqueda[i].Id_oferta,
```

```
129                 TipoOferta:           &tipoOferta,
130                 Empresa:              empresa,
131                 ServicioEmpresa:      servicioEmpresa,
132                 TituloOferta:         ofertaBusqueda[i].Titulo_oferta,
133                 Descripcion:          &ofertaBusqueda[i].Descripcion.
                        String,
134                 NotaDescripcion:      &ofertaBusqueda[i].
                        Nota_descripcion.String,
135                 Estatus:              estatusProceso,
136                 FechaInicioOferta:    &fechaInicio,
137                 FechaFinOferta:       &fechaFin,
138                 DomicilioOrigen:      domiclioOrigen,
139                 ResiduoAutorizado:    residuoAutorizado,
140                 ResiduoCantidad:      &residuoCantidad,
141                 ResiduoUnidad:        &residuoUnidad,
142                 CtlEsRegistroActivo:  ofertaBusqueda[i].
                        Ctl_es_registro_activo.Bool,
143                 CtlFechaRegistro:     ofertaBusqueda[i].
                        Ctl_fecha_registro.String(),
144                 CtlUsuarioRegistro:   &ofertaBusqueda[i].
                        Ctl_usuario_registro.String,
145                 CtlFechaModificacion: ofertaBusqueda[i].
                        Ctl_fecha_modificacion.String(),
146                 CtlUsuarioModificacion: &ofertaBusqueda[i].
                        Ctl_usuario_modificacion,
147             }
148
149             ofertas = append(ofertas, oferta)
150
151         }
152
153         return ofertas, err
154
155     }
156
157     func (p P1OfertaService) GetOfertaPorServicio(id_servicio int)
            ([]*model.P1Ofertas, error) {
158         ofertaBusqueda := []*models.P1_ofertas{}
159         err := p.DB.Where("id_servicio = ?", id_servicio).Find(&
                ofertaBusqueda).Error
160         if err != nil {
161             return nil, err
162         }
163
164         empresaService := EmpresaService{DB: p.DB}
```

```
165             rolesEmpresasService := Servicios_EmpresasService{DB: p.DB}
166             estatusProcesosService := EstatusProcesoService{DB: p.DB}
167             domicilioService := DomicilioService{DB: p.DB}
168             servicioService := ServicioService{DB: p.DB}
169
170             ofertas := []*model.P1Ofertas{}
171
172             for i := 0; i < len(ofertaBusqueda); i++ {
173                 empresa, _ := empresaService.GetEmpresa(int(
                        ofertaBusqueda[i].Id_empresa.Int64), "")
174                 servicioEmpresa, _ := rolesEmpresasService.
                        GetServicioEmpresaById(int(ofertaBusqueda[i].
                        Id_servicio_empresa.Int64))
175                 estatusProceso, _ := estatusProcesosService.
                        GetEstatusProceso(ofertaBusqueda[i].Id_estatus.String)
176                 fechaInicio := ofertaBusqueda[i].Fecha_inicio_oferta.
                        String()
177                 fechaFin := ofertaBusqueda[i].Fecha_fin_oferta.String()
178                 tipoOferta := int(ofertaBusqueda[i].Tipo_oferta.Int64)
179                 domiclioOrigen, _ := domicilioService.GetDomicilio(int(
                        ofertaBusqueda[i].Id_domicilio_origen.Int64))
180                 servicio, _ := servicioService.GetServicio(int(
                        ofertaBusqueda[i].Id_servicio.Int64), "")
181
182                 oferta := &model.P1Ofertas{
183                     IDOferta:            ofertaBusqueda[i].Id_oferta,
184                     TipoOferta:          &tipoOferta,
185                     Empresa:             empresa,
186                     ServicioEmpresa:     servicioEmpresa,
187                     TituloOferta:        ofertaBusqueda[i].Titulo_oferta,
188                     Descripcion:         &ofertaBusqueda[i].Descripcion.
                            String,
189                     NotaDescripcion:     &ofertaBusqueda[i].
                            Nota_descripcion.String,
190                     Estatus:             estatusProceso,
191                     FechaInicioOferta:   &fechaInicio,
192                     FechaFinOferta:      &fechaFin,
193                     DomicilioOrigen:     domiclioOrigen,
194                     Servicio:            servicio,
195                     CtlEsRegistroActivo: ofertaBusqueda[i].
                            Ctl_es_registro_activo.Bool,
196                     CtlFechaRegistro:    ofertaBusqueda[i].
                            Ctl_fecha_registro.String(),
197                     CtlUsuarioRegistro:  &ofertaBusqueda[i].
                            Ctl_usuario_registro.String,
```

```
198                 CtlFechaModificacion: ofertaBusqueda[i].
                        Ctl_fecha_modificacion.String(),
199                 CtlUsuarioModificacion: &ofertaBusqueda[i].
                        Ctl_usuario_modificacion,
200                 RutaFoto01Residuos:   &ofertaBusqueda[i].
                        Ruta_foto01_residuos.String,
201                 RutaFoto02Residuos:   &ofertaBusqueda[i].
                        Ruta_foto02_residuos.String,
202             }
203
204             ofertas = append(ofertas, oferta)
205         }
206         return ofertas, err
207     }
208
209     func (p P1OfertaService) CreatePOferta(input *model.
            P1OfertasInput, tx *gorm.DB) (*models.P1_ofertas, error) {
210         oferta := &models.P1_ofertas{
211             Tipo_oferta:       null.IntFrom(int64(*input.TipoOferta)),
212             Id_empresa:        null.IntFrom(int64(*input.IDEmpresa)),
213             Id_servicio_empresa: null.IntFrom(int64(*input.
                    IDServicioEmpresa)),
214             Titulo_oferta:     input.TituloOferta,
215             Descripcion:       null.StringFromPtr(input.Descripcion),
216             Nota_descripcion:  null.StringFromPtr(input.
                    NotaDescripcion),
217             Id_estatus:        null.StringFromPtr(input.IDEstatus),
218             Id_domicilio_origen: null.IntFrom(int64(*input.
                    IDDomicilioOrigen)),
219         }
220
221         var err error
222
223         if *input.TipoOferta == 1 { /*residuo*/
224             oferta.Id_residuo_autorizado = null.IntFrom(int64(*input.
                    IDResiduoAutorizado))
225             oferta.Residuo_cantidad = null.IntFrom(int64(*input.
                    ResiduoCantidad))
226             oferta.Residuo_unidad = null.IntFrom(int64(*input.
                    ResiduoUnidad))
227             oferta.Ruta_foto01_residuos = null.StringFromPtr(input.
                    RutaFoto01Residuos)
228             oferta.Ruta_foto02_residuos = null.StringFromPtr(input.
                    RutaFoto02Residuos)
```

```
229                 err = tx.Select("tipo_oferta", "id_empresa", "
                        id_rol_empresa", "titulo_oferta", "descripcion", "
                        nota_descripcion", "id_estatus", "fecha_inicio_oferta"
                        , "fecha_fin_oferta", "id_domicilio_origen", "
                        id_residuo_autorizado", "residuo_cantidad", "
                        residuo_unidad").Create(&oferta).Error
230             } else if *input.TipoOferta == 2 { /*servicio*/
231                 oferta.Id_servicio = null.IntFrom(int64(*input.IDServicio
                        ))
232                 err = tx.Select("tipo_oferta", "id_empresa", "
                        id_rol_empresa", "titulo_oferta", "descripcion", "
                        nota_descripcion", "id_estatus", "fecha_inicio_oferta"
                        , "fecha_fin_oferta", "id_domicilio_origen", "
                        id_servicio").Create(&oferta).Error
233             } else {
234                 return nil, errors.New("Valor no valido para
                        tipo_solicitud")
235             }
236
237             err = tx.Create(&oferta).Error
238
239             return oferta, err
240
241         }
242
243         func (p P1OfertaService) UpdatePOferta(id_oferta int, input *
                model.P1OfertasInput) error {
244             oferta := models.P1_ofertas{}
245             err := p.DB.First(&oferta, id_oferta).Error
246             if err != nil {
247                 return err
248             }
249             oferta.Titulo_oferta = app.ChangeString(&input.TituloOferta,
                    oferta.Titulo_oferta)
250             oferta.Descripcion = null.StringFrom(app.ChangeString(input.
                    Descripcion, oferta.Descripcion.String))
251             oferta.Nota_descripcion = null.StringFrom(app.ChangeString(
                    input.NotaDescripcion, oferta.Nota_descripcion.String))
252             oferta.Id_estatus = null.StringFrom(app.ChangeString(input.
                    IDEstatus, oferta.Id_estatus.String))
253             fechaOfertaInicio := oferta.Fecha_inicio_oferta.String()
254             fechaOfertaFin := oferta.Fecha_fin_oferta.String()
255             oferta.Fecha_inicio_oferta, _ = time.Parse(time.RubyDate, app.
                    ChangeString(input.FechaInicioOferta, fechaOfertaInicio))
256             oferta.Fecha_fin_oferta, _ = time.Parse(time.RubyDate, app.
```

