UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

GraphQL como lenguaje de consulta y manipulación de datos en el desarrollo de una aplicación Web

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Gerónimo Rubio Rivera

DIRECTOR DE TESIS

Dr. Guillermo Gilberto Molero Castillo

Ciudad Universitaria, Cd. Mx., 2023

Agradecimientos

Me gustaría expresar mi más profunda gratitud a mi director de tesis, el Dr. Guillermo Molero Castillo, por su inestimable orientación, apoyo y estímulo a lo largo de la finalización de mi trayectoria académica. Su dedicación a mi éxito ha sido una fuente constante de motivación y ha tenido un impacto significativo en mi crecimiento académico y personal. Estoy verdaderamente agradecido por la oportunidad de trabajar con un asesor y mentor tan increíble. Gracias por todo.

Me gustaría expresar mi más profunda gratitud a mi padre, por su inquebrantable confianza a lo largo de mi trayectoria académica. Su confianza en mí y en mis capacidades me ha dado la seguridad necesaria para perseguir mis sueños y ha sido la fuerza de mis logros. Estoy agradecido por sus sacrificios, las facilidades que me ha dado y por estar siempre ahí para mí. Gracias papá, por todo.

Me gustaría expresar mi más sincero agradecimiento a Ligia, por su apoyo y amor desde que nos conocimos. Su comprensión y su voluntad de estar a mi lado han tenido un impacto significativo en mi vida. Estoy agradecida por su bondad, compasión y por contar con ella siempre. Gracias Ligia, por todo.

Me gustaría expresar mi más sincero agradecimiento a mis abuelos, Bibis y Negrin, por su amor, apoyo y aliento a lo largo de mi trayectoria académica. Su sabiduría y sus experiencias vitales han sido una fuente constante de inspiración y orientación. Gracias por estar siempre a mi lado.

Quiero agradecer también a todas las personas que han contribuido directa o indirectamente a mi formación académica, a mi familia, amigos, al resto de los participantes en mi estudio, colegas y amigos. Sus contribuciones han sido esenciales para el término de mi carrera universitaria.

Por último, 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

La producción de residuos sólidos urbanos es un severo problema que enfrenta la Ciudad de México y va en aumento de manera exponencial conforme la población crece, siendo necesario contar con medidas para regular el manejo y aumentar la valorización de estos. **Problema de investigación.** Dado el crecimiento de la producción de residuos sólidos urbanos en la Ciudad de México, la Secretaría del Medio Ambiente (SEDEMA) propuso el desarrollo de una aplicación web para gestionar las actividades y residuos especificados en las normativas de LAU-CDMX, RAMIR, Plan de Manejo y Planes de Manejo de Bienes dentro de la Ciudad de México. **Objetivo.** La presente tesis se enfoca en el diseño, desarrollo e implementación de APIs, del lado del Back-End, utilizando como tecnología GraphQL, la cual es un lenguaje de consulta y manipulación de datos, enfocado en la gestión y administración de las operaciones de vinculación entre los diferentes usuarios. **Motivación.** En el presente, el consumismo ha ido en aumento gracias a diversos factores de la sociedad actual, provocando una crisis ambiental que debe ser tratada para garantizar el bienestar de la población. Gracias a esta plataforma web se busca gestionar y recuperar información útil para tomar mejores decisiones con respecto a los residuos sólidos urbanos generados en la Ciudad de México. **Método.** El método utilizado se basó en etapas de una metodología ágil, la cual fue estructurada en: a) diagnóstico, b) diseño, c) desarrollo, d) aseguramiento de la calidad, y e) desplegado. **Resultados.** Se implementó funcionalidades en forma de APIs para la gestión de ofertas, solicitudes, vinculación y administración de la plataforma web, denominado SEDEMA. Se logró integrar el Back-End y Front-End de manera eficiente gracias al uso de GraphQL. **Conclusión.** El desarrollo de las funciones basabas en GraphQL permitió implementar de manera exitosa APIs de consulta y manipulación de datos de los catálogos generales, de procesos, de referencia y de interfaz que forman parte de la aplicación web. Con base en esto se proporcionó una importante variedad de funciones útiles para el despliegue y llenado de formularios en la aplicación web.

Índice general

Índice de figuras

CAPÍTULO 1

Introducción

1.1 Contexto del proyecto

Desde el inicio de la Revolución Industrial, alrededor de 1750, diversas actividades humanas, como la quema de combustibles fósiles para la generación de energía; así como la ganadería, la silvicultura, la deforestación, han repercutido en la composición química de la atmósfera, ocasionando gradualmente lo que se conoce en el presente como efecto invernadero, consecuencia de la generación y acumulación de gases y otros compuestos producidos por los residuos de estas actividades (Cantú Martínez, 2014). Esta situación representa un problema que es de vital importancia atender para preservar la existencia de la vida y las especies en el planeta (Díaz Cordero, 2012).

De acuerdo con el Panel Intergubernamental de expertos sobre el cambio climático (IPCC, por sus siglas en inglés), el promedio global de la temperatura en la superficie de la tierra se ha incrementado en aproximadamente 0.75 °C durante el siglo pasado, de 1906 a 2005 (Semarnat, 2015). Además, los reportes recientes apuntan a que se estaría llegando a 1 °C arriba, con respecto a condiciones previas al inicio de la revolución industrial, y alcanzaría un nivel de 1.5 °C entre los años 2030 y 2052 (Arias et al., 2021). A manera representativa, la Figura 1 muestra la variación de la temperatura mundial entre 1880 a 2009, notándose un aumento cada vez mayor, en grados centígrados, de la temperatura media del planeta, ocasionado principalmente por las emisiones de gases de efecto invernadero.

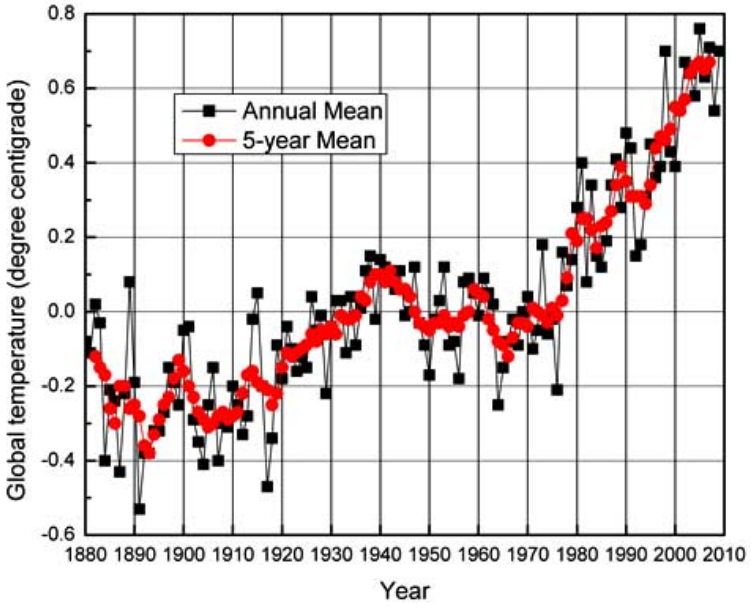

Figura 1: Variación de la temperatura mundial de 1880 a 2009.
Fuente: (Zeng et al., 2010)

Aunado a lo anterior, de acuerdo con el sexto informe de evaluación del IPCC (DE, 2018), los tres principales contaminantes son: Dióxido de carbono (CO_2), Metano (CH_4) y aerosoles (PM2.5 y PM10). El CO_2 se genera cuando se quema cualquier sustancia que contiene carbono, es producto también de la respiración y la fermentación. Las plantas absorben dióxido de carbono durante la fotosíntesis. Por su parte, el CH_4 es un gas incoloro, inflamable, no tóxico, que se produce de forma natural por la descomposición de la materia orgánica. Los humedales, el ganado y la generación de energía son las principales fuentes que emiten metano a la atmósfera, donde actúa como gas de efecto invernadero. Por último, pero no menos importante, son PM2.5 y PM10, que han sido los principales causantes de agrandar el agujero de la capa de ozono, que ha empeorado notablemente el calentamiento global. Esto se debe a que absorben y dispersan la luz solar, puesto que contienen una sustancia nociva llamada Clorofluorocarbonos (CFC's).

Por otro lado, a raíz de la pandemia por COVID-19 surgieron tres aspectos de observación para la humanidad. En primer lugar, el aire, el agua, los alimentos, la energía limpia y un entorno de vida basado en la atención médica general, el bienestar, la salud mental y el apoyo familiar son esenciales para la supervivencia y la sustenibilidad de la humanidad. En otras palabras, solo se puede llevar una vida plena en un entorno sano y seguro. En segundo lugar, las tecnologías digitales que utilizan Internet, servicios basados en la nube, inteligencia artificial y dispositivos móviles permiten el análisis de datos masivos desde cualquier punto de atención y en cualquier momento. De igual manera, el teletrabajo se ha convertido en una parte integral de la sociedad moderna, afianzando así el uso de las tecnologías digitales.

En tercer lugar, la sociedad moderna está inundada de elementos no esenciales, como viajes de ocio, discotecas y entretenimiento. En otras palabras, en la actualidad se consume más recursos por persona en comparación con la sociedad pre-moderna. No obstante, esta sociedad moderna es aún próspera gracias al abundante suministro de materiales, energía y agua, y es accesible para miles de millones de personas en todo el mundo. Por lo que, la generación de residuos sólidos crece con el consumo, debido a que los desechos a menudo no se reciclan adecuadamente y, por lo tanto, terminan en el suelo, el agua y el aire del planeta. Es decir, se está provocando el agotamiento de los recursos naturales y el aumento de la contaminación del ecosistema, lo que a su vez afecta la salud y el bienestar de los seres humanos.

1.2 Problemática

En la actualidad, el modelo de consumo vigente que se utiliza es el lineal, el cual consiste en "*tomar, hacer y tirar*", que confía en la disposición de grandes cantidades baratas y fácilmente accesibles de materiales y energía, además de medios baratos para deshacerse de lo que ya no es útil (Cerdá y Khalilova, 2016). Actualmente, aun una tercera parte del plástico no es recolectado o tratado globalmente (Agenda, 2016). Este modelo ya no es sostenible debido a la gran cantidad de desechos diarios que genera la población, por lo que una economía circular es una opción atractiva que se está empezando a utilizar en el ámbito empresarial (Foundation, 2015).

La economía circular tiene como objetivo generar prosperidad económica, proteger el ambiente y prevenir la contaminación, facilitando así el desarrollo sostenible mediante la filosofía de compartir, arrendar, reutilizar, reparar, renovar y reciclar los materiales y productos existentes durante el mayor tiempo posible (Sandoval et al., 2017). Algunas características claves para implementar esta economía circular son: menor uso de recursos naturales; mayor presencia de recursos y energías renovables; aumento de la proporción de materiales reciclables que puedan sustituir el uso de materiales vírgenes, reducción de emisiones mediante ciclos de materiales limpios; reducción de desechos y pérdidas de material; limitación de la incineración y el vertido al mínimo; entre otros.

Por lo que, para cumplir con una economía circular es recomendable considerar factores facilitadores, como la elaboración de diseños ecológicos mediante la reutilización de materiales, tener en cuenta la reparación, renovación y remanufacturación, antes de desechar un producto. Al igual, es necesario invertir fondos monetarios para los recursos naturales y la contaminación, removiendo subvenciones perjudiciales para el ambiente. Por último, se

debe educar e incentivar la participación de los procesos de reciclaje y reutilización de los materiales.

En este sentido, para el caso de la Ciudad de México, objeto de estudio en este trabajo de tesis, y con base en los requerimientos y necesidades de software establecidos por la Secretaría de Medio Ambiente de la Ciudad de México (SEDEMA), a través de la Secretaría de Educación, Ciencia, Tecnología e Innovación de la Ciudad de México (SECTEI), resulta necesario contar con una plataforma tecnológica capaz de proveer información de las cantidades de residuos que se generan y son manejados por los tres actores en este flujo: generador, transportista y reciclador. Esto con el fin de tener un control adecuado para la toma de decisiones dentro de la SEDEMA, para así contribuir en la conservación del ambiente y la salud de la población.

1.3 Objetivos

1.3.1. Objetivo general

- Contribuir en el desarrollo de una aplicación Web para la Secretaría del Medio Ambiente de la Ciudad de México, mediante la implementación de GraphQL como lenguaje de consulta y manipulación de datos en la plataforma Web para la administración del encadenamiento productivo de residuos sólidos urbanos producidos en la Ciudad de México.

1.3.2. Objetivos específicos

- Realizar el análisis y diseño de los componentes de software requeridos por el usuario final.
- Diseñar e implementar una base de datos de usuarios que permita identificar el tipo de material o servicio que se oferta, cantidad del material que se oferta o requiere, o tipo de servicio que se oferta, y la ubicación geográfica del material o de dónde se requiere el mismo.
- Construir y validar el funcionamiento de APIs (interfaz de programación de aplicaciones) implementadas en GraphQL.

1.4 Motivación

Los desechos producidos por las actividades humanas pueden, en cierta medida, ser manipulados por la naturaleza a través de su capacidad de reabsorción de la fracción orgánica de los residuos, por ejemplo, se descomponen rápidamente en climas cálidos y húmedos bajo la acción de microorganismos. Sin embargo, cuando se excede la capacidad de reabsorción de la naturaleza, debido a la gran cantidad de desechos que se generan, se produce una acumulación de desechos que conduce a efectos adversos sobre los ecosistemas y seres humanos. Para hacer frente a esto, se han desarrollado tecnologías de tratamiento de residuos al final de su vida útil, para así reducir la intensidad de los procesos de producción y otras actividades antes de descargarlos al ambiente.

Los residuos sólidos que se producen en el planeta se clasifican, de acuerdo a sus características de origen, como: i) Residuos Sólidos Urbanos (RSU), ii) Residuos de Manejo Especial (RME), y iii) Residuos Peligrosos (RP). En México, según la cifra más reciente publicada en 2015, la generación de Residuos Sólidos Urbanos alcanzó 53.1 millones de toneladas, lo que representó un aumento del 61.2 % con respecto a 2003, esto es, 10.24 millones de toneladas más generadas en ese período. Si se expresa por habitante, alcanzó 1.2 kilogramos en promedio diariamente en el mismo año (Liu et al., 2019).

Aunado a lo anterior, la producción y el consumo de bienes y servicios generan inevitablemente algún tipo de residuos. Estos pueden ser sólidos (ya sea de naturaleza orgánica o inorgánica), líquidos (que incluyen a los que se vierten disueltos como parte de aguas residuales), y en forma de gases. Todos estos, en función de su composición, tasa de generación y manejo, pueden tener efectos diversos en la población y el ambiente. En algunos casos, sus efectos pueden ser graves, sobre todo cuando involucran compuestos tóxicos que se manejan de manera inadecuada o se vierten de manera accidental. Por lo que, en la Ciudad de México se tienen diferentes instrumentos de política ambiental, en los que se concentran diversas obligaciones ambientales de los responsables de fuentes fijas que están sujetos a las disposiciones de la Ley Ambiental de Protección a la Tierra en la Ciudad de México.

El primero de estos instrumentos es el Registro y Autorización de establecimientos y/o unidades de transporte relacionados con el Manejo Integral de Residuos de competencia local que operen y transiten en la Ciudad de México (RAMIR). Otra es la Licencia Ambiental Única (LAU-CDMX) que se encarga del control de la contaminación de las empresas que tenga como finalidad desarrollar operaciones o procesos industriales, de servicios o actividades que generen o puedan generar emisiones contaminantes a la atmósfera. Mediante estos instrumentos, las empresas presentan las cantidades y características de los residuos sólidos

generados a la Secretaría de Medio Ambiente y Recursos Naturales (SEMARNAT).

Es importante mencionar que cada uno de estos permisos cuentan con su propio sistema de registros. Sin embargo, los reportes anuales de los residuos llegan a tardar más de un año en ser generados debido a que es un proceso que se realiza en físico, llegando a ser tardado. Por lo que, no se puede tener una visión realista del estado de generación de residuos y contaminación ambiental real. Por lo que, fue necesario el desarrollo de una plataforma web, que busca agilizar este proceso para tener los registros en tiempo real, y en cualquier momento.