```
                ChangeString(input.FechaFinOferta, fechaOfertaFin))
257
258             oferta.Id_domicilio_origen = null.IntFrom(int64(app.ChangeInt
                    (input.IDDomicilioOrigen, int(oferta.Id_domicilio_origen.
                    Int64))))
259
260             if oferta.Tipo_oferta.Int64 == 1 { // residuo
261                 oferta.Id_residuo_autorizado = null.IntFrom(int64(app.
                        ChangeInt(input.IDResiduoAutorizado, int(oferta.
                        Id_residuo_autorizado.Int64))))
262                 oferta.Residuo_cantidad = null.IntFrom(int64(app.
                        ChangeInt(input.ResiduoCantidad, int(oferta.
                        Residuo_cantidad.Int64))))
263                 oferta.Residuo_unidad = null.IntFrom(int64(app.ChangeInt(
                        input.ResiduoUnidad, int(oferta.Residuo_unidad.Int64))
                        ))
264                 oferta.Ruta_foto01_residuos = null.StringFrom(app.
                        ChangeString(input.RutaFoto01Residuos, oferta.
                        Ruta_foto01_residuos.String))
265                 oferta.Ruta_foto02_residuos = null.StringFrom(app.
                        ChangeString(input.RutaFoto02Residuos, oferta.
                        Ruta_foto02_residuos.String))
266             } else if oferta.Tipo_oferta.Int64 == 2 { // servicio
267                 oferta.Id_servicio = null.IntFrom(int64(app.ChangeInt(
                        input.IDServicio, int(oferta.Id_servicio.Int64))))
268             }
269
270             err = p.DB.Model(&oferta).Where("id_oferta = ?", id_oferta).
                    Updates(oferta).Error
271
272             return err
273
274         }
275
276         func (p P1OfertaService) DeletePOferta(id_oferta int) error {
277             oferta := &models.P1_ofertas{}
278             err := p.DB.Delete(oferta, id_oferta).Error
279             return err
280         }
```

Código A.34: Cuerpo de las consultas del repositorio del catálogo de procesos P1 ofertas.

```
 1          func (r *queryResolver) GetOfertas(ctx context.Context, idOferta
                int) (*model.P1Ofertas, error) {
 2              oferta, err := r.P1Ofertas.GetOferta(idOferta)
```

```
3             if err != nil {
4                 return nil, err
5             }
6             return oferta, err
7         }
8
9         func (r *queryResolver) GetOfertaPorResiduo(ctx context.Context,
               idResiduoAutorizado int) ([]*model.P1Ofertas, error) {
10            oferta, err := r.P1Ofertas.GetOfertaPorResiduo(
                  idResiduoAutorizado)
11            if err != nil {
12                return nil, err
13            }
14            return oferta, err
15        }
16
17        func (r *queryResolver) GetOfertaPorServicio(ctx context.Context,
               idServicio int) ([]*model.P1Ofertas, error) {
18            oferta, err := r.P1Ofertas.GetOfertaPorServicio(idServicio)
19            if err != nil {
20                return nil, err
21            }
22            return oferta, err
23        }
24        func (r *mutationResolver) CreatePOferta(ctx context.Context,
              input model.P1OfertasInput) (int, error) {
25            var idOferta int
26            err := r.DB.Transaction(func(tx *gorm.DB) error {
27                oferta, err := r.P1Ofertas.CreatePOferta(&input, tx)
28                if err != nil {
29                    fmt.Println(err)
30                    return err
31                }
32                idOferta = oferta.Id_oferta
33                return nil
34            })
35            if err != nil {
36                return 0, err
37            }
38            return idOferta, nil
39        }
40
41        func (r *mutationResolver) UpdatePOferta(ctx context.Context,
              idOferta int, input model.P1OfertasInput) (string, error) {
42            err := r.P1Ofertas.UpdatePOferta(idOferta, &input)
```

```
43              if err != nil {
44                  return "ERROR", err
45              }
46
47              return "Actualizacion exitosa", nil
48          }
49
50          func (r *mutationResolver) DeleteP0ferta(ctx context.Context,
                idOferta int) (string, error) {
51              err := r.P1Ofertas.DeleteP0ferta(idOferta)
52              if err != nil {
53                  return "ERROR", err
54              }
55              successMessage := "Borrado exitoso"
56              return successMessage, nil
57          }
```

Código A.35: Resolutores del query.resolvers.go y mutation.resolvers.go para los operadores del catálogo general usuarios tipo.

```
1           package models
2
3           import (
4               "github.com/guregu/null"
5               "time"
6           )
7
8           type P2_ofertas_solicitantes_candidatos struct {
9               Id_ofertas_sol_cand    int        `json:"
                    id_ofertas_solicitante_candidato" gorm:"primaryKey"`
10              Id_oferta              int        `json:"id_oferta"`
11              Id_empresa_solicitante int        `json:"
                    id_empresa_solicitante"`
12              Id_empresa_ofertante   int        `json:"id_empresa_ofertante"
                    `
13              Fecha_oferta_solicitud time.Time `json:"
                    fecha_oferta_solicitud"`
14              Id_estatus             null.String `json:"id_estatus"`
15              Notas_observaciones    null.String `json:"notas_observaciones
                    "`
16              Ctl_es_registro_activo bool       `json:"
                    ctl_es_registro_activo"`
17              Ctl_fecha_registro     time.Time  `json:"ctl_fecha_registro"`
18              Ctl_usuario_registro   null.String `json:"
                    ctl_usuario_registro"`
```

```
19            Ctl_fecha_modificacion time.Time `json:"
                  ctl_fecha_modificacion"`
20            Ctl_usuario_modificacion int      `json:"
                  ctl_usuario_modificacion"`
21        }
```

Código A.36: Modelo para el catálogo de procesos P2 ofertas solicitantes candidatos.

```
1         # p2_ofertas_solicitantes_candidatos
2         type P2_ofertas_solicitantes_candidato{
3            id_oferta_solicitante_candidato: ID!
4            oferta: P1_ofertas
5            empresa_solicitante: Cpi_empresas
6            empresa_ofertante: Cpi_empresas
7            fecha_oferta_solicitud: Timestamp
8            estatus: Cpr_estatus_proceso
9            notas_observaciones: String
10           ctl_es_registro_activo: Boolean!
11           ctl_fecha_registro: Timestamp!
12           ctl_usuario_registro: String
13           ctl_fecha_modificacion: Timestamp!
14           ctl_usuario_modificacion: Int
15        }
```

Código A.37: Prototipo del schema.graphqls del catálogo de procesos P2 ofertas solicitantes candidatos.

```
1         GetOfertaSolicitudesCandidatos(id_ofertas_solicitante_candidato:
              ID!): P2_ofertas_solicitantes_candidato!
2         GetOfertaSolicitudesCandidatosPorOferta(id_oferta: ID!): [
              P2_ofertas_solicitantes_candidato!]!
3         CreateOfertaSolicitantesCandidatos(input:
              P2OfertasSolicitantesCandidatosInput!): ID!
4         UpdateOfertaSolicitudCandidato(id_ofertas_solicitante_candidato:
              ID!, input: P2OfertasSolicitantesCandidatosInput!): String!
5         DeleteOfertaSolicitanteCandidato(
              id_ofertas_solicitante_candidato: ID!): String!
```

Código A.38: Firmas de tipo query del query.graphqls y mutation.graphqls del catálogo de procesos P2 ofertas solicitantes candidatos.