1.5 Organización del documento de tesis

El documento de tesis fue divido en cinco capítulos. En este primer capítulo se establece el contexto de la investigación, las bases del problema identificado, se definen los objetivos y se establece la motivación del trabajo realizado. Dando lugar al desarrollo de la investigación, cuyos resultados se presentan en los capítulos siguientes.

En el Capítulo 2 se presentan los bases teóricas sobre la contaminación ambiental, los residuos sólidos urbanos y la economía circular. Asimismo, se describen los fundamentos del desarrollo Web, las tecnologías y frameworks utilizados, la importancia de las APIs, entre otros. Por otra parte, presentan los trabajos relacionados con este desarrollo y una síntesis del mismo.

El Capítulo 3 presenta la propuesta de solución de la construcción de la aplicación Web mediante la implementación de GraphQL como lenguaje de consulta y manipulación de datos. Para esta construcción se siguió el ciclo de desarrollo de software mediante la metodología ágil SCRUM.

El Capítulo 4 presenta los resultados obtenidos, basados en la creación de la plataforma Web y las funcionalidades de la aplicación. El Capítulo 5 presenta las conclusiones generales y particulares del proyecto realizado, y se establecen los trabajos futuros que se podrían incluir, con base en nuevos requerimientos definidos por los usuarios finales, esto es, usuario SEDEMA.

Finalmente, en el Anexo A se presenta el diccionario de datos de la base datos, mientras que en el Anexo B se presenta el código fuente de las diferentes consultas y mutaciones implementadas en GraphQL.

CAPÍTULO 2

Marco teórico y estado del arte

Actualmente, alrededor del mundo hay una gran demanda de productos, que ocasiona que las actividades humanas hagan uso de los recursos naturales; al mismo tiempo se genera una gran cantidad de residuos y gases. Esto conlleva a la acumulación de gases de efecto invernadero (GEI), que se encuentran en el vapor de agua (H_2O), el dióxido de carbono (CO_2), el metano (CH_4), el óxido nitroso (N_2O) y el ozono (O_2). Los GEI son aquellos compuestos gaseosos en la atmósfera terrestre que atrapan el calor proveniente de la radiación infrarroja proveniente del Sol, actuando como aislantes globales que provocan el aumento en la temperatura del planeta, a esta situación se conoce como efecto invernadero.

De acuerdo con la ONU, 2019, las concentraciones de los principales gases de efecto invernadero alcanzaron niveles récord en 2018, donde el dióxido de carbono aumentó un 147 %, el metano un 259 %, y el óxido nitroso un 123 %. El aumento de metano debe su gran aumento debido a las actividades humanas como la cría de ganado, arrozales, minas, vertederos y la quema de biomasa, contribuyendo aproximadamente a un 60 % del metano producido. Es por esto que mediante la construcción de una aplicación Web se busca contribuir en promover una economía circular para el manejo y aprovechamiento de los residuos sólidos urbanos en la Ciudad de México.

En este capítulo se describe la fundamentación teórica y el estado del arte del trabajo de investigación, donde se presenta los residuos sólidos urbanos, su composición y regulación en la Ciudad de México. Además, se describen las características que posee la economía circular. Por otra parte, se presentan las diferentes arquitecturas, herramientas y tecnologías utilizadas para el desarrollo Web. Finalmente, se dan a conocer los trabajos relacionados que tienen incidencia en el manejo de residuos sólidos urbanos.

2.1 Residuos sólidos

Debido a la constante demanda de bienes y servicios en la Ciudad de México, como ha venido resultando año tras año, la generación de residuos urbanos (RSU) se ha incrementado de manera proporcional con respecto al crecimiento de la población, debido a los hábitos de consumo y uso de productos; así como a los sistemas de producción y economía lineal que actualmente domina. Durante 2016, la Ciudad de México generó 12920 toneladas de residuos sólidos urbanos por día, mientras que para 2019 esta cifra alcanzó las 13149 toneladas diarias (SEDEMA, 2019), tal como se muestra en la Figura 2.

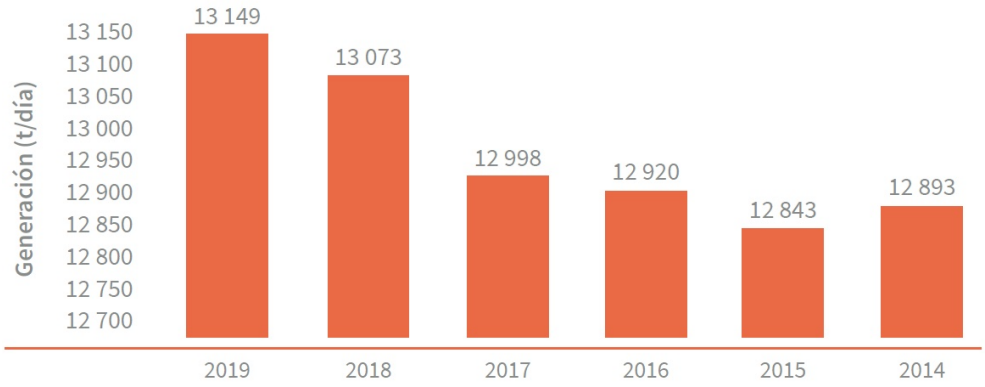

Figura 2: Generación de RSU en la Ciudad de México de 2014 a 2019
Fuente: (SEDEMA, 2019)

En la Ciudad de México, la generación de RSU no es acorde al tamaño de cada alcaldía, sino que hay diversos factores como la población, el nivel económico, las actividades económicas, sociales y ambientales que predominan en la alcaldía, la cultura del reciclaje, entre otros factores. En 2019, las alcaldías de Iztapalapa, Gustavo A. Madero y Cuauhtémoc fueron las que más residuos generaron, representando el 42.79 % de todos los residuos generados en la ciudad (Figura 3), en comparación con Milpa Alta, Cuajimalpa de Morelos y La Magdalena Contreras, que representaron solo el 4.58 %.

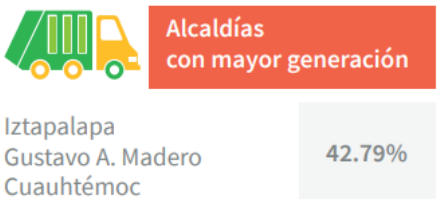

Figura 3: Alcaldías con mayor generación 2019
Fuente: (SEDEMA, 2019)

En otro aspecto de interés es que la generación per cápita no es acorde respecto a la población ni extensión de la alcaldía. Las alcaldías que generan la mayor cantidad de RSU por

persona son Cuauhtémoc, Miguel Hidalgo y Venustiano Carranza. Esto se debe a que se ubican en el centro de la ciudad y hay mayor actividad comercial y lugares turísticos. En cambio, las alcaldías de Cuajimalpa de Morelos y Milpa Alta, al contar con zonas de conservación, presentan una menor generación por habitante. En la Figura 4 se puede observar la generación total de RSU total y per cápita para cada alcaldía.

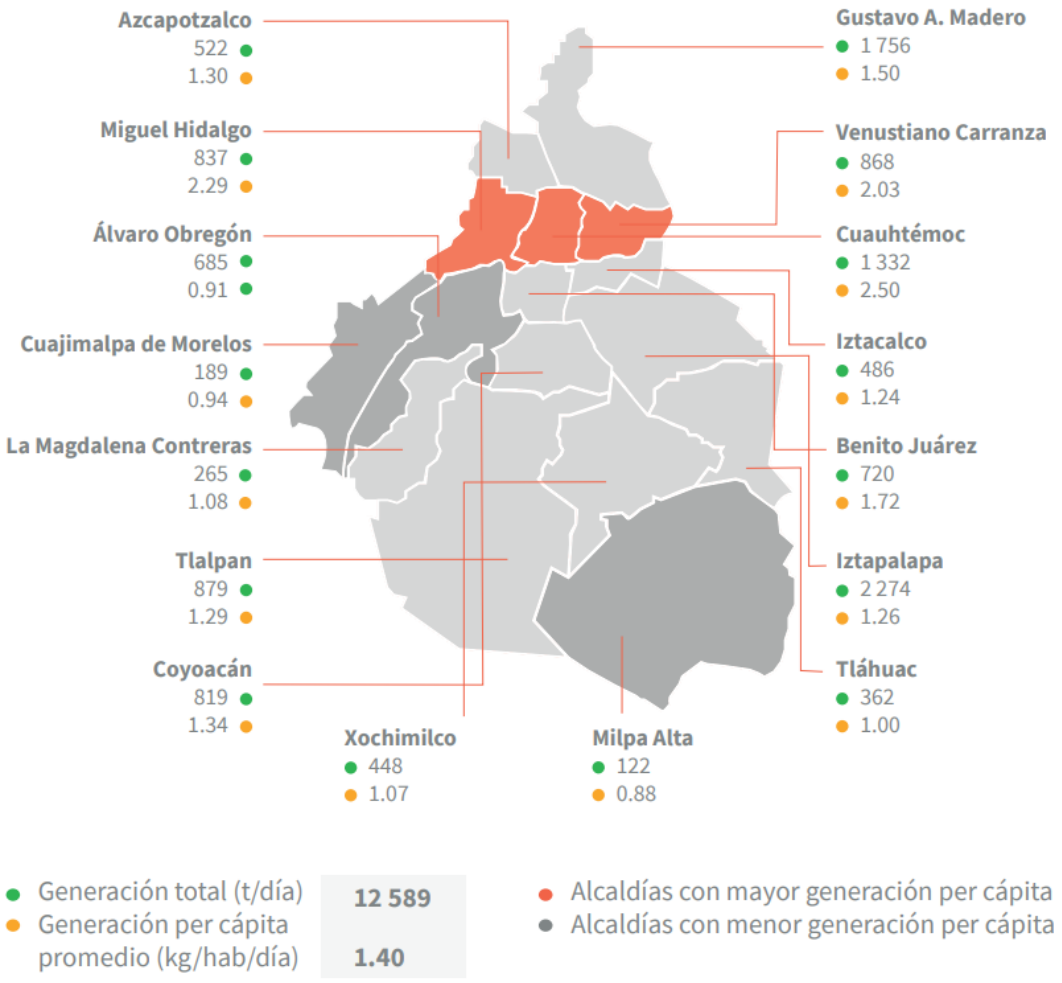

Figura 4: Mapa de alcaldías con mayor generación en 2019
Fuente: (SEDEMA, 2019)

De acuerdo con lo anterior, la Ciudad de México genera aproximadamente 12589 toneladas de residuos sólidos diarios y aproximadamente 1.4 kilogramos por habitante durante el día. Donde, como se mencionó previamente, las alcaldías que generan la mayor cantidad de residuos por persona son: Cuauhtémoc con 2.5 kg/hab/día, Miguel Hidalgo con 2.29 kg/hab/día, y Venustiano Carranza con 2.03 kg/hab/día. Según el gobierno local, esta cantidad de residuos equivale a llenar la plancha del Zócalo de la Ciudad de México a una altura de tres metros.

2.1.1. Composición

En general, la predominancia de residuos se asocia a la condición económica de la población. En México, los componentes importantes de los residuos sólidos urbanos son la comida, jardines y materiales orgánicos (52.4 %), el papel y sus derivados (13.8 %), los plásticos (10.9 %), vidrio (5.9 %), entre otros, tal como se observa en la Tabla 2.1.

Residuos sólidos urbanos	Porcentaje
Residuos de comida, jardines y materiales orgánicos similares	0.524
Papel, cartón, y otros productos de papel	0.138
Otro tipo de basura	0.121
Plásticos	0.109
Vidrio	0.059
Aluminio	0.017
Textiles	0.014
Metales ferrosos	0.011
Otros metales no ferrosos	0.006

Cuadro 2.1: Composición de residuos sólidos urbanos en México. **Fuente:** (Sedesol, 2012)

De acuerdo con Sedesol, 2012, desde el punto de vista ambiental y de salud pública, el manejo adecuado de los residuos sólidos, en las etapas que siguen a su generación, permite mitigar los impactos negativos en el ambiente, la salud y reducir así la presión sobre los recursos naturales. Por lo que, el reuso y reciclaje de materiales son fundamentales para reducir la presión sobre los ecosistemas. A su vez, desde el punto de vista económico, el monto destinado al manejo y tratamiento de residuos en los países miembro de la OECD (Organización para la Cooperación y el Desarrollo Económicos) asciende a cerca de una tercera parte de los recursos financieros que se destina para el control de la contaminación.

2.1.2. Regulación

La Secretaría de Medio Ambiente y Recursos Naturales (SEMARNAT) y la Secretaria del Medio Ambiente de la Ciudad de México (SEDEMA) cuentan con diversos instrumentos de regulación para controlar y prevenir el impacto ambiental debido a la generación de residuos sólidos urbanos por las actividades humanas en la Ciudad de México.

2.1.2.1. Licencia Ambiental Única

La Licencia Ambiental Única (LAU) es un instrumento de regulación emitida por la SEMARNAT para los sectores industriales considerados de jurisdicción federal, para el proceso de evaluación, dictamen y resolución de los trámites ambientales que los responsables

de establecimientos industriales deben cumplir ante la SEMARNAT en materia de impacto y riesgo ambiental, emisiones a la atmósfera, generación y tratamiento de residuos peligrosos. La LAU aplica a los sectores industriales establecidos en el Artículo 111 Bis de la Ley General del Equilibrio Ecológico y Protección al Ambiente; estos son: automotriz, química, celulosa y papel, asbesto, pinturas y tintas, vidrio, cemento y cal, generación de energía eléctrica, tratadores de residuos peligrosos, metalúrgica, siderúrgica, petróleo y petroquímica.

Se emite por única vez y en forma definitiva conforme a la actividad productiva principal y la localización del establecimiento y es única por establecimiento industrial. Tiene que renovarse por cambio de giro industrial o de localización y debe actualizarse por aumento de la producción, cambios de proceso, ampliación de instalaciones, manifestación de nuevos residuos peligrosos o cambio de razón social.

2.1.2.2. RAMIR

El 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 (RAMIR), es una autorización otorgada por la Secretaría del Medio Ambiente del Gobierno de la Ciudad de México a las personas físicas y morales, titulares de los establecimientos mercantiles, de servicios o de unidades de transporte de residuos sólidos que transiten dentro de la jurisdicción territorial de dicha entidad, relacionados con la recolección, acopio, almacenamiento, aprovechamiento, valorización, manejo, transporte, tratamiento, reutilización, reciclaje, y disposición final de los residuos sólidos urbanos y de manejo especial de competencia local.

La finalidad de RAMIR es vigilar el correcto funcionamiento de los mismos, comprobando los controles inherentes para su manejo y validando que su operación se enmarque en las disposiciones jurídicas y ambientales aplicables. Además, el RAMIR tiene una vigencia de 2 años y debe ser renovada a su término. Este tipo de autorización se puede obtener bajo las siguientes modalidades:

- Recolección y transporte.
- Acopio y almacenamiento.
- Reciclaje, reutilización, tratamiento o valorización.
- Disposición final.

2.1.2.3. Plan de manejo de residuos

El Plan de manejo de residuos de competencia local no sujetos a Licencia Ambiental Única para la CDMX es una autorización dirigida a las personas físicas o morales que se dediquen a la recolección, transporte, tratamiento, reciclaje, reúso, acopio, almacenamiento, o disposición final que genere un promedio igual o superior a 50 kilos diarios de residuos sólidos urbanos o cualquier cantidad de residuos de manejo especial de competencia local.

Es a través de estos planes, programas y marcos regulatorios que se promueven que el manejo de los residuos sólidos urbanos se realice bajo esquemas de gestión integral, que incluyan la prevención y reducción de su generación, su valorización económica y su disposición de manera adecuada.

2.2 Economía circular

Durante las últimas décadas, el consumo de materias primas ha aumentado conforme a la demanda de la población, comprometiendo a la mantenibilidad del ecosistema del planeta. Los gobiernos y empresas han abordado esta problemática con diferentes modelos industriales y desde una perspectiva lineal, mediante técnicas correctivas y modernización tecnológica, que si bien puede comprar tiempo, no asegurar la mantenibilidad a largo plazo (Sandoval et al., 2017). Es por esto que una economía circular es una opción necesaria en el modelo actual.

La economía circular tiene como objetivo generar prosperidad económica, proteger el ambiente y prevenir la contaminación, facilitando así el desarrollo sostenible de las generaciones presentes sin comprometer a las generaciones futuras (Sandoval et al., 2017). De acuerdo con AIR et al., 2016, para implementar una economía circular es necesario tener en cuenta características claves, como:

1. **Menor entrada y uso de recursos naturales.** Reducir la dependencia de los recursos naturales, minimizando y optimizando la explotación de los mismos. Así como reducir el uso de energía y agua.

2. **Mayor presencia de recursos y energías renovables y reciclables.** Sustitución de los recursos no renovables por otros renovables dentro de unos niveles de suministro sostenibles. Aumento de la proporción de materiales reciclables y reciclados que pueden sustituir el uso de materiales vírgenes.

3. **Reducción de emisiones.** Menos contaminación mediante ciclos de materiales limpios.

4. **Reducción de desechos y pérdidas de material.** Minimizar la acumulación de residuos. Limitación de la incineración y el vertido al mínimo.

5. **Mantener el valor de los productos, componentes y materiales en la economía.** Extender la vida útil de los productos, reutilización de los componentes y un reciclaje adecuado.

2.3 Desarrollo web

Actualmente, navegar por la Web es parte fundamental de la vida cotidiana de las personas, mediante la cual se puede compartir y acceder información, documentos y recursos; así como servir de medio para comunicación con otros usuarios. La Web apareció a principios de 1989, inventada por el científico británico Tim Berners Lee. La Web se diseñó en primera instancia para satisfacer la demanda de intercambio de información entre universidades e institutos científicos del mundo.

En sus inicios la Web fue una colección de sitios creados en texto y con formato HTML, pero con pocos o carentes contenidos de estilo. No obstante, desde su creación, no ha dejado de cambiar y perfeccionarse, ha pasado de la Web 1.0 a la 2.0 y 3.0 (Figura 5):

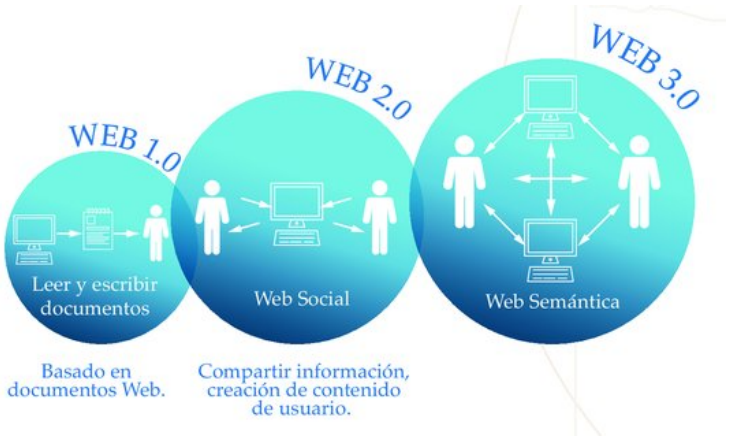

Figura 5: Diagrama de la Web 1.0, 2.0 y 3.0.

- La Web 1.0 consistía en tan solo páginas con texto plano estático sin la posibilidad de publicar nada, el usuario tan solo podía buscar y leer la información disponible. Las tecnologías utilizadas en la Web 1.0 son HTML, HTTP y URI (Choudhury, 2014), así como, XML o XHTML como protocolos para el intercambio de información. Del lado del servidor se utiliza JavaScript, VBscript y Flash en el cliente (Nath et al., 2014). Por otra parte, se utilizaban tecnologías variadas para el cliente y el servidor, como: ASP, PHP, JSP, CGI, PERL, y los sitios con estilos utilizaban CSS. La Web 1.0 es lenta

y el usuario necesita refrescar el sitio cada vez que se añade nueva información a las páginas Web.

- La segunda versión de la Web fue presentado en 2004 por Dale Dougherty conocida como la Web de escritura y lectura, o simplemente Web social, que permite la transferencia de datos de manera bidireccional, permite además al usuario publicar. La infraestructura tecnológica utilizada en la Web 2.0 consiste en RSS, Atom y RDF, empleadas para la creación de los servicios de la Web. Se utiliza también tecnología Ajax como JavaScript y XML, DOM, REST, XML y CSS. Esta versión permite a los usuarios tener la capacidad de crear actividades sociales y comunicarse entre sí. Sin embargo, estas propiedades presentaban también problemas, debido a que el usuario puede ser hackeado, poniendo en riesgo su privacidad y la seguridad de la información personal.
- La versión actual de la Web iniciada en 2014, conocida como Web ejecutable, o Web semántica, permite al usuario darle la capacidad de interactuar con aplicaciones dinámicas. La teoría de Conrad Wolfram sobre la Web 3.0 trata de hacer que el ordenador sea capaz de pensar y sea más inteligente para la búsqueda de nuevos datos en lugar de las personas, accediendo a la información de manera más rápida (Nath et al., 2014). Asimismo, las aplicaciones Web deben considerar ser ejecutadas en cualquier plataforma, y que estén interconectadas mediante un perfil que se usará para personalizar la navegación por la Web, basada en su historial. Esto permite que cada usuario obtenga de una búsqueda diferentes resultados acordes a su interés.

2.4 Tecnologías

Las tecnologías utilizadas para el desarrollo de la plataforma Web fueron seleccionadas con base en los conocimientos previos del equipo de trabajo, así como el interés por innovar con nuevas tecnologías. Por otra parte, se tomó en cuenta que son herramientas de uso gratuito o código libre, con documentación disponible, y con una amplia comunidad de apoyo en caso de requerir la clarificación ante alguna duda en el desarrollo. Por último, se eligieron las herramientas de mayor uso en el desarrollo Web, tomando como referencia a empresas como Google, Facebook, Twitter, entre otros.

2.4.1. PostgreSQL

PostgreSQL es un motor de bases de datos que con el paso del tiempo ha logrado posicionarse de manera importante en el negocio de las bases de datos. Este motor cuenta con una amplia documentación, así como actualizaciones frecuentes. Hoy en día, es el cuarto gestor de base de datos más utilizado y el primero que cuenta con todas las funcionalidades de manera gratuita (Statista Research Department, 2022). PostgreSQL utiliza y amplía

el lenguaje SQL combinado con otras características que almacenan y escalan las cargas de trabajo de datos de manera segura.

Por otra parte, PostgreSQL otorga a los desarrolladores facilidades para crear aplicaciones, así como a los administradores funciones capaces de proteger la integridad de los datos y crear entornos tolerantes a fallos, además de ser gratuito y de código abierto. Una de las empresas importantes en el mundo que utiliza PostgreSQL es Apple, que en 2010 sustituyó MySQL por PostgreSQL como base de datos integrada en la versión de OS X Lion, que es la octava versión de OS X, el sistema operativo de Apple para las computadoras de escritorio, portátiles y servidores (Shimel, 2011). Otras empresas que utilizan PostgreSQL como gestor de base de datos son: Instagram, Reddit, Skype, Spotify; esto debido a su accesibilidad de implementación y su poder de almacenamiento y escalabilidad (Engineering, 2016).

2.4.2. Go

Se eligió Go como lenguaje de programación para el desarrollo del Back-End. Go es un lenguaje de programación de código abierto desarrollado por Google en 2007 por Robert Griesemer, Rob Pike, y Ken Thompson. Inicialmente, era de uso privado, pero en 2009 se hizo público y finalmente en marzo de 2012 se liberó la versión 1.0. Algunas de las características más significativas de Go son:

- Tipado estático (Statically typed), Los tipos de datos de las variables se declaran explícitamente y son revisados en tiempo de compilación.
- Recolección de basura (Garbage collection). Recuperación sistemática del almacenamiento acumulado que está siendo utilizado por un programa cuando este ya no lo necesita. Esto libera espacio en almacenamiento para que se utilice en otros procesos.
- Ejecución simultánea de múltiples tareas, aprovechando al máximo los recursos disponibles para completar las solicitudes de manera más rápida.

De igual manera, Go ofrece ventajas como la facilidad de aprender y empezar a usar, puesto que utiliza una sintaxis similar a C y C++. Por otro lado, con Go es posible desarrollar todo tipo de aplicaciones, ya sea para sistemas operativos Windows, Linux y también para dispositivos móviles. Estos factores han ocasionado que los desarrolladores se interesen a usarlo, y se amplíe la comunidad de usuarios detrás de este lenguaje de programación.

En otros aspectos de interés, el código fuente se compila directamente en código máquina y en un solo archivo ejecutable. Este archivo no sufre ningún tipo de dependencia y puede

ser cargado y ejecutado en cualquier lugar. Esto hace que el proceso de ejecución del código sea rápido.

2.4.3. Flutter

La ingeniería de software basada en componentes (CBSE, por sus siglas en inglés), también llamada desarrollo basado en componentes (CBD, por sus siglas en inglés) es una rama de la ingeniería de software que hace énfasis en la separación de funcionalidades que se requieran en un sistema de software determinado. Se trata de un enfoque basado en la reutilización para definir e implementar componentes independientes, débilmente acoplados en el sistema. Esta práctica pretende aportar beneficios, tanto a corto y largo plazo, para el propio software y para las organizaciones que así lo requieran.

En este sentido, Flutter es un kit de desarrollo de software (SDK) de código libre para el desarrollo de aplicaciones para dispositivos móviles, web y de escritorio creado por Google, que utiliza Dart como lenguaje de programación. Diseñado con el objetivo de proporcionar a los desarrolladores una herramienta para construir aplicaciones responsivas de forma nativa para las diferentes plataformas que soporta, maximizando la reutilización de código.

Flutter sigue el enfoque de desarrollo basado en componentes. Esto consiste en dividir el software en piezas identificables que los desarrolladores de aplicaciones escriben y despliegan de forma independiente. Para Flutter estos componentes básicos son los widgets. Cada widget dentro de Flutter es utilizado para crear un widget de mayor nivel y la composición de varios de estos en widgets aún más complejos, y así hasta crear una aplicación totalmente funcional.

En la versión 2.10 de Flutter existen 172 widgets disponibles incluidos de base con el SDK (Documentation, 2022). De acuerdo con Napoli, 2019, una manera de organizar esta gran cantidad de componentes es la siguiente:

- **Widgets de valor**: Son los widgets que contienen un valor, en ocasiones valores procedentes del almacenamiento local, de un servicio o una entrada del usuario. Se utilizan para mostrar valores al usuario y para obtener valores del usuario en la aplicación.
- **Widgets de diseño**: Son los widgets que proporcionan la organización de la escena. Indican la localización de los widgets, así como, hacerlos móviles, entre otras características de diseño.

- **Widgets de navegación**: Estos controlan la forma en que el usuario navega entre las escenas de la aplicación. Algunos ejemplos son los botones o la barra de desplazamiento.
- **Otros widgets**: Los widgets restantes entran dentro de esta clasificación, en las que se encuentran algunos como GestureDetector utilizado para detectar gestos en dispositivos móviles o incluso en el panel táctil de laptops, o Cupertino especializado en interfaces de usuario para productos de aplicaciones de iOS.

2.4.4. Docker

Docker es una plataforma de código abierto que facilita el proceso de desarrollo, distribución y ejecución de aplicaciones con el objetivo de reducir el tiempo que transcurre entre la escritura del código fuente y el despliegue en producción. Con Docker es posible separar todas las dependencias necesarias para que una aplicación se ejecute dentro de zonas controladas y aisladas conocidas como contenedores.

Un contenedor es la abstracción de la capa de aplicación que empaqueta el código y todas sus dependencias para que el software se ejecute de manera uniforme, a pesar de las diferencias del entorno actual. Una imagen de un contenedor es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación (Figura 6): código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones.

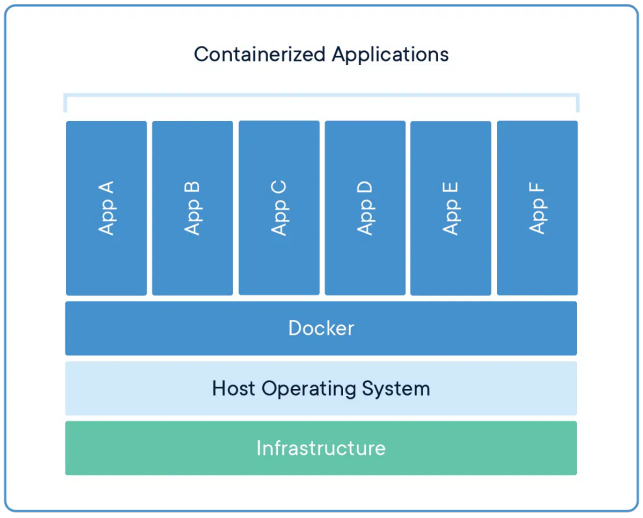

Figura 6: Estructura de Docker. **Fuente:** (Docker, 2022)

A diferencia de las máquinas virtuales que virtualizan el hardware, los contenedores virtualizan el software del sistema operativo. Cuando se crea una máquina virtual, esta realiza

una instalación completa de todo el sistema operativo, incluyendo aplicaciones, bibliotecas y archivos binarios, requiriendo gigabytes de espacio de almacenamiento (Figura 7). En cambio, un contenedor se ejecuta como proceso aislado en el espacio del usuario. Los contenedores ocupan menos espacio que las máquinas virtuales. Además, pueden manejar más aplicaciones y requieren menos máquinas virtuales y sistemas operativos.

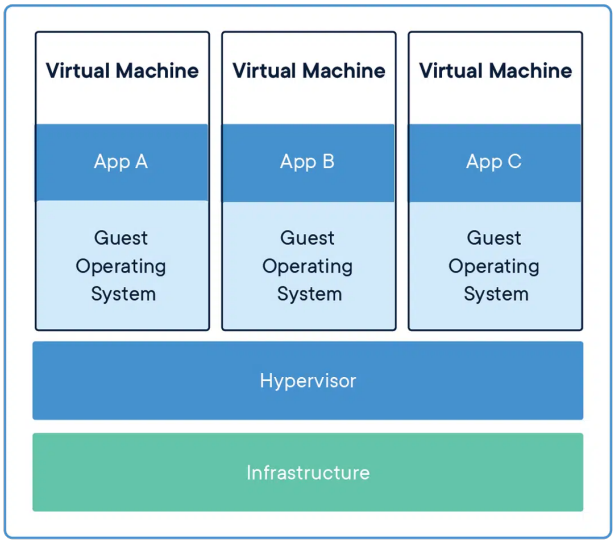

Figura 7: Estructura de las máquinas virtuales. **Fuente:** (Docker, 2022)

2.5 APIs

Una interfaz de programación de aplicaciones (API) es un conjunto de reglas que definen como se comunican las aplicaciones y sistemas entre sí. En este caso, la comunicación es entre el cliente y el servidor, esto con el objetivo de integrar la funcionalidad de diversos componentes, sin importar como están implementados a través de una interfaz documentada. Hoy en día, el uso de las APIs es esencial dentro de la industria del software, puesto que simplifica el desarrollo de las aplicaciones, permitiendo ahorrar tiempo y dinero.

Una API al actuar como intermediario provee la abstracción de funcionalidad entre dos sistemas, garantizando un nivel de seguridad adicional, esto debido a que en el punto final de la API se desvincula la aplicación que consume la infraestructura que proporciona el servicio. Así, las llamadas a la API suelen incluir credenciales de autorización para reducir el riesgo de ataques al servidor y limitar el acceso para minimizar las amenazas a la seguridad.