```
1         package repository
2
3         import (
4            "apiCatalogosGenerales/app"
5            "apiCatalogosGenerales/app/models"
```

```
6           "apiCatalogosGenerales/graph/model"
7           "errors"
8           "github.com/guregu/null"
9           "gorm.io/gorm"
10          "time"
11      )
12
13      type P2OfertasSolicitantesCandidatosRepository interface {
14          CreateOfertaaSolicitantesCandidatos(input *model.
                P2OfertasSolicitantesCandidatosInput, tx *gorm.DB) (*
                models.P2_ofertas_solicitantes_candidatos, error)
15          GetOfertaSolicitudesCandidatos(
                id_ofertas_solicitante_candidato int) (*model.
                P2OfertasSolicitantesCandidato, error)
16          GetOfertaSolicitudesCandidatosPorOferta(id_oferta int) ([]*
                model.P2OfertasSolicitantesCandidato, error)
17          UpdateOfertaSolicitudCandidato(
                id_ofertas_solicitante_candidato int, input *model.
                P2OfertasSolicitantesCandidatosInput) error
18          DeleteOfertaSolicitanteCandidato(
                id_ofertas_solicitante_candidato int) error
19      }
20
21      type P2OfertasSolicitantesCandidatosRepositoryService struct {
22          DB *gorm.DB
23      }
24
25      func (p P2OfertasSolicitantesCandidatosRepositoryService)
            CreateOfertaaSolicitantesCandidatos(input *model.
            P2OfertasSolicitantesCandidatosInput, tx *gorm.DB) (*models.
            P2_ofertas_solicitantes_candidatos, error) {
26          ofertas := &models.P2_ofertas_solicitantes_candidatos{
27              Id_oferta:            input.IDOferta,
28              Id_empresa_solicitante: input.IDEmpresaSolicitante,
29              Id_empresa_ofertante: input.IDEmpresaOfertante,
30              Fecha_oferta_solicitud: time.Now(),
31              Id_estatus:           null.StringFrom(input.IDEstatus),
32              Notas_observaciones:  null.StringFromPtr(input.
                    NotasObservaciones),
33          }
34
35          err := tx.Select("id_oferta", "id_empresa_solicitante", "
                id_empresa_ofertante", "fecha_oferta_solicitud", "
                id_estatus", "notas_observaciones").Create(&ofertas).
                Error
```

```
36
37              return ofertas, err
38          }
39
40          func (p P2OfertasSolicitantesCandidatosRepositoryService)
                GetOfertaSolicitudesCandidatos(
                id_ofertas_solicitante_candidato int) (*model.
                P2OfertasSolicitantesCandidato, error) {
41              ofertaCandidato := models.P2_ofertas_solicitantes_candidatos
                    {}
42              err := p.DB.Where("id_ofertas_solicitante_candidato = ?",
                    id_ofertas_solicitante_candidato).First(&ofertaCandidato).
                    Error
43              if err != nil {
44                  return nil, errors.New("No existe el candidato de
                        solicitud de oferta. " + err.Error())
45              }
46              //Generamos la busqueda para la oferta deseada por el su id
47              ofertaService := P1OfertaService{DB: p.DB}
48              oferta, err := ofertaService.GetOferta(ofertaCandidato.
                    Id_oferta)
49              if err != nil {
50                  return nil, err
51              }
52              //Generamos la busqueda para la empresa solicitante por su id
53              empresaService := EmpresaService{DB: p.DB}
54              empresaSolicitante, err := empresaService.GetEmpresa(
                    ofertaCandidato.Id_empresa_solicitante, "")
55              if err != nil {
56                  return nil, err
57              }
58              empresaOfertante, err := empresaService.GetEmpresa(
                    ofertaCandidato.Id_empresa_ofertante, "")
59              if err != nil {
60                  return nil, err
61              }
62              //Generamos la busqueda para del Estatus por su id
63              estatusService := EstatusProcesoService{DB: p.DB}
64              estatus, err := estatusService.GetEstatusProceso(
                    ofertaCandidato.Id_estatus.String)
65              if err != nil {
66                  return nil, err
67              }
68              fechaOfertaSolicitud := ofertaCandidato.
                    Fecha_oferta_solicitud.String()
```

```
69
70          ofertaQ := &model.P2OfertasSolicitantesCandidato{
71              IDOfertaSolicitanteCandidato: ofertaCandidato.
                    Id_ofertas_sol_cand,
72              Oferta:                     oferta,
73              EmpresaSolicitante:         empresaSolicitante,
74              EmpresaOfertante:           empresaOfertante,
75              FechaOfertaSolicitud:       &fechaOfertaSolicitud,
76              Estatus:                    estatus,
77              NotasObservaciones:         &ofertaCandidato.
                    Notas_observaciones.String,
78              CtlEsRegistroActivo:        ofertaCandidato.
                    Ctl_es_registro_activo,
79              CtlFechaRegistro:           ofertaCandidato.
                    Ctl_fecha_registro.String(),
80              CtlUsuarioRegistro:         &ofertaCandidato.
                    Ctl_usuario_registro.String,
81              CtlFechaModificacion:       oferta.CtlFechaModificacion,
82              CtlUsuarioModificacion:     &ofertaCandidato.
                    Ctl_usuario_modificacion,
83          }
84
85          return ofertaQ, err
86
87      }
88
89      func (p P2OfertasSolicitantesCandidatosRepositoryService)
            GetOfertaSolicitudesCandidatosPorOferta(id_oferta int) ([]*
            model.P2OfertasSolicitantesCandidato, error) {
90          ofertaBusqueda := []*models.
                P2_ofertas_solicitantes_candidatos{}
91          err := p.DB.Where("id_oferta = ?", id_oferta).First(&
                ofertaBusqueda).Error
92          if err != nil {
93              return nil, errors.New("Error en la busqueda de oferta." +
                     err.Error())
94          }
95          //Declaracion de los servicios para oferta, empresa y estatus
96          ofertaService := P1OfertaService{DB: p.DB}
97          empresaService := EmpresaService{DB: p.DB}
98          estatusService := EstatusProcesoService{DB: p.DB}
99
100         ofertas := []*model.P2OfertasSolicitantesCandidato{}
101
102         for i := 0; i < len(ofertaBusqueda); i++ {
```

```
103             oferta, err := ofertaService.GetOferta(ofertaBusqueda[i].
                    Id_oferta)
104             if err != nil {
105                 return nil, err
106             }
107             empresaSolicitante, err := empresaService.GetEmpresa(
                    ofertaBusqueda[i].Id_empresa_solicitante, "")
108             if err != nil {
109                 return nil, err
110             }
111             empresaOfertante, err := empresaService.GetEmpresa(
                    ofertaBusqueda[i].Id_empresa_ofertante, "")
112             if err != nil {
113                 return nil, err
114             }
115             estatus, err := estatusService.GetEstatusProceso(
                    ofertaBusqueda[i].Id_estatus.String)
116             if err != nil {
117                 return nil, err
118             }
119             fechaOfertaSolicitud := ofertaBusqueda[i].
                    Fecha_oferta_solicitud.String()
120
121             ofertaQ := &model.P2OfertasSolicitantesCandidato{
122                 IDOfertaSolicitanteCandidato: ofertaBusqueda[i].
                        Id_ofertas_sol_cand,
123                 Oferta:                    oferta,
124                 EmpresaOfertante:          empresaOfertante,
125                 EmpresaSolicitante:        empresaSolicitante,
126                 FechaOfertaSolicitud:      &fechaOfertaSolicitud,
127                 Estatus:                   estatus,
128                 NotasObservaciones:        &ofertaBusqueda[i].
                        Notas_observaciones.String,
129                 CtlEsRegistroActivo:       ofertaBusqueda[i].
                        Ctl_es_registro_activo,
130                 CtlFechaRegistro:          ofertaBusqueda[i].
                        Ctl_fecha_registro.String(),
131                 CtlUsuarioRegistro:        &ofertaBusqueda[i].
                        Ctl_usuario_registro.String,
132                 CtlFechaModificacion:      ofertaBusqueda[i].
                        Ctl_fecha_modificacion.String(),
133                 CtlUsuarioModificacion:    &ofertaBusqueda[i].
                        Ctl_usuario_modificacion,
134             }
135             ofertas = append(ofertas, ofertaQ)
```