Las APIs se sitúan entre las aplicaciones y los servidores, actuando como una capa intermediaria que procesa la transferencia de datos entre sistemas. El funcionamiento general de una API, mediante el cual se acceden a los recursos dentro del servidor, es la siguiente:

1. El cliente inicia una solicitud a la API para recuperar información. Esta solicitud se procesa desde una aplicación al servidor Web a través del Identificador Uniforme de Recursos, o URI, que está conformada por un verbo de solicitud, cabeceras y un cuerpo de solicitud.

2. Tras recibir una solicitud válida, la API realiza una llamada al servidor Web, extrayendo con la implementación interna de los datos requeridos.

3. El servidor envía una respuesta a la API con la información solicitada.

4. El cliente recibe la información enviada por el servidor.

Para que se tenga comunicación entre el servidor y el cliente es importante definir el formato de intercambio de datos, que no son más que un estándar de archivos que utilizan las aplicaciones para trasportar la información. Antes, el formato más utilizado fue XML (Extensible Markup Language). Esto significa que, a diferencia de otros lenguajes, XML no está predefinido, por lo que se debe definir las etiquetas. En el Código 1 se muestra, a manera de ejemplo, un extracto de código XML.

```xml
<?xml version="1.0"?>
<Catalog>
   <Book id="bk102">
      <Author>Garcia, Debra</Author>
      <Title>Midnight Rain</Title>
      <Genre>Fantasy</Genre>
      <Price>5.95</Price>
      <PublishDate>2000-12-16</PublishDate>
      <Description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</Description>
   </Book>
</Catalog>
```

Código Fuente 1: *Ejemplo de XML (Extensible Markup Language).*

Sin embargo, en la actualidad uno de los formatos más utilizados es JSON (JavaScript Object Notation), el cual se compone de una colección de pares: nombre y valor. En varios lenguajes esto es similar a diferentes estructuras de datos como un objeto, registro, estructura, diccionario, tabla Hash o lista de claves. Los mensajes JSON se intercambian en un formato de texto legible para el ser humano y es interoperable entre la mayoría de los lenguajes de programación y no tiene ningún esquema asociado. El Código 2 se muestra un extracto de código JSON.

```
{ "nombre":"Jonh Doe",
  "profesion":"Programador",
  "edad":25,
  "lenguajes":["PHP","Javascript","Dart"],
  "disponibilidadParaViajar":true,
  "rangoProfesional": {
      "aniosDeExperiencia": 12,
      "nivel": "Senior"
  }}
```

Código Fuente 2: *Ejemplo de código JSON.*

2.5.1. RESTful

REST (REpresentational State Transfer) es un estilo de reglas que conforman una arquitectura de software para servicios Web, las aplicaciones que siguen un estilo REST son conocidas como API RESTful, las cuales utilizan solicitudes HTTP para acceder a los datos dentro de una arquitectura cliente-servidor con las siguientes características:

- Gestión de recursos vía métodos HTTP/1.1. POST para crear recursos, GET para solicitar recursos, PUT/PATCH para actualizar recursos y DELETE para eliminar recursos.
- Comunicación sin estado, esto es, que el estado necesario para gestionar la solicitud está contenido en la propia solicitud y el servidor no almacena nada relacionado con la sesión. El cliente debe incluir toda la información como parte de los parámetros de consulta, las cabeceras o la URI.
- Cada respuesta se debe incluir si esta se guarda en el caché, y por cuánto tiempo pueden ser almacenadas en el lado del cliente.
- La arquitectura de la aplicación está compuesta por múltiples capas, únicamente comunicadas con las capas inmediatas.

La ventaja de utilizar una arquitectura API RESTful (Figura 8) es que permite una mayor escalabilidad, puesto que al tener una comunicación sin estado, el servidor no tiene que enfocarse en manejar el estado de la sesión. Por otra parte, una buena implementación del caché permite aún más escalabilidad y un mejor rendimiento de la aplicación. Sin embargo, actualmente el principal problema es el uso del protocolo HTTP/1.1, puesto que el incremento de solicitudes HTTP en los sitios web actuales aumenta la carga de los servidores y ralentiza los tiempos de carga, afectando la latencia.

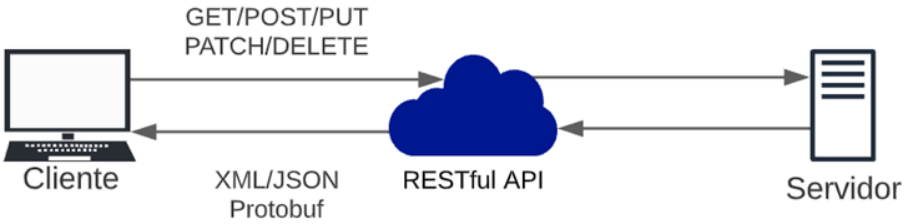

Figura 8: Arquitectura RESTful.

RESTful es una arquitectura recomendada para la integración de APIs, fácil de usar y con restricciones de protocolo sencillas de implementar. Por otra parte, posee una amplia documentación y una comunidad de usuarios a quienes consultar dudas. Como desventaja esta arquitectura genera congestiones de tráfico cuando se presentan llamadas al servidor al mismo tiempo, y la latencia se ve afectada, necesitando un gran ancho de banda.

2.5.2. GraphQL

GraphQL (Graph Query Language) un lenguaje de consulta para la construcción de una API, utilizando un sistema de tipos definidos de acuerdo a las necesidades de los datos y proporcionando funciones para obtener los recursos desde la base de datos o realizando operaciones de control. Por otra parte, GraphQL no está vinculado a ninguna base de datos o motor de almacenamiento específico, sino que está respaldado por el código y los datos existentes.

GraphQL se basa en la consulta de campos de objetos. Estos objetos son el componente básico del esquema de GraphQL, que son representados por el tipo y sus respectivos campos, los cuales se pueden consultar del servicio. Por ejemplo, en el Código 3 se observa la declaración de un tipo para los domicilios.

```
type Cpi_domicilios {
  id_domicilio: ID!
  colonia: Cg_colonias!
  calle: Cg_calles!
  numero_exterior: String!
  numero_interior: String
  referencia_domicilio: String!
}
```

Código Fuente 3: *Declaración de un tipo con el esquema GraphQL*

Con base en el fragmento de código, se puede observar lo siguiente: **Cpi_domicilios** es un tipo de objeto de GraphQL que contiene algunos campos; **id_domicilio**, **colonia**, **calle**, **numero_exterior**, **numero_interior** y **referencia_domicilio** son campos que pertenecen al

tipo de Cpi_domicilios; **String** es uno de los tipos de datos nativos que maneja GraphQL; **String!** significa que el esquema de GraphQL asegura retornar un valor no nulo cuando se realice una consulta de este campo (esto es representado por el signo de admiración al final); y **Cg_colonias** y **Cg_calles** representan un tipo de objeto previamente declarado.

Por otra parte, los tipos de datos predefinidos dentro de GraphQL son: **Int**: entero de 32 bits con signo; **Float**: valor de punto flotante de doble precisión con signo; **String**: secuencia de caracteres UTF-8; **Boolean**: verdadero o falso; **ID**: representa un identificador único, a menudo utilizado para recuperar un objeto o clave de una caché. El tipo ID se serializa dependiendo del lenguaje, implementación o necesidades requeridas. Sin embargo, definirlo como ID significa que no se pretende que sea legible para las personas.

2.5.2.1. Consultas (Query)

Las consultas son la forma en la que se obtienen los recursos del servicio mediante la solicitud de los campos utilizados. Un ejemplo de la sintaxis se muestra en el Código 4, donde se solicita del tipo Cpi_domicilios, los campos: id_domicilio, id_calle, numero_exterior, numero_interior y la referencia_domicilio. Como se puede observar, no es necesario solicitar todos los campos de un objeto.

```
Cpi_domicilios {
  id_domicilio
  calle {
    id_calle
  }
  numero_exterior
  numero_interior
  referencia_domicilio
}
```

Código Fuente 4: *Consulta simple en GraphQL.*

2.5.2.2. Mutaciones (Mutation)

Al igual que las consultas, las mutaciones tienen las mismas características y sería posible realizar una operación de escritura dentro de esta. Sin embargo, por convención de GraphQL se debe realizar por medio de una mutación cualquier operación que modifique los datos. El Código 5 se utiliza para la creación de una calle y tiene los siguientes componentes: **mutation**, palabra reservada que indica que es una operación de escritura; **$nombre_calle**, variable de entrada no nulo de tipo cadena; **$nombre_corto**, variable de entrada que puede ser nulo de tipo cadena; y **CreateCalle**, nombre único de la mutación.

```
mutation(
    $nombre_calle: String!,
    $nombre_corto: String,
    $nombre_coloquial: String,
    $nota_ubicacion: String
){
    CreateCalle(
        input: {
            nombre_calle: $nombre_calle,
            nombre_corto: $nombre_corto,
            nombre_coloquial: $nombre_coloquial,
            nota_ubicacion: $nota_ubicacion
        }
    ){
        id_calle
        nombre_calle
        nombre_corto
        nombre_coloquial
        nota_ubicacion
    }
}
```

Código Fuente 5: *Mutación en GraphQL.*

2.6 Metodología ágil

De manera general, una metodología ágil de desarrollo de software promueve un proceso disciplinado de gestión de proyectos que fomenta la inspección y la adaptación frecuente, una filosofía de liderazgo que fomenta el trabajo en equipo, la organización y la responsabilidad. Todas estas son un conjunto de prácticas recomendadas de ingeniería destinadas a permitir la entrega rápida de software de alta calidad y un enfoque empresarial que alinea el desarrollo con las necesidades del cliente y los objetivos de la empresa.

Las etapas que conforman esta metodología; como se puede observar en la Figura 9, son: i) diagnóstico, ii) diseño, iii) desarrollo, iv) aseguramiento de la calidad, y v) desplegado de la plataforma.

Figura 9: Etapa de la matodología ágil de desarrollo de software.

- Diagnóstico. Se realizan sesiones de trabajo con los usuarios involucrados para definir el alcance del proyecto, identificar los requerimientos funcionales y no funciona-

les, para luego documentarlos en el reporte de Especificación de Requerimientos del Sistema (ERS).

- Diseño. En esta etapa se realizan las consideraciones respecto a como va a funcionar la plataforma, se toma en cuenta los diferentes factores como la red, el hardware, los casos de uso, la base de datos, el aspecto visual, entre otros.
- Desarrollo. En esta etapa se hace la programación de los prototipos de la aplicación, acordados en conjunto con el usuario en la etapa anterior.
- Aseguramiento de la calidad. Es el conjunto de actividades que validan la funcionalidad de la aplicación, asegurando que se cumplan los requisitos de calidad del producto software.
- Desplegado de la plataforma. La aplicación es puesta en producción y se toma en cuenta el soporte técnico y el mantenimiento de esta.

2.7 Trabajos relacionados

En la literatura actual se identificaron diferentes esfuerzos tecnológicos para el estudio, control, manejo y reducción de los residuos sólidos urbanos. Entre estos trabajos destacan:

- En el trabajo de (Quintero Fernandez, 2018) se realizó una aplicación web con Wix, que es una plataforma para el desarrollo web basado en la nube para la gestión de residuos sólidos no tóxicos dentro de la Universidad de la Costa en Colombia. La finalidad de este proyecto fue la difusión sobre el valor del reciclaje para contribuir a la gestión de residuos sólidos, esto se realizó mediante el mapeo de las ubicaciones de los cestos de basura dentro de las instalaciones de dicha universidad; el propósito fue impulsar nuevas iniciativas para la gestión de los residuos sólidos en sus espacios universitarios.
- En el trabajo de (Gutierrez Galicia et al., 2019) se exponen varios factores que involucran el manejo de los residuos sólidos urbanos en la Ciudad de México, la cual es una de las ciudades del país con mayor generación de RSU, y que tiene múltiples áreas de oportunidad en las que se pueden mejorar. Se resalta que el gobierno va atrasado en tomar acciones adecuadas para un adecuado manejo de los RSU en la ciudad. Se enfatiza en que se carece de datos confiables para conocer el porcentaje real de estos residuos que deben ser reciclados, y que tan solo el 32 % de los vehículos utilizados para la recolección y transporte de los RSU están en buenas condiciones.

- En el reporte realizado por (Hotspot, 2021) se observó el estado del manejo de los residuos sólidos en la Ciudad de México durante el 2021. Se describe la composición, el transporte y disposición de los diferentes tipos de residuos, al igual que la jerarquía de cómo está organizado este sector. Se discuten algunas iniciativas como El Sarape, que fue una planta de valorización térmica a gran escala, que por su alto costo de implementación fue abandonada. De igual manera, se destaca la poca inversión por parte del gobierno en el manejo de los RSU, esto debido a la pandemia por COVID-19. En este reporte se pone énfasis también en que en México se necesita incluir nuevas reformas para incluir al sector informal de recolección de residuos y así aumentar su efectividad; y, por último, se menciona que se requiere de soluciones tecnológicas adecuadas para una recolección efectiva de datos.

- En el trabajo de (Botello-Álvarez et al., 2018) se expone como en los países de Latinoamérica, que son en general países en desarrollo o en vías de desarrollo, cuentan con un gran sector informal para la recolección de basura, el cual tiene un gran impacto negativo en sus ecosistemas y propician el cambio climático que se vive hoy en día. México no es la excepción, por lo que, se expone que la gestión adecuada de los RSU es fundamental para que las empresas puedan utilizar materiales reciclados como el PET, papel, aluminio, entre otros. Sin embargo, los que se dedican son secciones de la población marginada.

La Tabla 2.2 resume las principales características de los trabajos identificados como parte de la revisión de la literatura.

Autor	**Aplicación**	**Método**	**Limitaciones**
Quintero Fernandez, 2018.	Diseño de un sistema de gestión de residuos institucionales SIGRI.	Mapeo de ubicaciones de disposición de residuos sólidos.	– Reducida población objetivo (ámbito universitario).
Gutierrez Galicia et al., 2019.	Análisis del estado de los RSU en la Ciudad de México.	Elección de prioridades para la mejora de la gestión de los RSU.	– Carencia de datos confiables.
Hotspot, 2021.	Estado del manejo de los RSU en la Ciudad de México y su impacto durante la pandemia por COVID-19.	Identificación de áreas de oportunidad para la implementación de reformas.	– Falta de soluciones tecnológicas adecuadas.
Botello-Álvarez et al., 2018.	Estado de países en vías de desarrollo para el manejo de RSU y el sector informal dedicado a esta actividad.	Reporte sobre la inclusión de los trabajadores informales de recolecta de RSU reciclables.	– Carencia de reformas para la gestión adecuada de RSU.

Cuadro 2.2: Trabajos relacionados.

Es evidente la preocupación por entender el impacto ambiental que está teniendo la inadecuada gestión de los residuos sólidos urbanos en la sociedad actual. Sin embargo, cuando la cantidad de residuos supera la capacidad de los procesos naturales, es cuando los ecosistemas empiezan a sufrir serios daños. De igual manera, en la Ciudad de México el acarreo y la gestión de los residuos sólidos urbanos deben tener regulaciones, que comprende un conjunto de requerimientos, licencias y trámites relacionados con la recolección, acopio, almacenamiento, manejo, transporte, tratamiento, reutilización, reciclaje y disposición final de los residuos sólidos urbanos.

2.8 Síntesis

En este capítulo se presentó el alcance que está teniendo la generación constante de residuos sólidos urbanos y el efecto invernadero que se produce como consecuencia de los gases que estos generan. Se mostraron estadísticas sobre la generación de los residuos sólidos urbanos en la Ciudad de México a lo largo del tiempo, identificando la composición mayoritaria de los RSU y las alcaldías con mayor generación. Además, se presentaron las normativas reguladoras vigentes en la Ciudad de México, como la licencia ambiental única, Ramir y el plan de manejo de residuos. También se describieron las características de una economía circular. Por otra parte, se dieron a conocer los espectros técnicos de la tecnología Web y las herramientas utilizadas en la actualidad para la construcción de este tipo de aplicaciones, basadas principalmente en arquitecturas cliente-servidor y de microservicios. Además, se describió la importancia de las APIs (interfaz de programación de aplicaciones) como parte de las funcionalidades y comunicación entre el cliente y el servidor.