```
136             }
137
138             return ofertas, err
139         }
140
141         func (p P2OfertasSolicitantesCandidatosRepositoryService)
                UpdateOfertaSolicitudCandidato(
                id_ofertas_solicitante_candidato int, input *model.
                P2OfertasSolicitantesCandidatosInput) error {
142             ofertas := models.P2_ofertas_solicitantes_candidatos{}
143             err := p.DB.First(&ofertas, id_ofertas_solicitante_candidato).
                    Error
144             if err != nil {
145                 return errors.New("Error en la busqueda de la oferta")
146             }
147             ofertas.Id_estatus = null.StringFrom(app.ChangeString(&input.
                    IDEstatus, ofertas.Id_estatus.String))
148             ofertas.Notas_observaciones = null.StringFrom(app.
                    ChangeString(input.NotasObservaciones, ofertas.
                    Notas_observaciones.String))
149             err = p.DB.Model(&ofertas).Where("
                    id_ofertas_solicitante_candidato = ?",
                    id_ofertas_solicitante_candidato).Updates(ofertas).Error
150             if err != nil {
151                 return errors.New("Error en la actualizacion. " + err.
                        Error())
152             }
153             return err
154         }
155
156         func (p P2OfertasSolicitantesCandidatosRepositoryService)
                DeleteOfertaSolicitanteCandidato(
                id_ofertas_solicitante_candidato int) error {
157             oferta := &models.P2_ofertas_solicitantes_candidatos{}
158             err := p.DB.Delete(oferta, id_ofertas_solicitante_candidato).
                    Error
159             return err
160         }
161
162         func NewP2OfertasSolicitantesCandidatosRepositoryService(db *
                gorm.DB) *P2OfertasSolicitantesCandidatosRepositoryService {
163             return &P2OfertasSolicitantesCandidatosRepositoryService{DB:
                    db}
164         }
165
```

```
166         var _ P2OfertasSolicitantesCandidatosRepository = &
                P2OfertasSolicitantesCandidatosRepositoryService{}
```

Código A.39: Cuerpo de las consultas del repositorio del catálogo de procesos P2 ofertas solicitantes candidatos.

```
1           func (r *queryResolver) GetOfertaSolicitudesCandidatos(ctx
                context.Context, idOfertasSolicitanteCandidato int) (*model.
                P2OfertasSolicitantesCandidato, error) {
2               oferta, err := r.P2OfertasSolicitantes.
                    GetOfertaSolicitudesCandidatos(
                    idOfertasSolicitanteCandidato)
3               if err != nil {
4                   return nil, err
5               }
6               return oferta, err
7           }
8
9           func (r *queryResolver) GetOfertaSolicitudesCandidatosPorOferta(
                ctx context.Context, idOferta int) ([]*model.
                P2OfertasSolicitantesCandidato, error) {
10              oferta, err := r.P2OfertasSolicitantes.
                    GetOfertaSolicitudesCandidatosPorOferta(idOferta)
11              if err != nil {
12                  return nil, err
13              }
14              return oferta, err
15          }
16          func (r *mutationResolver) CreateOfertaSolicitantesCandidatos(ctx
                 context.Context, input model.
                P2OfertasSolicitantesCandidatosInput) (int, error) {
17              var id int
18              err := r.DB.Transaction(func(tx *gorm.DB) error {
19                  oferta, err := r.P2OfertasSolicitantes.
                        CreateOfertaaSolicitantesCandidatos(&input, tx)
20                  if err != nil {
21                      return err
22                  }
23                  id = oferta.Id_ofertas_sol_cand
24                  return nil
25              })
26              if err != nil {
27                  return 0, err
28              }
29              return id, err
```

```
30          }
31
32          func (r *mutationResolver) UpdateOfertaSolicitudCandidato(ctx
                context.Context, idOfertasSolicitanteCandidato int, input
                model.P2OfertasSolicitantesCandidatosInput) (string, error) {
33              err := r.P2OfertasSolicitantes.UpdateOfertaSolicitudCandidato
                    (idOfertasSolicitanteCandidato, &input)
34              if err != nil {
35                  return "ERROR", err
36              }
37
38              return "Actualizacion exitosa", nil
39          }
40
41          func (r *mutationResolver) DeleteOfertaSolicitanteCandidato(ctx
                context.Context, idOfertasSolicitanteCandidato int) (string,
                error) {
42              err := r.P2OfertasSolicitantes.
                    DeleteOfertaSolicitanteCandidato(
                    idOfertasSolicitanteCandidato)
43              if err != nil {
44                  return "ERROR", err
45              }
46              return "Borrado exitoso", nil
47          }
```

Código A.40: Resolutores del query.resolvers.go y mutation.resolvers.go para los operadores del catálogo de procesos P2 ofertas solicitantes candidatos.

B

Pruebas de funcionalidad en Postman

Índice

B.1. Pruebas de los catálogos generales

`cg_estados` (GetEstado y GetAllEstados): APIs utilizadas para las consultas de los estados. Se devuelven los estados existentes en el catálogo o un estado específico filtrado por el campo de id_estado. GetEstado devuelve el registrado del catálogo por su identificador (Figura B.1), y la segunda GetAllEstados devuelve todos los registros existentes en el catálogo (Figura B.2).

QUERY

```
query(
    $id_estado: ID,
    $nombre_estado: String,
    $clave_estado: String
){
    GetEstado(
        id_estado: $id_estado,
        nombre_estado: $nombre_estado,
        clave_estado: $clave_estado
    ){
        id_estado
        nombre_estado
        clave_estado
    }
}
```

GRAPHQL VARIABLES

```
{
  "id_estado": 15
}
```

```
{
  "data": {
    "GetEstado": [
      {
        "id_estado": 15,
        "nombre_estado": "Estado de México",
        "clave_estado": "EDOMEX"
      }
    ]
  }
}
```

Figura B.1: API para la consulta de un estado específico.

QUERY

```
query{
    GetAllEstados{
        id_estado
        nombre_estado
        clave_estado
    }
}
```

GRAPHQL VARIABLES

```

```

```
{
  "data": {
    "GetAllEstados": [
      {
        "id_estado": 1,
        "nombre_estado": "Aguascalientes",
        "clave_estado": "AGS"
      },
      {
        "id_estado": 2,
        "nombre_estado": "Baja California",
        "clave_estado": "BC"
      },
      {
        "id_estado": 3,
        "nombre_estado": "Baja California Sur",
        "clave_estado": "BCS"
      },
      {
        "id_estado": 4,
        "nombre_estado": "Campeche",
        "clave_estado": "CAMP"
      },
```

Figura B.2: API para la consulta de los estados existentes en el catálogo.

`cg_calles` (GetCalle, CreateCalle, UpdateCalle y DeleteCalle): APIs utilizadas para realizar las operaciones CRUD del catálogo general Calles. En la Figura B.3 se crea una calle con el nombre de Avenida Universidad. En la Figura B.4 se realiza la consulta de una calle filtrada por su identificador. En la Figura B.5 se realiza la actualización del campo `nombre_corto` asignándole 'Universidad', y en la Figura B.6 se realiza la eliminación del registro con el identificador tres.

Figura B.3: API para la creación de un registro en el catálogo general calles.

Figura B.4: API para la consulta de un registro en el catálogo general calles.

Figura B.5: API para la actualización de un registro en el catálogo general calles.

Figura B.6: API para la eliminación de un registro en el catálogo general calles.

`cg_codigos_postales` (GetCodigoPostal): API utilizada para la consulta de la información de un código postal, donde através de su ejecución figura B.7 se obtiene la información del código postal 51901 que pertenece al Estado de México que delimita los municipios de Ixtapan de la Sal y Coatepec Harinas.