CAPÍTULO 3

Propuesta de solución

En el capítulo anterior se presentaron las principales características de RSU, su impacto en el ambiente y la importancia de gestionar su disposición y promover una economía circular, donde los RSU deben ser aprovechados lo máximo posible, inculcando una cultura del reciclaje. Por otra parte, se introdujeron las diferentes normativas aplicables en la Ciudad de México para el control de los RSU, siendo estos: Licencia ambiental única, RAMIR y Plan de manejo de residuos. Por último, se presentaron las principales tecnologías utilizadas en el desarrollo de aplicaciones Web de la actualidad; así como la importancia de las APIs en la comunicación entre el cliente y el servidor de las aplicaciones Web y los trabajos relacionados.

En este capítulo se presenta el método utilizado como propuesta de solución en el desarrollo de la plataforma Web para el encadenamiento productivo de los RSU en la Ciudad de México. Por lo que, como método se emplearon las etapas de una metodología ágil de desarrollo de software, centrada en una construcción iterativa, en la que los requisitos y las soluciones evolucionan a través de la colaboración entre equipos. En este sentido, como solución se utilizaron las primeras tres etapas del desarrollo de software: i) diagnóstico, ii) diseño, y iii) desarrollo; dejando el aseguramiento de la calidad, y el desplegado de la plataforma Web para la discusión del Capítulo 4 (*Resultados*).

3.1 Diagnóstico

Para esta actividad se realizaron varias reuniones de trabajo en el periodo comprendido del 22 octubre de 2021 al 21 de febrero de 2022. Para esto, participaron los equipos de trabajo de la SECTEI (Secretaría de Educación, Ciencia, Tecnología e Innovación de la Ciudad de México), SEDEMA (Secretaría del Medio Ambiente de la Ciudad de México) y la Facultad de

Ingeniería de la Universidad Nacional Autónoma de México. Derivado de estas reuniones de trabajo, se identificaron los requerimientos: de usuario, funcionales y no funcionales, consideradas significativas para la construcción de la plataforma Web.

3.1.1. Requerimientos de usuarios

Se identificaron tres tipos de usuarios: administrador general, generador y prestador de servicios.

- *Administrador general*. Es el usuario SEDEMA con los permisos para acceder y administrar cualquier parte del sistema. Dar seguimiento a la información presentada en el manifiesto de entrega-recepción de los RSU.
- *Generador*. Persona física o moral que genera un promedio igual o superior a 50 kg diarios de residuos sólidos, los cuales puede ofertar o solicitar algún residuo o servicio a través de la plataforma Web.
- *Prestador de servicios*. Persona física o moral que publica sus requerimientos de residuos y servicios. Puede ofertar o solicitar servicios o residuos para las actividades del acopio, almacenamiento, reciclaje, reutilización, reúso y tratamiento de residuos; así como para la recolección y transporte. Debe contar con una autorización RAMIR vigente. Existen cuatro tipos de modalidades:

    a) *Recolección y transporte*. Recolecta los residuos y los transporta hacia sitios autorizados.
    b) *Acopio y almacenamiento*. Reúne y deposita temporalmente los residuos sólidos en un lugar determinado.
    c) *Reciclaje, reutilización, tratamiento o valorización*. Se dedica a la reutilización de los materiales contenidos en los residuos o para la reducción de estos, ya sea de manera directa o aplicando diversos procesos.
    d) *Disposición final*. Deposita los residuos en una ubicación final.

3.1.2. Requerimientos funcionales

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

1. Registro

- 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.
- Se debe contar con un registro de eventos para la atención de fallas.
- Se hará el registro de usuarios (Generadores y/o Prestadores de servicio) solicitando el número de registro regulatorio vigente (número de LAU, RAMIR o Planes de manejo), nombre del representante legal, razón social, ubicación, teléfono, correo electrónico y tipo de usuario.
- 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.
- La plataforma de encaminamiento 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 ante la SEDEMA.
- Se hará el registro, actualización y cancelación de oferta de material por el Generador y/o Prestador de servicio, donde indique el tipo de material, la cantidad y la ubicación geográfica.
- Se hará el registro, actualización y cancelación de servicio, donde se especifique la actividad que puede realizar conforme a sus permisos de registros ambientales.
- 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.
- Los residuos y servicios que son candidatos son únicamente los validados por la LAU (grandes generadores), Plan de manejo y RAMIR (prestadores de servicios).
- Una misma empresa registrada en la plataforma puede tener funciones de: Solicitante, Ofertante y Transportista.
- Las empresas que se registren como Prestadores de servicios deben de contar con la autorización RAMIR y/o Plan de manejo vigente, según aplique.
- La interfaz gráfica de usuario debe estar apegada al manual de identidad institucional del Gobierno de la Ciudad de México del periodo 2021-2024 (Gobierno de la Ciudad de México, 2021), previamente proporcionado.

2. Búsqueda

- Búsqueda de residuos en el registro de oferta.
- Solicitud del residuo que se desea adquirir/solicitar.
- Mostrar el tipo y cantidad del residuo ofertado.
- Mostrar la ubicación física del residuo ofertado.
- Mostrar el tiempo de oferta del residuo.

3. Solicitud

   - Se realizará la solicitud del pedido y se solicitará la confirmación por parte del ofertador.

4. Confirmación

   - 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

   - El administrador (usuario SEDEMA) solicita la generación del informe donde se muestre los usuarios que interactúan (empresas), la cantidad y el tipo de residuo o servicio, con información sobre la confirmación final, la ubicación de procedencia y el destino final, o el lugar donde se presta el servicio.
   - Para la confirmación y reporte de la interacción, se genera el manifiesto de Entrega-Recepción con la siguiente información: datos de la empresa generadora, residuos generados, datos de la empresa transportista, fecha de recolección, empresa solicitante, residuos recibidos y fecha de recepción.
   - Los administradores tendrán la posibilidad de descargar una base de datos con la información generada por los usuarios.
   - 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.
   - Alertas de correo electrónico a la SEDEMA:
     - Confirmación de usuarios registrados.
     - Confirmación de transacciones finalizadas (tipo de residuo y la cantidad).
     - Carga del manifiesto en el formulario y en formato digital en 30 días naturales posterior a la confirmación, en caso de que no se adjunten en este tiempo se emitirá una alerta.

        - Exceso del límite de residuo transportado diario (base 2019).
    - Alertas de correo electrónico a los usuarios:
        - Nuevas ofertas de productos.
        - Nuevas solicitudes de productos.
        - Nuevas ofertas de servicios de transporte y aprovechamiento.
        - Recordatorio a los 20 días de realizar una transacción para adjuntar manifiesto físico.
    - Generación de estadísticas mediante filtros de los productos generados conforme a determinados periodos para la elaboración de reportes.
    - Manual para cada tipo de usuario.

3.1.3. Requerimientos no funcionales

1. Capacidad para 5000 usuarios iniciales con opción a escalabilidad.
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 residuos sólidos 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 Web debe incluir un sistema de bases de datos.
7. Se emplearán como herramientas de desarrollo: Go, Flutter, Dart, Docker y PostgreSQL, entre otras.

Con base en la identificación de los requerimientos presentados, en el que participaron los equipos de trabajo de la SECTEI, SEDEMA y la Facultad de Ingeniería de la UNAM, se hizo una descomposición de tareas a nivel del equipo de desarrollo, 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 funcionalidades del lado del servidor (Back-End).

Para el caso particular de este trabajo de tesis, se asignó la codificación y pruebas de funcionalidad de operadores CRUD (Create, Read, Update and Delete) en forma de APIs de

algunos catálogos de la base de datos a través de GraphQL (generales, de referencia, de interfaz, y de procesos); la elaboración de diagramas de casos de uso con el objetivo de identificar y organizar los procesos de la aplicación desde el punto de vista del usuario; así como los diagramas de secuencia correspondientes y diagramas de flujo.

3.2 Diseño

En el stack tecnológico para el desarrollo de la plataforma Web se consideró dos capas: Back-End y Front-End. Una capa inferior, del lado del servidor (Back-End), donde estará el sistema operativo, ya sea Linux o Windows. Dado que al trabajar con los contenedores de Docker, la plataforma no será dependiente del sistema operativo subyacente. Además, para la gestión de la base de datos se utilizó PostgreSQL y como lenguaje de programación Go (Figura 10). Mientras que en la segunda capa, del lado del cliente (Figura 11), se utilizaron como lenguajes de programación: Dart y JavaScript; así como el framework Flutter, Scripts y hojas de estilo HTML5 y CSS.

Figura 10: Tecnología del lado del servidor (Back-End).

Figura 11: Tecnología del lado del cliente (Front-End).

La ventaja de utilizar Flutter es que permite la creación de interfaces prácticamente para cualquier dispositivo con la misma base de código. Pudiendo después extender las funcionalidades de la plataforma a dispositivos basados en Android, iOS o de escritorio.

3.2.1. Herramientas de desarrollo utilizadas

a) GoLand

GoLand es un entorno de desarrollo integrado (IDE) que ofrece una amplia gama de características para la creación de soluciones eficientes con Go. Con este editor se simplifica el proceso de codificación de Go gracias a la detección de errores y a las sugerencias en tiempo real. Este IDE también cuenta con diversas herramientas integradas en la aplicación que permite ejecutar y depurar las aplicaciones de Go, dentro de las más relevantes se encuentra el soporte para Git esencial para el desarrollo colaborativo. La Figura 12 muestra, a modo de ejemplo, la vista del IDE de GoLand.

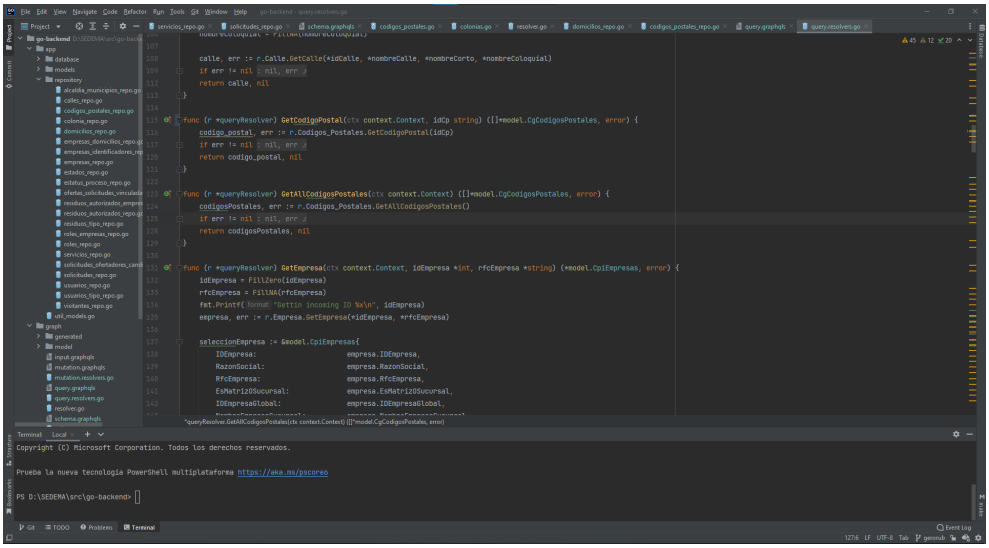

Figura 12: Vista del IDE de GoLand.

b) Visual Studio Code

Visual Studio Code (VS Code) es un editor de texto gratuito de código abierto de Microsoft. VS Code está disponible para Windows, Linux y macOS, y cuenta con soporte para operaciones de desarrollo, como: depuración, ejecución de tareas y control de versiones. Cuenta con soporte para varios lenguajes de programación y adicionalmente cuenta con una amplia variedad de plugins creados por la comunidad de desarrollo. La Figura 13 muestra la vista del IDE de Visual Studio Code.

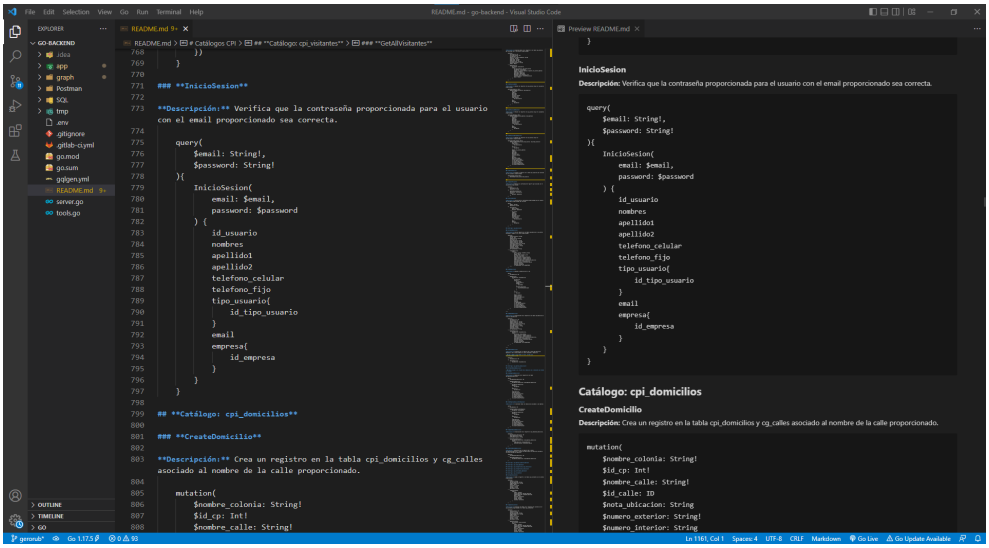

Figura 13: Vista del IDE de Visual Studio Code.

c) PSQL y pgAdmin 4

PSQL es un terminal interactivo para ejecutar consultas de PostgreSQL y examinar los resultados. PSQL también tiene una serie de metacomandos y capacidades similares a las de la línea de comandos que facilitan la construcción de secuencia y automatización de una serie de actividades en la base de datos. Las consultas de igual manera pueden provenir desde un archivo o los argumentos en la línea de comandos. La Figura 14 muestra la vista de PSQL.

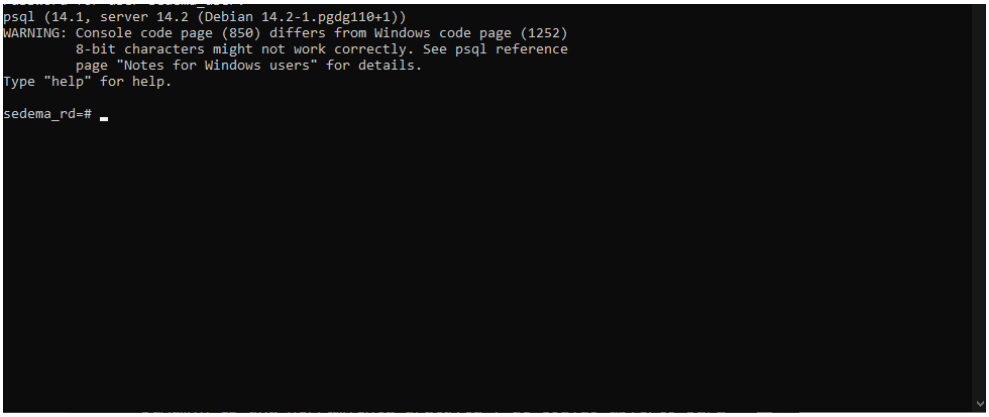

Figura 14: Vista de PSQL.

Por otro lado, pgAdmin es una herramienta gratuita y de código abierto para usar con bases de datos en PostgreSQL de manera local, en la nube, en contenedores o en cualquier otro ambiente mediante una herramienta de administración gráfica para hacer fácil la manipulación del esquema y los datos. La Figura 15 muestra la vista de pgAdmin4.