```
query($id_cp: ID!){
    GetCodigoPostal(id_cp: $id_cp){
        id_cp
        estado{
            id_estado
            nombre_estado
            clave_estado
        }
        alcaldia_municipio{
            id_alcaldia_municipio
            nombre
            nombre_corto
            nombre_coloquial
        }
    }
}
```

```
VARIABLES
{
  "id_cp": 51901
}
```

```
{
  "data": {
    "GetCodigoPostal": [
      {
        "id_cp": 51901,
        "estado": {
          "id_estado": 15,
          "nombre_estado": "Estado de México",
          "clave_estado": "EDOMEX"
        },
        "alcaldia_municipio": {
          "id_alcaldia_municipio": 150040,
          "nombre": "Ixtapan de la Sal",
          "nombre_corto": null,
          "nombre_coloquial": null
        }
      },
      {
        "id_cp": 51901,
        "estado": {
          "id_estado": 15,
          "nombre_estado": "Estado de México",
          "clave_estado": "EDOMEX"
        },
        "alcaldia_municipio": {
          "id_alcaldia_municipio": 150021,
          "nombre": "Coatepec Harinas",
          "nombre_corto": null,
          "nombre_coloquial": null
        }
      }
    ]
  }
}
```

Figura B.7: API para obtener la información de un registro del catálogo general códigos postales con el identificador 51901.

B.2. Pruebas de los catálogos de procesos

`cpr_roles` (GetAllRoles y GetRol): APIs utilizadas para la consulta y descripción de los tipos de roles. GetAllRoles obtiene todos los roles disponibles en el catálogo de procesos de roles, los cuales se encuentran asociados a una determinada asignación,

esto se muestra en la figura B.8. Mientras que GetRol obtiene un rol específico filtrado por el identificador o por el nombre, como se muestra en la figura B.9, donde se filtró por el nombre 'Reciclaje'.

QUERY

```
query{
    GetAllRoles{
        id_rol
        nombre_rol
        descripcion_rol
        ctl_es_registro_activo
        ctl_fecha_registro
        ctl_fecha_modificacion
        ctl_fecha_modificacion
        ctl_usuario_modificacion
    }
}
```

Pretty Raw Preview Visualize JSON


```
{
    "data": {
        "GetAllRoles": [
            {
                "id_rol": 1,
                "nombre_rol": "Acopio y almacenamiento",
                "descripcion_rol": "rol para acopio y almacenamiento de residuos",
                "ctl_es_registro_activo": true,
                "ctl_fecha_registro": "2022-06-04T19:08:03.151314Z",
                "ctl_fecha_modificacion": null,
                "ctl_usuario_modificacion": null
            },
            {
                "id_rol": 2,
                "nombre_rol": "Recolección y transporte",
                "descripcion_rol": "rol para actividades de recolección y transporte",
                "ctl_es_registro_activo": true,
                "ctl_fecha_registro": "2022-06-04T19:08:03.151314Z",
                "ctl_fecha_modificacion": null,
                "ctl_usuario_modificacion": null
            },
            {
                "id_rol": 3,
                "nombre_rol": "Reciclaje",
                "descripcion_rol": "rol para actividades de reciclaje de residuos",
```


Figura B.8: API para la consulta de los tipos de roles en el catálogo de procesos de roles.

QUERY

```
query(
    $id_rol: ID,
    $nombre: String
){
    GetRol(
        id_rol: $id_rol,
        nombre: $nombre
    ){
        id_rol
        nombre_rol
        descripcion_rol
        ctl_es_registro_activo
        ctl_fecha_registro
        ctl_fecha_modificacion
        ctl_fecha_modificacion
        ctl_usuario_modificacion
    }
}
```

Pretty Raw Preview Visualize JSON


```
{
    "data": {
        "GetRol": {
            "id_rol": 3,
            "nombre_rol": "Reciclaje",
            "descripcion_rol": "rol para actividades de reciclaje de residuos",
            "ctl_es_registro_activo": true,
            "ctl_fecha_registro": "2022-06-04T19:08:03.151314Z",
            "ctl_fecha_modificacion": null,
            "ctl_usuario_modificacion": null
        }
    }
}
```


Figura B.9: API para la consulta de un tipo de rol específico en el catálogo de procesos de roles.

`cpr_estatus_procesos` (GetEstatusProceso): API utilizada para la consulta de la situación de los procesos, es decir, obtener la información del catálogo de procesos estatus procesos filtrando por el identificador, en la figura B.10 se filtró por el identificador 'id123'.

Figura B.10: API para la consulta de un determinado proceso.

`cpi_empresas_identificadores` (GetEmpresaIdentificador, GetEmpresaIdentificadoresPorEmpresa, CreateEmpresaIdentificador, UpdateEmpresaIdentificador y DeleteEmpresaIdentificador): APIS utilizadas para las operaciones CRUD de los registros del catálogo de procesos de empresas identificadores. GetEmpresaIdentificador devuelve un registro específico filtrado por su identificador figura B.11. GetEmpresaIdentificadoresPorEmpresa devuelve un registro específico filtrado por el identificador asociado a la empresa figura B.12. CreateEmpresaIdentificador crea un registro nuevo en el catálogo figura B.13. UpdateEmpresaIdentificador actualiza un registro específico filtrado por su identificador figura B.14. DeleteEmpresaIdentificador borra un registro específico del catálogo por su identificador figura B.15.

QUERY

```
query(
    $id_empresa_identificador: ID!
){
    GetEmpresaIdentificador(
        id_empresa_identificador: $id_empresa_identificador
    )
    {
        id_empresa_identificador
        id_empresa
        texto_identificador
        tipo_identificador
        otro_tipo_identificador
    }
}
```

GRAPHQL VARIABLES

```
--variables
{
    "id_empresa_identificador": 4
}
```

Figura B.11: API utilizada para obtener la información de un registro específico del catálogo de procesos empresas identificadores.

QUERY

```
query(
    $id_empresa: ID
){
    GetEmpresaIdentificadoresPorEmpresa(
        id_empresa: $id_empresa
    ){
        id_empresa_identificador
        id_empresa
        texto_identificador
        tipo_identificador
        otro_tipo_identificador
    }
}
```

GRAPHQL VARIABLES

```
{
    "id_empresa": 2
}
```

Figura B.12: API utilizada para obtener la información de un registro específico del catálogo de procesos empresas identificadores por el identificador de la empresa asociada.

QUERY

```
mutation(
    $id_empresa: ID!
    $texto_identificador: String!
    $tipo_identificador: String!
    $otro_tipo_identificador: String
){
    CreateEmpresaIdentificador(
        input:{
            id_empresa: $id_empresa,
            texto_identificador: $texto_identificador,
            tipo_identificador: $tipo_identificador,
            otro_tipo_identificador: $otro_tipo_identificador
        }
    ){
        id_empresa_identificador
        id_empresa
        texto_identificador
        tipo_identificador
        otro_tipo_identificador
    }
}
```

GRAPHQL VARIABLES