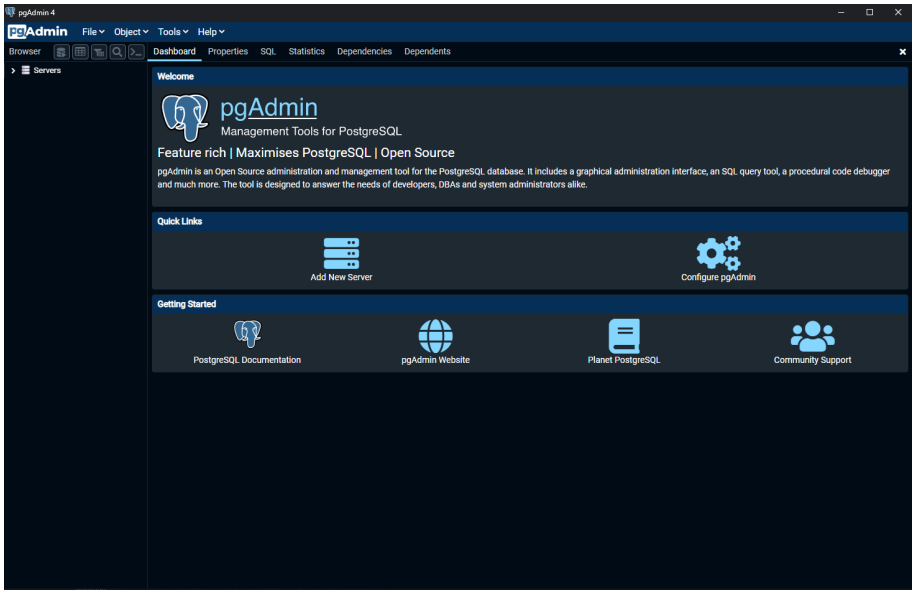

Figura 15: Vista de pgAdmin4.

d) Postman

Para comprobar el funcionamiento de las consultas y mutaciones asociadas a los catálogos desarrollados, se hicieron pruebas en Postman, la cual es una herramienta para diseñar, construir, probar y modificar APIs. Una de las ventajas que tiene Postman es su capacidad de crear colecciones y distintos ambientes de pruebas. Además, ayuda a optimizar el tiempo de ejecución de las pruebas. En este sentido, la Figura muestra la vista de Postman 16.

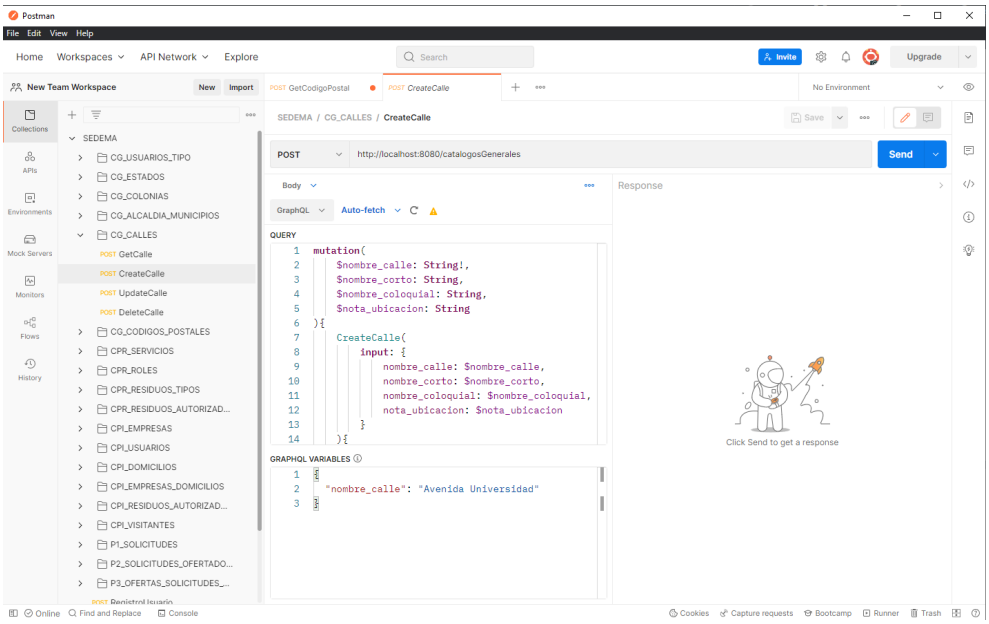

Figura 16: Vista de Postman.

e) Git y Gitlab

Git es un sistema de control de versiones distribuido, gratuito y de código abierto, diseñado para gestionar proyectos con rapidez y eficacia. Git hace un seguimiento de los cambios que se realizan en los archivos, de modo que se tiene un registro de lo que se ha hecho, y se puede volver a versiones específicas en caso de necesitar. Git también facilita la colaboración, permitiendo que los cambios realizados por varias personas se fusionen en una sola fuente.

Por lo tanto, GitLab es un repositorio Git basado en la web que proporciona repositorios abiertos y privados gratuitos, permitiendo a los equipos colaborar y construir un mejor software, reduciendo los ciclos de vida de los productos y aumentando la productividad, lo que a su vez crea valor para los clientes. La aplicación no requiere que los usuarios gestionen autorizaciones para cada herramienta. Si los permisos se establecen una vez, todos los miembros de la organización tienen acceso a cada componente. La Figura 17 muestra la vista de GitLab.

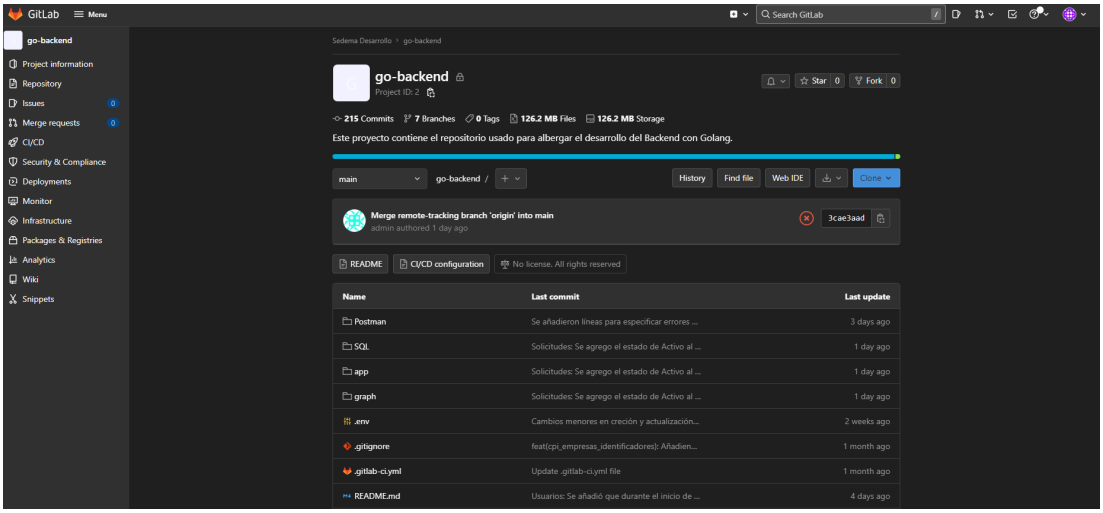

Figura 17: Vista de Gitlab.

3.2.2. Flujo general

Por otra parte, como parte del diseño se elaboró también una estructura del flujo general que tendrá el usuario al ingresar a la plataforma Web. Para esto, a través de un diagrama de flujo se estableció el proceso y operaciones de utilidad para el usuario. La Figura 18 muestra el flujo en el cual el usuario trata de ingresar a la plataforma. En primera instancia, se requerirá de un inicio de sesión, en caso de no estar registrado, el usuario tendrá la oportunidad de registrarse si cuenta con un instrumento regulatorio vigente, según corresponda. En caso de no tener un instrumento regulatorio, el usuario podrá ingresar a la plataforma en calidad de visitante, donde únicamente podrá visualizar los residuos y servicios que se oferten. Por

lo tanto, dado el caso en que el usuario inicie sesión, podrá realizar alguna de las subrutinas que abarcan las opciones de ofertar o solicitar servicios o residuos.

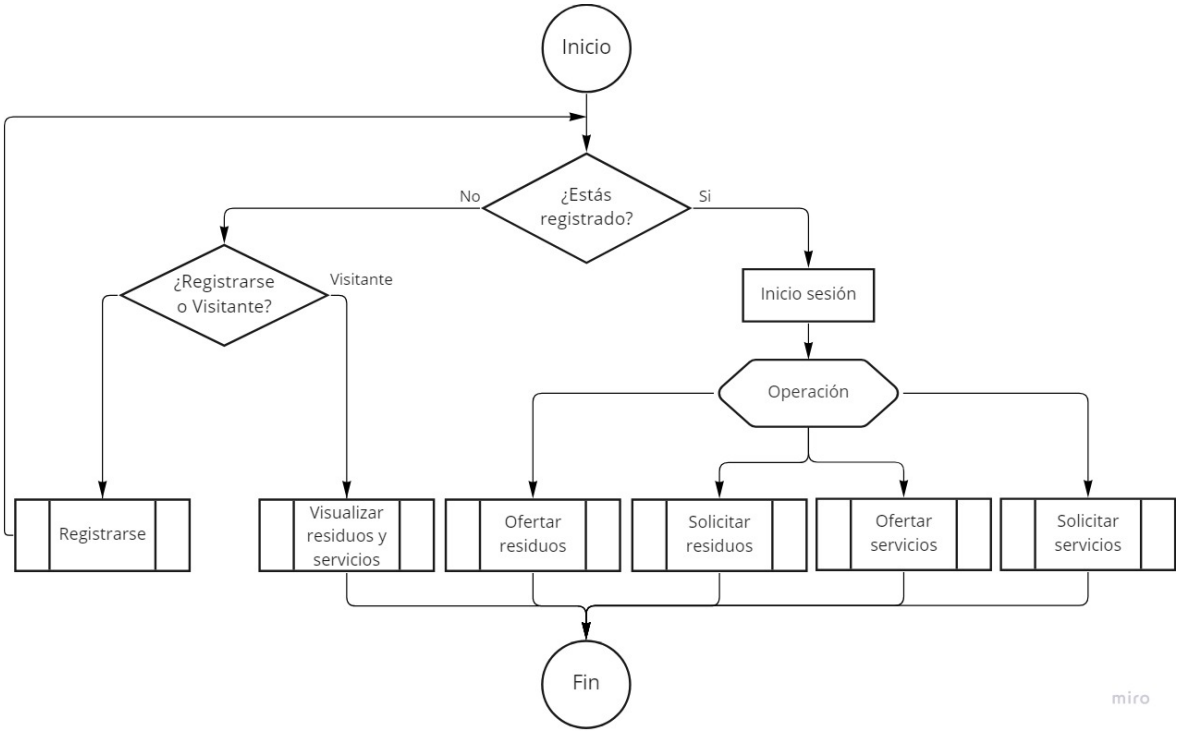

Figura 18: Diagrama del flujo general de la plataforma Web.

La Figura 19 corresponde a la subrutina *Ofertar residuos*, en la cual un usuario puede crear una oferta de un residuo y esperar la solicitud de posibles candidatos para continuar con el proceso de vinculación. Una vez elegido el candidato se prosigue a seleccionar el prestador de servicios para la recolección y transporte del residuo. Para esto puede ser el propio Ofertante, en caso de tener el permiso correspondiente, o mediante la búsqueda de otro para realizar el trabajo. Una vez elegido el prestador de servicios se realiza la vinculación y la subrutina termina.

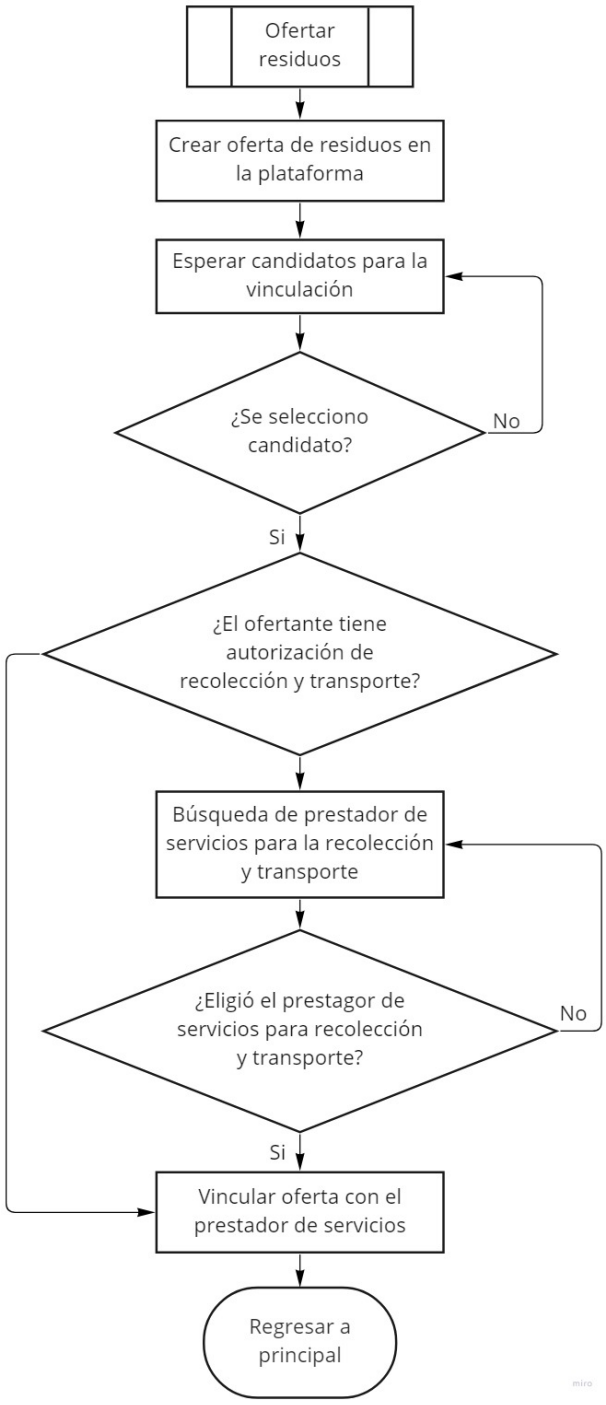

Figura 19: Diagrama de la subrutina *Ofertar residuos.*

3.2.3. Casos de uso comunes

Como parte del diseño, se realizaron diferentes casos de uso con el objetivo de identificar y organizar los procesos de la aplicación desde el punto de vista del usuario y el flujo que se tendrá para la oferta y solicitud de residuos y servicios. Adicionalmente, mediante estos casos de uso se elaboraron los diagramas de secuencia correspondientes.

Para el desarrollo de los casos de uso, se contempló al usuario *Prestador de servicios* a toda empresa que tiene el permiso correspondiente para desarrollar las siguientes actividades:

- Recolección y transporte.
- Acopio y almacenamiento.
- Reciclaje.
- Reutilización.
- Tratamiento o valorización.
- Disposición final.

Caso 1: Generador oferta residuo

Actores participantes:

1. Generador (Empresa A).
2. Prestador de servicios para la recolección y transporte (Empresa B).
3. Prestador de servicios (Empresa C).

Descripción:

1. Empresa A genera y oferta un residuo.
2. Empresa C solicita en la plataforma Web el residuo.
3. Empresa A contrata a un transportista (Empresa B) que lleve el residuo a la Empresa C.
4. Se cierra vinculación.

Figura 20: Caso de uso común sobre el generador que oferta residuos.