```
{
    "id_empresa": 2,
    "texto_identificador": "Texto1",
    "tipo_identificador": "Texto2",
    "otro_tipo_identificador": "Texto3"
}
```

Figura B.13: API utilizada para la creación de un registro en el catálogo de procesos de empresas identificadores.

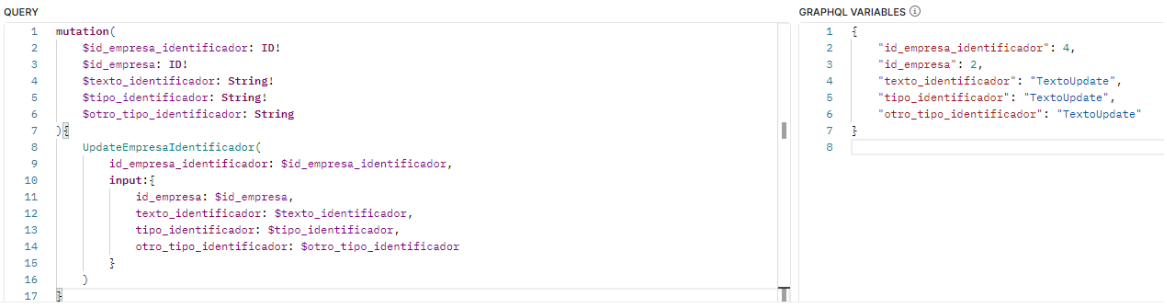

Figura B.14: API utilizada para la actualización de un registro por su identificador en el catálogo de procesos de empresas identificadores.

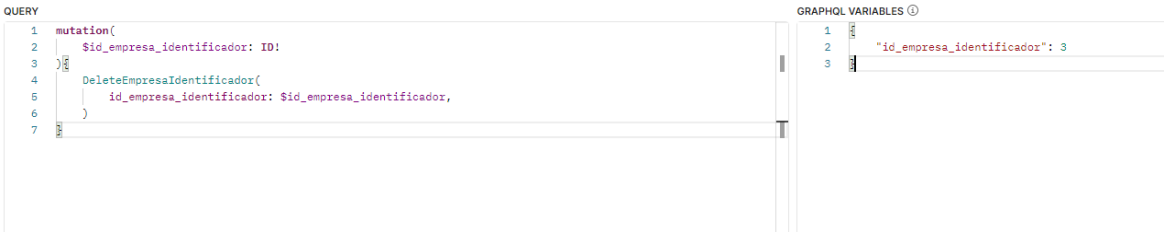

Figura B.15: API utilizada para el borrado de un registro específico por su identificador del catálogo de procesos de empresas identificadores.

`p1_ofertas` (GetOfertas, GetOfertaPorResiduo, GetOfertaPorServicio, CreatePOferta, UpdatePOferta y DeletePOferta): APIs utilizadas para las operaciones CRUD del catálogo de procesos de p1 ofertas. GetOferta obtiene la información de un registro específico por su identificador, figura B.16. GetOfertaPorResiduo obtiene la información de los registros por el identificador de un residuo específico, figura B.17. GetOfertaPorServicio obtiene la información de los registros por el servicio específico, figura B.18. CreatePOferta crea un registro en el catálogo figura B.19. UpdatePOferta actualiza la información de un registro específico, figura B.20. DeletePOferta borra un registro específico del catálogo por su identificador, figura B.21.

Figura B.16: API utilizada para la obtención de la información de un registro específico del catálogo de procesos p1 ofertas.

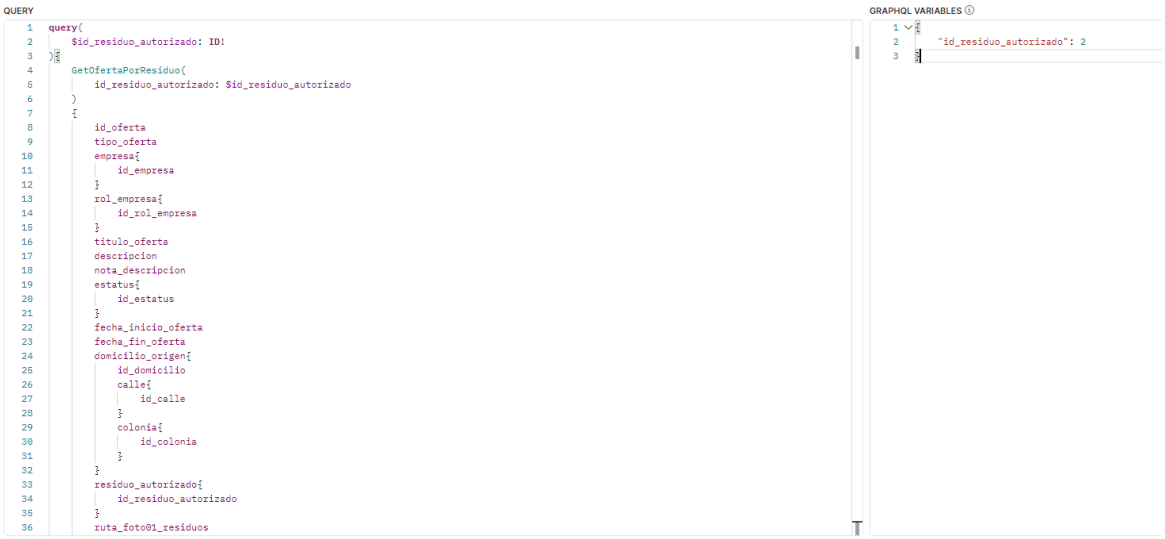

Figura B.17: API utilizada para la obtención de los registros de las ofertas por el identificador de un residuo específico del catálogo de procesos de p1 ofertas.

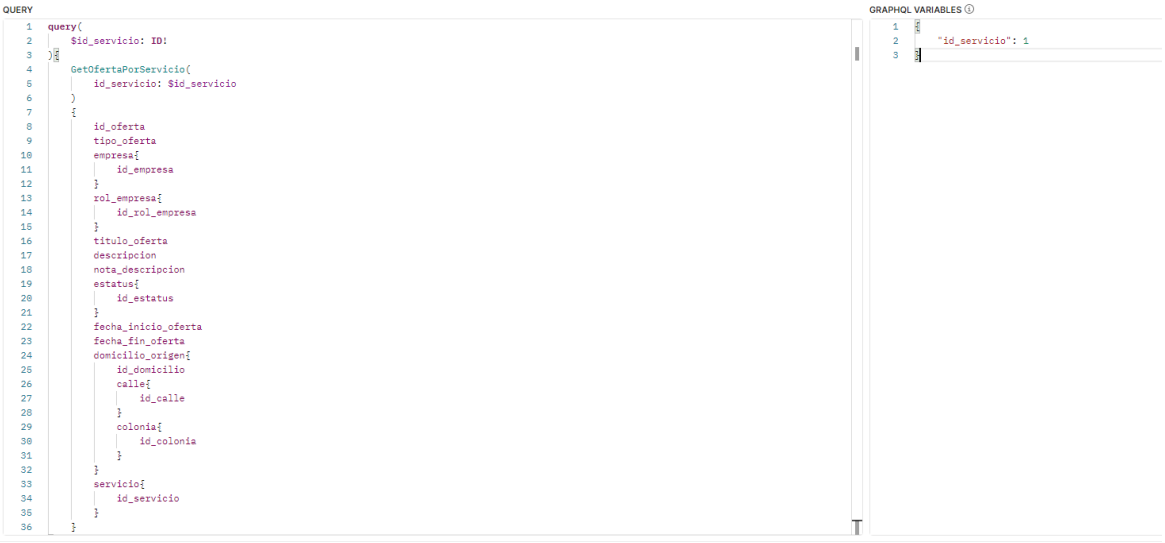

Figura B.18: API utilizada para obtener los registros de las ofertas por el identificador de un servicio específico del catálogo de procesos de p1 ofertas.

Figura B.19: API utilizada para la creación de un registro en el catálogo de procesos de p1 ofertas.

Figura B.20: API utilizada para la actualización de un registro específico del catálogo de procesos de p1 ofertas.

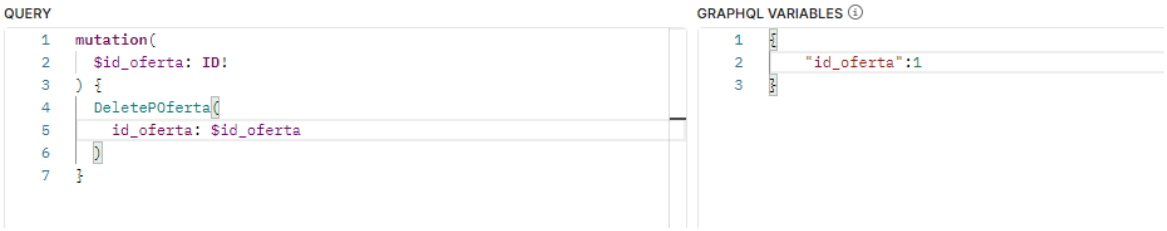

Figura B.21: API utilizada para borrar un registro específico del catálogo de procesos de p1 ofertas.

`p2_ofertas_solicitantes_candidatos` (GetOfertaSolicitudesCandidatos , GetOfertaSolicitudesCandidatosPorOferta, CreateOfertaSolicitantesCandidatos, UpdateOfertaSolicitudCandidato, y DeleteOfertaSolicitanteCandidato): APIs utilizadas para las operaciones CRUD del catálogo de procesos de p2 ofertas solicitantes candidatos. GetOfertaSolicitudesCandidatos se utiliza para obtener un registro específico del catálogo filtrado por su identificador, figura B.23. GetOfertaSolicitudesCandidatosPorOferta se utiliza para obtener un registro específico por el identificador de la oferta, figura B.24. CreateOfertaSolicitantesCandidatos crea un registro en el catálogo, figura B.22. UpdateOfertaSolicitudCandidato actualiza los datos de un registro por su identificador, figura B.25. DeleteOfertaSolicitanteCandidato elimina un registro del catálogo por su identificador, figura B.26.

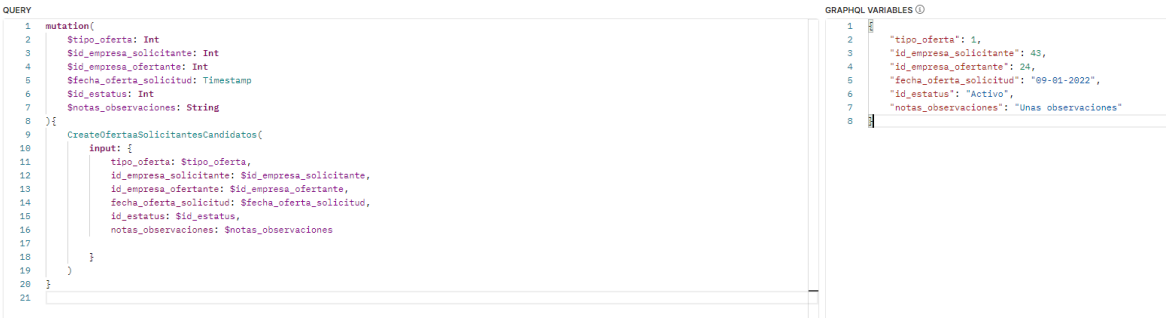

Figura B.22: API utilizada para la creación de un registro para el catálogo de procesos p2 oferta solicitantes candidatos.

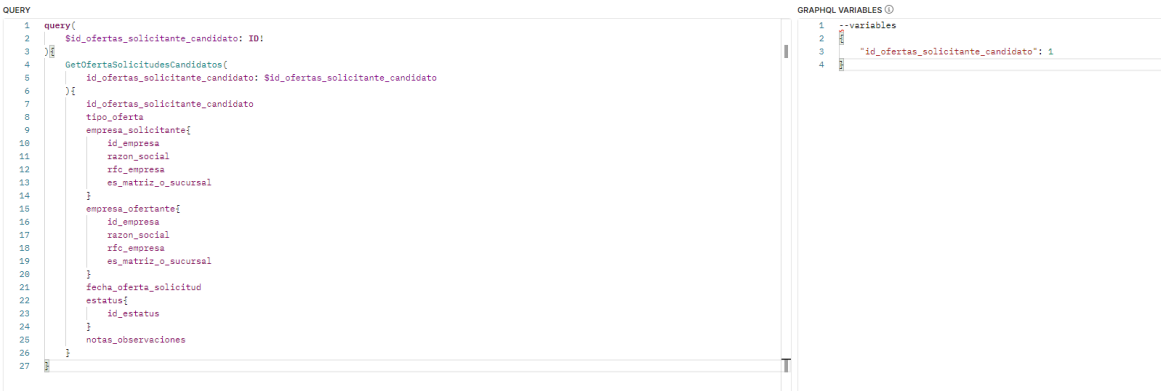

Figura B.23: API utilizada para obtener el registro filtrado por su identificador del catálogo de procesos p2 oferta solicitantes candidatos.

Figura B.24: API utilizada para obtener un registro específico filtrado por el identificador de la oferta del catálogo de procesos p2 oferta solicitantes candidatos.

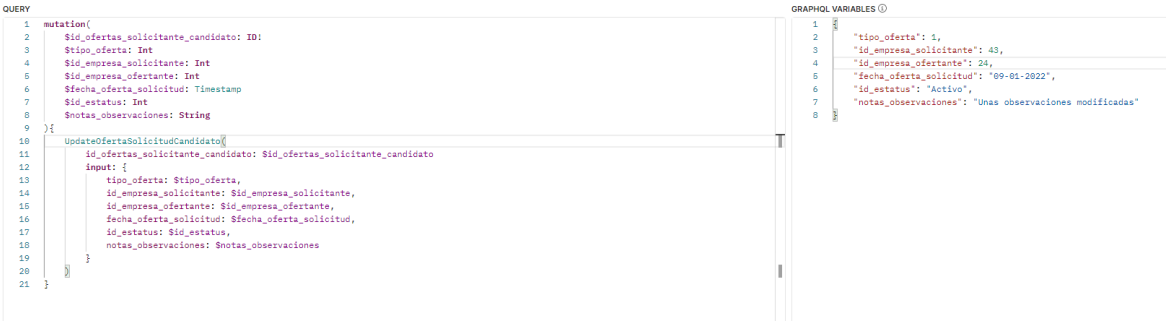

Figura B.25: API utilizada para la actualización de un registro por su identificador del catálogo de procesos p2 oferta solicitantes candidatos.

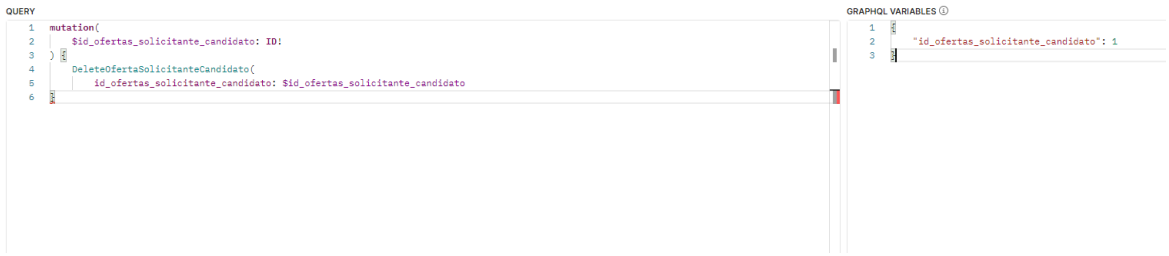

Figura B.26: API utilizada para la eliminación de un registro por su identificador del catálogo de procesos p2 oferta solicitantes candidatos.

C

Reporte 2: Implementación de un servidor GraphQL

Índice

C.1. Implementación de un servidor GraphQL con Go

El propósito de **graphql-go** es proveer un soporte en la espicificación de GraphQL con un conjunto idiomático, fácil de usar para los paquetes de Go (graph-gophers, 2022 (En línea)).

- API mínima.
- Soporte para estándares del tipo contex.Contex.
- Soporte para el estándar OpenTracing.
- Esquema de revisión de tipo frente a solucionadores.

- Los solucionadores están emparejados con el esquema basado en conjuntos de métodos (pueden resolver un esquema de GraphQL con una interface o estructura de Go).
- Manejador de pánico en los resolutores.
- Ejecución paralela de los resolutores.
- Suscripciones.

C.2. Código