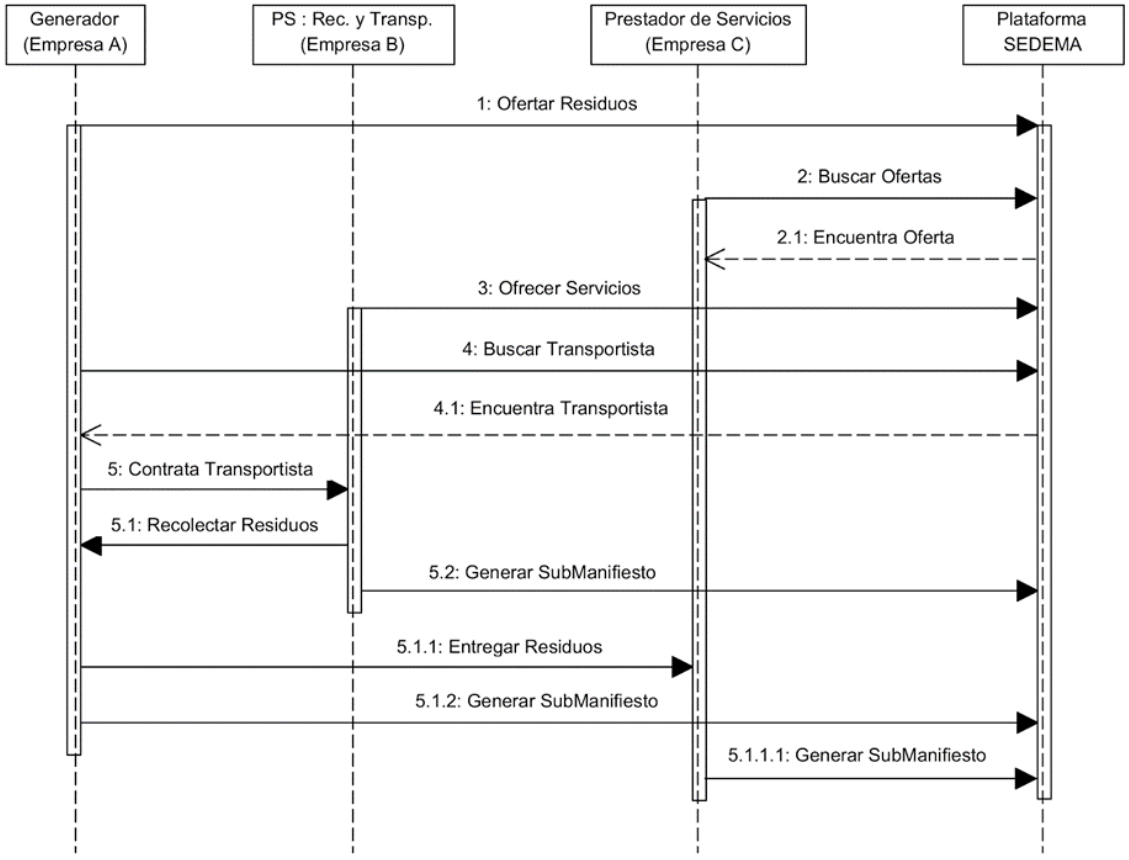

Figura 21: Diagrama de secuencia para el caso común 1.

Caso 2: Prestador de servicios oferta residuo

Actores participantes:

1. Prestador de servicios (Empresa A).
2. Prestador de servicios para la recolección y transporte (en caso de que la Empresa A no se encuentre autorizado para la recolección y transporte) (Empresa B).
3. Actor que requiere el residuo (generador u otro prestador de servicios) (Empresa C).

Descripción:

1. Centro de acopio de residuos tecnológicos (Empresa A) oferta 100 kg de residuos plásticos.
2. Empresa C compra el residuo. El centro de acopio también es prestador de servicios de transporte (Empresa A y B), por lo tanto, no tiene que contratar a un transportista.
3. Entrega a la Empresa C.
4. Se cierra vinculación.
5. El manifiesto debe ser llenado en todas sus partes, aunque el nombre del generador y transportista sea el mismo.

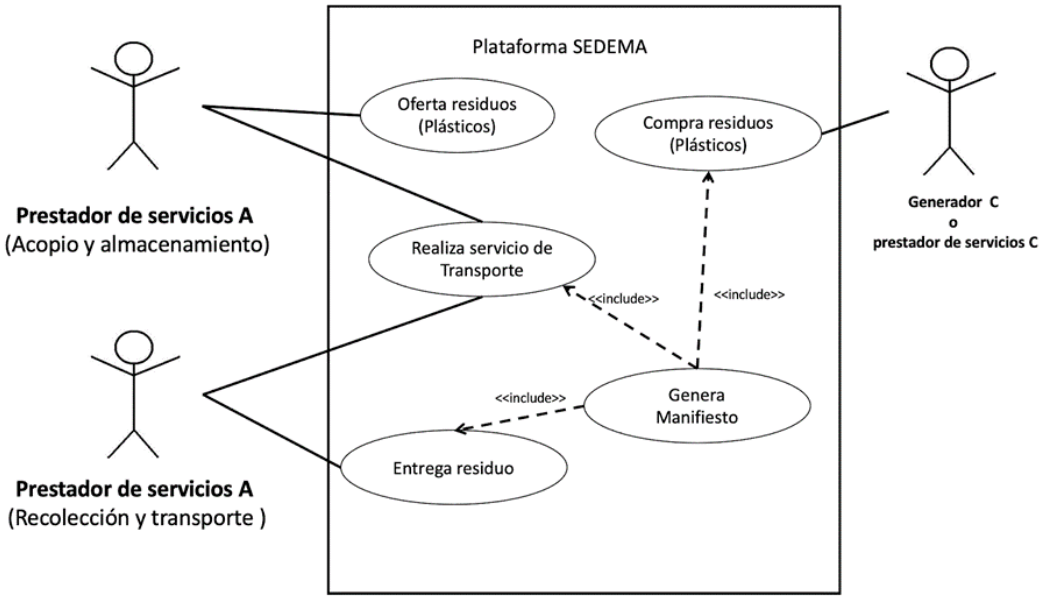

Figura 22: Caso de uso común sobre el prestador de servicios que oferta residuos.

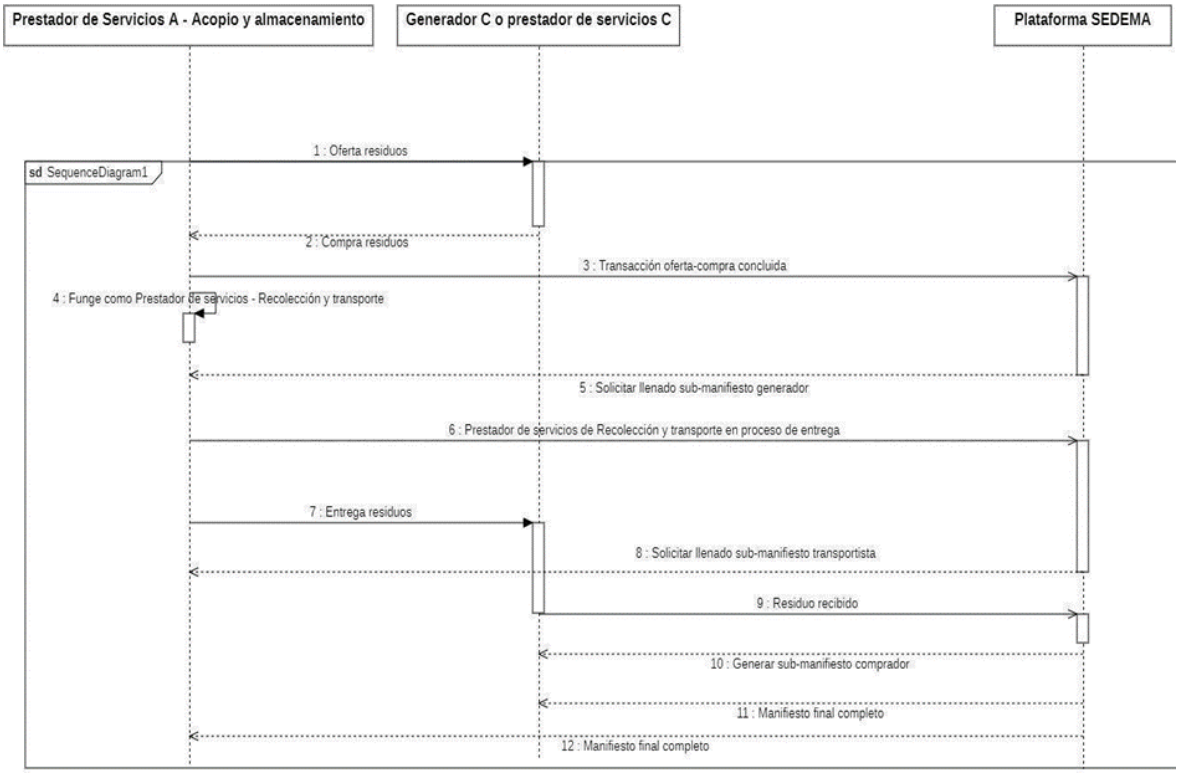

Figura 23: Diagrama de secuencia para el caso común 2.

Caso 3: Prestador de servicios oferta un servicio

Actores participantes:

1. Prestador de servicios (Empresa A).
2. Prestador de servicios para la recolección y transporte (en caso de la Empresa A o C no se encuentren autorizados para la recolección y transporte) (Empresa B).
3. Actor que requiera el servicio (generador u otro prestador de servicios) (Empresa C).

Descripción:

1. Centro de acopio de residuos tecnológicos (Empresa A) oferta 100 kg de residuos plásticos.
2. Empresa C compra el residuo. El centro de acopio también es prestador de servicio de transporte (Empresa A y B), por lo tanto, no tiene que contratar a un transportista.
3. Entrega a la Empresa C.
4. Se cierra vinculación.

5. El manifiesto debe ser llenado en todas sus partes, aunque el nombre del generador y transportista sea el mismo.

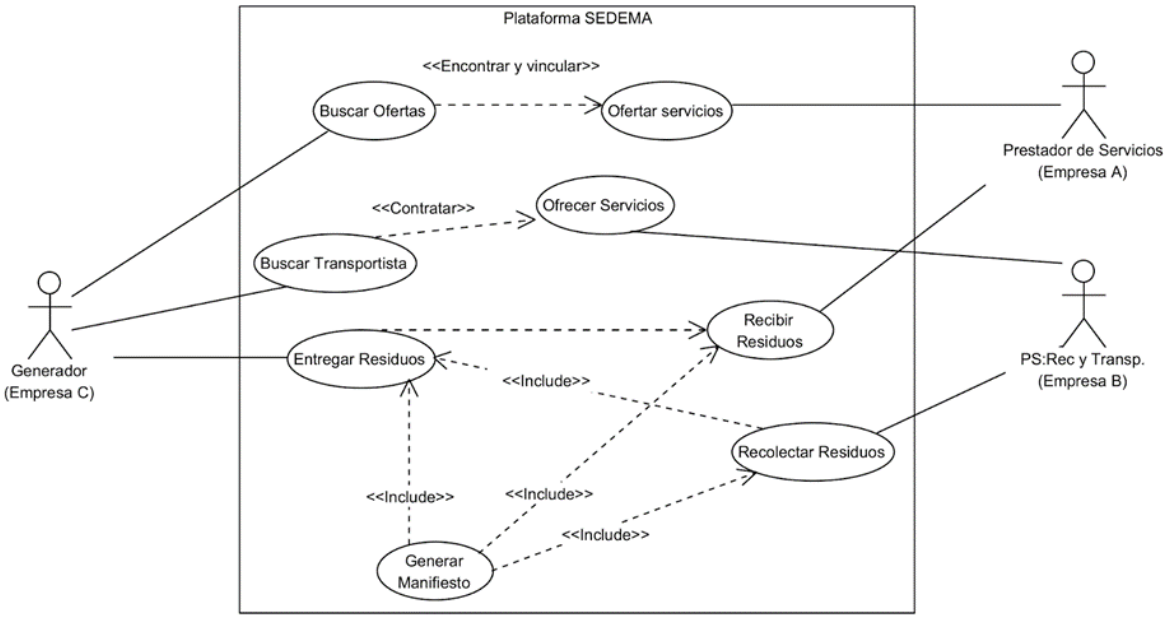

Figura 24: Caso de uso común sobre el prestador de servicios que oferta un servicio.

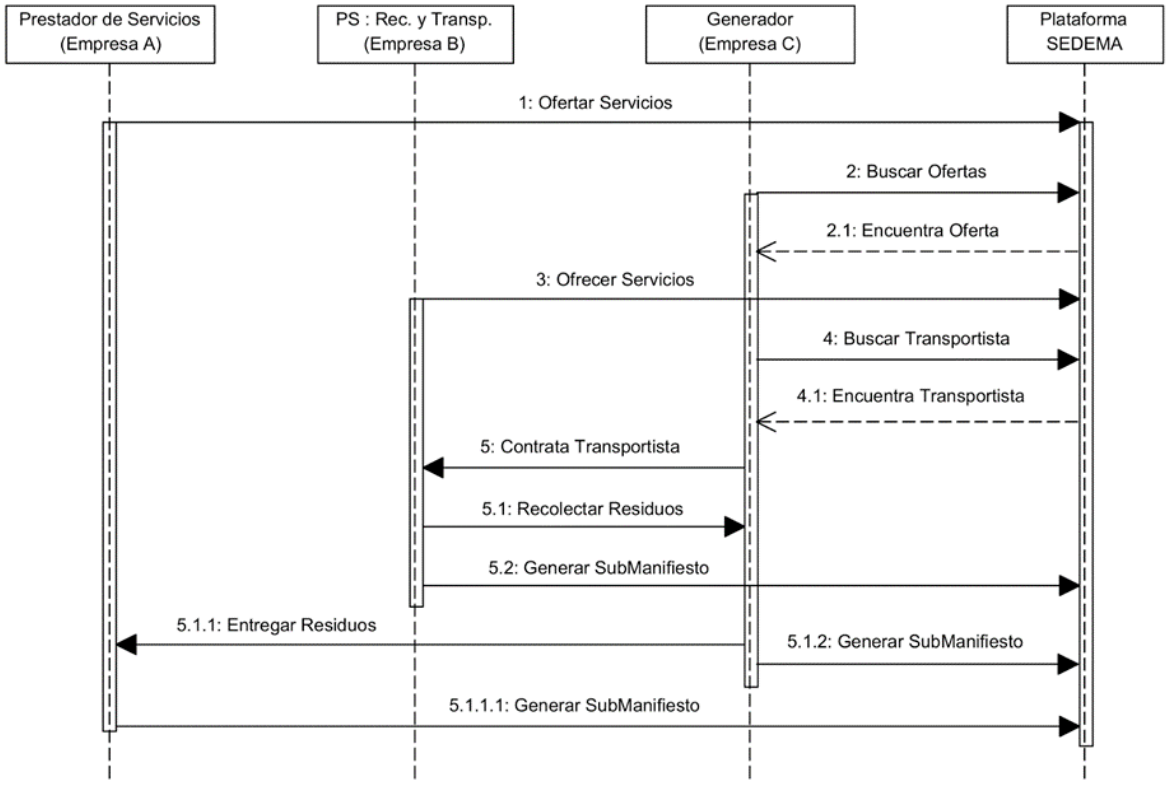

Figura 25: Diagrama de secuencia para el caso común 3.

Caso 4.1: Generador solicita residuo

Actores participantes:

1. Generador (Empresa A).
2. Prestador de servicios para la recolección y transporte (en caso de que la Empresa C no se encuentre autorizada para la recolección y transporte) (Empresa B).
3. Prestador de servicios (Empresa C).

Descripción:

1. Una empresa se dedica a fabricar jabones (Empresa A).
2. Busca ofertas en la plataforma Web el residuo "grasas y aceites"
3. Encuentra prestadores de servicio que realizan el acopio y almacenamiento de grasas y aceites (Empresa C).
4. Empresa A busca en la plataforma un transportista (Empresa B).
5. La Empresa B recolecta los residuos de la Empresa C para su posterior entrega a la Empresa A.
6. Se cierra vinculación.