```
1   package main
2
3   import (
4       "log"
5       "net/http"
6
7       "github.com/graph-gophers/graphql-go"
8       "github.com/graph-gophers/graphql-go/relay"
9   )
10
11  type query struct{}
12
13  func (_ *query) Hello() string { return "Hello, world!" }
14
15  func main() {
16      s := `
17                  type Query {
18                         hello: String!
19                  }
20          `
21      schema := graphql.MustParseSchema(s, &query{})
22      http.Handle("/query", &relay.Handler{Schema: schema})
23      log.Fatal(http.ListenAndServe(":8080", nil))
24  }
```

Código C.1: Implementación de un servidor local con GraphQL.

Al ejecutar el servidor local del código C.1 se obtiene la respuesta de “Hello, World!” que se muestra en la Figura C.1.

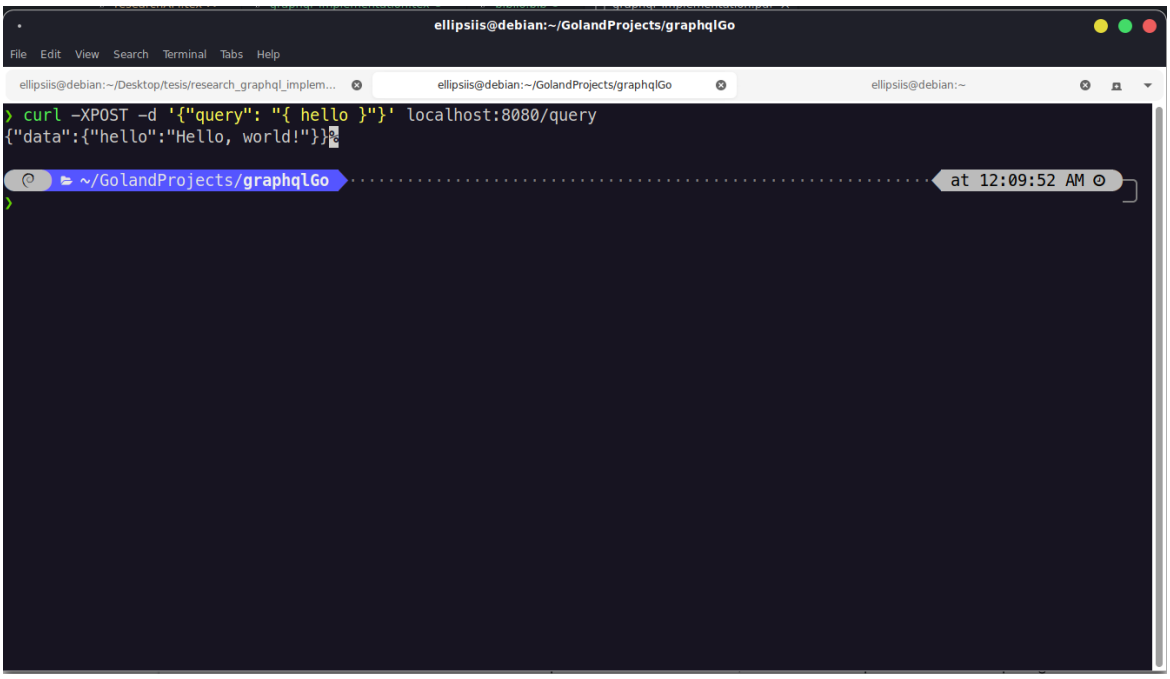

Figura C.1: Ejecución de consulta en el servidor local.

Referencias bibliográficas

Al-Debagy, O., & Martinek, P. (2020). Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach. *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, 289-294. https://doi.org/10.1109/SoSE50414.2020.9130466

Benslimane, D., Dustdar, S., & Sheth, A. (2008). Services Mashups: The New Generation of Web Applications. *IEEE Internet Computing*, *12*(5), 13-15. https://doi.org/10.1109/MIC.2008.110

Bouchiha, D., & Malki, M. (2010). Towards re-engineering Web applications into Semantic Web Services. *2010 International Conference on Machine and Web Intelligence*, 350-353. https://doi.org/10.1109/ICMWI.2010.5648057

Buckley, F. J., & Poston, R. (1984). Software Quality Assurance. *IEEE Transactions on Software Engineering*, *SE-10*(1), 36-41. https://doi.org/10.1109/TSE.1984.5010196

Buenrostro, O., Bocco, G., & Bernache, G. (2001). Urban solid waste generation and disposal in Mexico: a case study. *Waste Management & Research*, *19*(2), 169-176. https://doi.org/10.1177/0734242X0101900208

Chemuturi, M. (2011). Quality Assurance Basics. En *Software Quality Assurance: Best Practices, Tools and Techniques for Software Developers* (1.ª ed., pp. 1-23). J. Ross.

DOF. (2003). Ley general para la prevención y gestión integral de los residuos [DOF 22-05-2015]. *Secretaria de servicios parlamentarios*.

Doxsey, C. (2016). Getting Started. En *Introducing Go: Build Reliable, Scalable Programs* (1.ª ed.). 0'Reilly Media, Inc.

Galin, D. (2004). What is software quality? En *Software Quality Assurance* (1.ª ed., pp. 14-33). Pearson Education Limited.

Gomaa, H. (2011). Designing Client/Server Software Architectures. En *Software Modeling and Design* (pp. 253-276).

graph-gophers. (2022 (En línea)). *Getting started*. GitHub. https://github.com/graph-gophers/graphql-go

Gutiérrez Galicia, F., Coria Páez, A. L., & Tejeida Padilla, R. (2019). A Study and Factor Identification of Municipal Solid Waste Management in Mexico City. *Sustainability*, *11*(22). https://doi.org/10.3390/su11226305

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, 1-84. https://doi.org/10.1109/IEEESTD.1990.101064

ISO. (2015). *Quality management systems – Fundamentals and vocabulary* (Standard). International Organization for Standardization. Geneva, Switzerland. https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en

Jazayeri, M. (2007). Some Trends in Web Application Development. *Future of Software Engineering (FOSE '07)*, 199-213. https://doi.org/10.1109/FOSE.2007.26

Kumar, A. (2014). Human Activities and their Impact on Environment. En *Enviroment and Ecology* (pp. 12-13). New Age International Limited.

Matthew, N., & Stones, R. (2005). Introduction to PostgresSQL. En *Beginning Databases with PostgresSQL: From Novice to Professional* (2.ª ed., pp. 1-17). Apress.

McLellan, R., Iyengar, L., Jeffries, B., & Oerlemans, N. (2014). *Living Planet Report 2014: Species and spaces, people and places* [Accesado en marzo 13, 2022]. WWF International.

Moreno, A. D., Rodríguez, M. G., Velasco, A. R., Enríquez, J. C. M., Lara, R. G., Gutiérrez, A. M., & Hernández, N. A. D. (2012). Mexico City's Municipal Solid Waste Characteristics And Composition Analysis. http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S0188-49992013000100004&lng=es&nrm=iso

Muñoz-Cadena, C., Arenas-Huertero, F., & Ramón-Gallegos, E. (2009). Comparative analysis of the street generation of inorganic urban solid waste (IUSW) in two neighborhoods of Mexico City. *Waste Management*, *29*(3), 1167-1175. https://doi.org/https://doi.org/10.1016/j.wasman.2008.06.039

Nickoloff, J. (2016). Keeping a Tidy Computer. En *Docker In Action* (1.ª ed., pp. 4-7). Manning.

Rueda-Avellaneda, J. F., Rivas-García, P., Gomez-Gonzalez, R., Benitez-Bravo, R., Botello-Álvarez, J. E., & Tututi-Avila, S. (2021). Current and prospective situation of municipal solid waste final disposal in Mexico: A spatio-temporal evaluation. *Renewable and Sustainable Energy Transition*, *1*, 100007. https://doi.org/https://doi.org/10.1016/j.rset.2021.100007

Sarita & Sebastian, S. (2017). Transform Monolith into Microservices using Docker. *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, 1-5. https://doi.org/10.1109/ICCUBEA.2017.8463820

SEMARNAT. (2013). Residuos. En *Informe de la situación del medio ambiente en México. Compendio de estadísticas ambientales. Indicadores clave y de desempeño ambiental* (1.ª ed., pp. 318-328).

SEMARNAT. (2019). Impacto de las Actividades Humanas en el Ambiente. En *Informe de la Situación del Medio Ambiente en México* (1.ª ed., pp. 63-66).

Serain, D. (1995). Client/server: Why? What? How? *International Seminar on Client/Server Computing. Seminar Proceedings (Digest No. 1995/184)*, *1*, 1/1-111 vol.1. https://doi.org/10.1049/ic:19951128

Somerville, I. (2016). Introduction to Software Engineering. En M. Goldstein (Ed.), *Software Engineering* (10.ª ed., pp. 27-50). Pearson.

Westby, E. J. H. (2015). Branching Strategies. En *Git for Teams* (1.ª ed., pp. 54-85). O'Reilly Media Inc.

Wolff, E. (2017). Preliminaries. En L. Lyons (Ed.), *Microservices: Flexible Software Architecture* (pp. 3-11).

Yadav, D., Gupta, D., Singh, D., Kumar, D., & Sharma, U. (2018). Vulnerabilities and Security of Web Applications. *2018 4th International Conference on Computing*

Communication and Automation (ICCCA), 1-5.
https://doi.org/10.1109/CCAA.2018.8777558