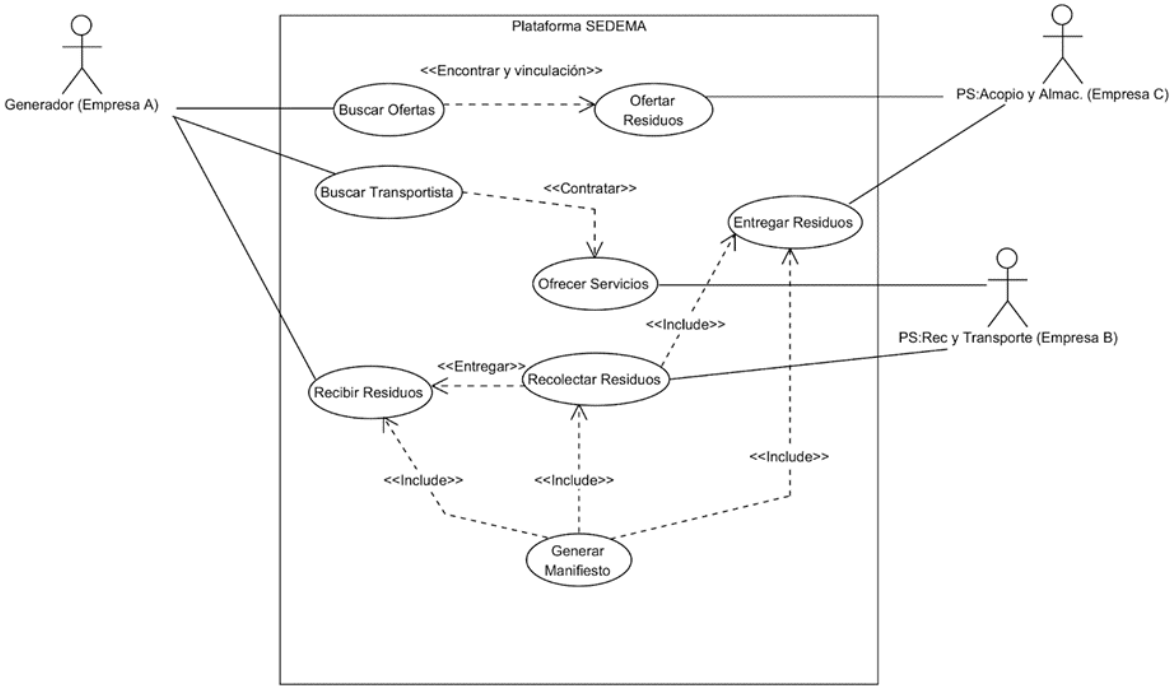

Figura 26: Caso de uso común sobre el generador que solicita residuos -4.1-.

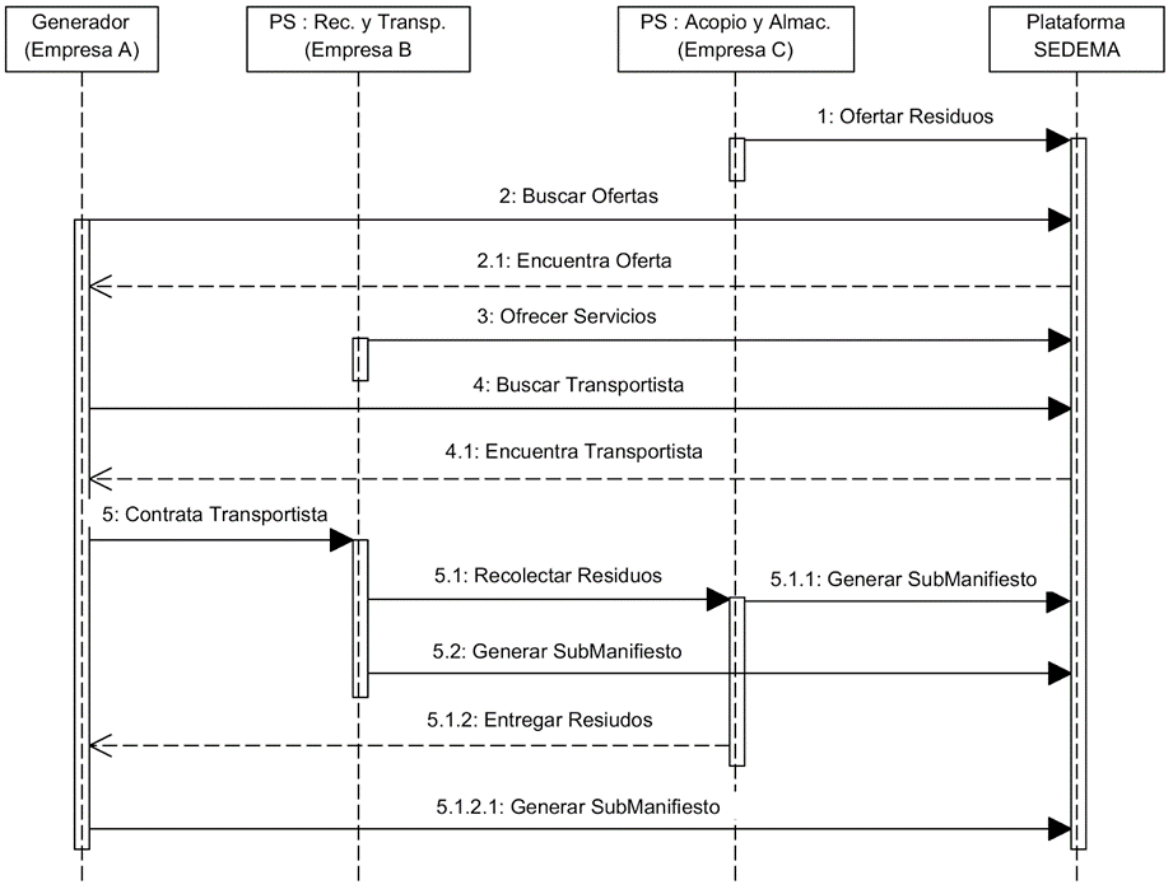

Figura 27: Diagrama de secuencia para el caso común 4.1.

Caso 4.2: Generador solicita residuo

Actores participantes:

1. Generador 1 (Empresa A).
2. Prestador de servicios para la recolección y transporte (Empresa B).
3. Generador 2 (Empresa C).

Descripción:

1. Empresa que se dedica a la crianza de ganado (Empresa A) busca ofertas en la plataforma Web el residuo "bagazo"de una empresa productora de alimentos (Empresa C).
2. Selecciona la oferta que le interesa con base en la ubicación.
3. La empresa productora de alimentos (Empresa C) busca en la plataforma Web el servicio del transportista (Empresa B) para entregar el residuo a la empresa de crianza de ganado (Empresa A).

4. Se contrata el transporte por la Empresa B.
5. Se cierra vinculación.

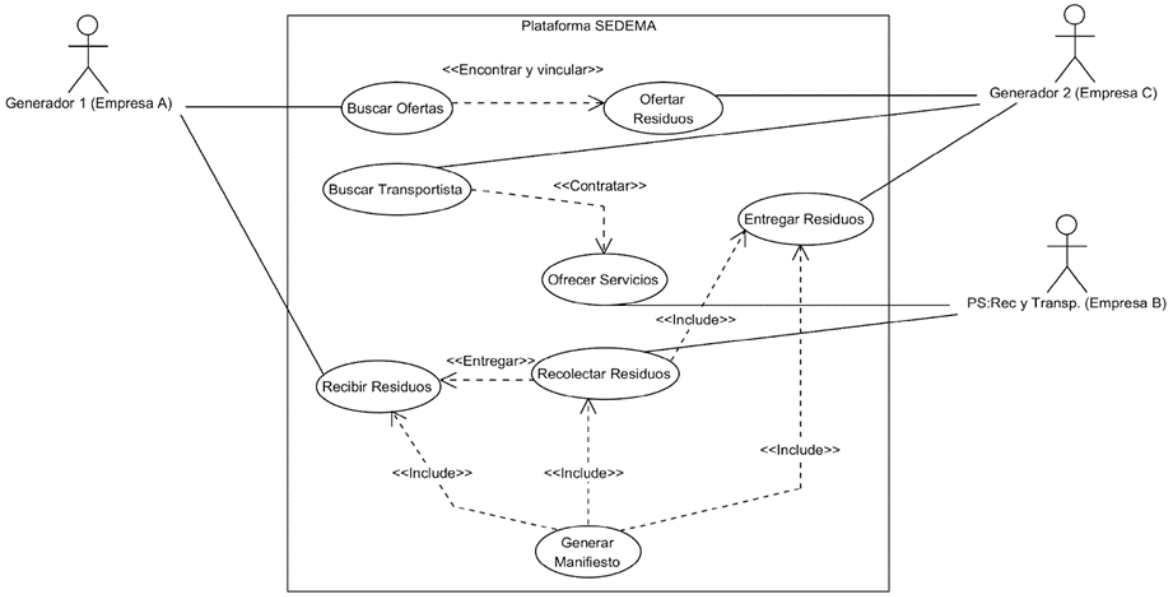

Figura 28: Caso de uso común sobre el generador que solicita residuos -4.2-.

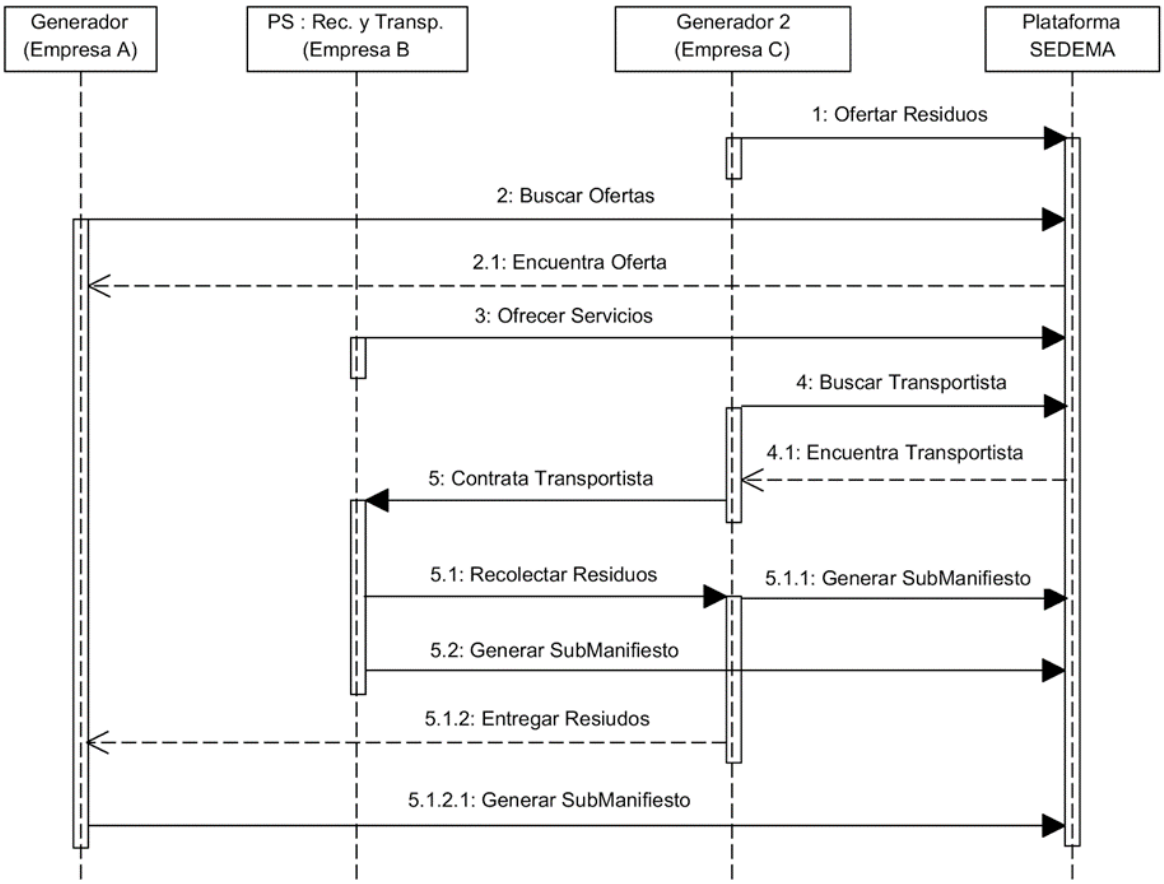

Figura 29: Diagrama de secuencia para el caso común 4.2.

Caso 5: Generador solicita residuo

Actores participantes:

1. Prestador de servicios 1* (Empresa A1).
2. Prestador de servicios 2* (Empresa A2).
3. Prestador de servicios para la recolección y transporte (en caso de que el actor anterior no se encuentre autorizado para la recolección y transporte) (Empresa B).
4. Actor que tenga en oferta el residuo (generador u otro prestador de servicios) (Empresa C).

Descripción:

1. La Empresa C genera y oferta residuos de papel, alimentos, grasas y aceites.
2. A la Empresa A1 le interesa los residuos de papel.
3. A la Empresa A2 le interesa residuos de alimentos y de grasas y aceite.
4. La Empresa C contrata al Transportista (Empresa B) para las Empresas A1 y A2.
5. Se cierra vinculación con Empresa A1, generando su manifiesto considerando únicamente el residuo de papel.
6. Se cierra vinculación con Empresa A2, generando su manifiesto por ambos residuos (alimentos y grasas y aceites).

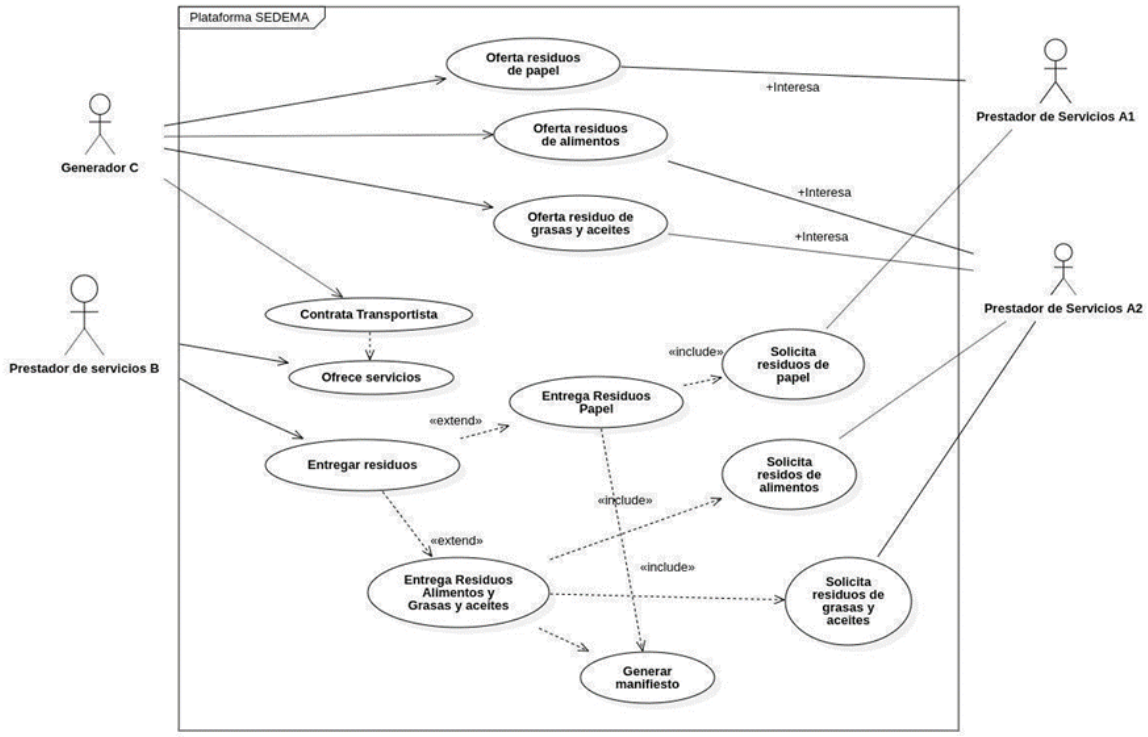

Figura 30: Caso de uso común sobre el generador que solicita residuos.

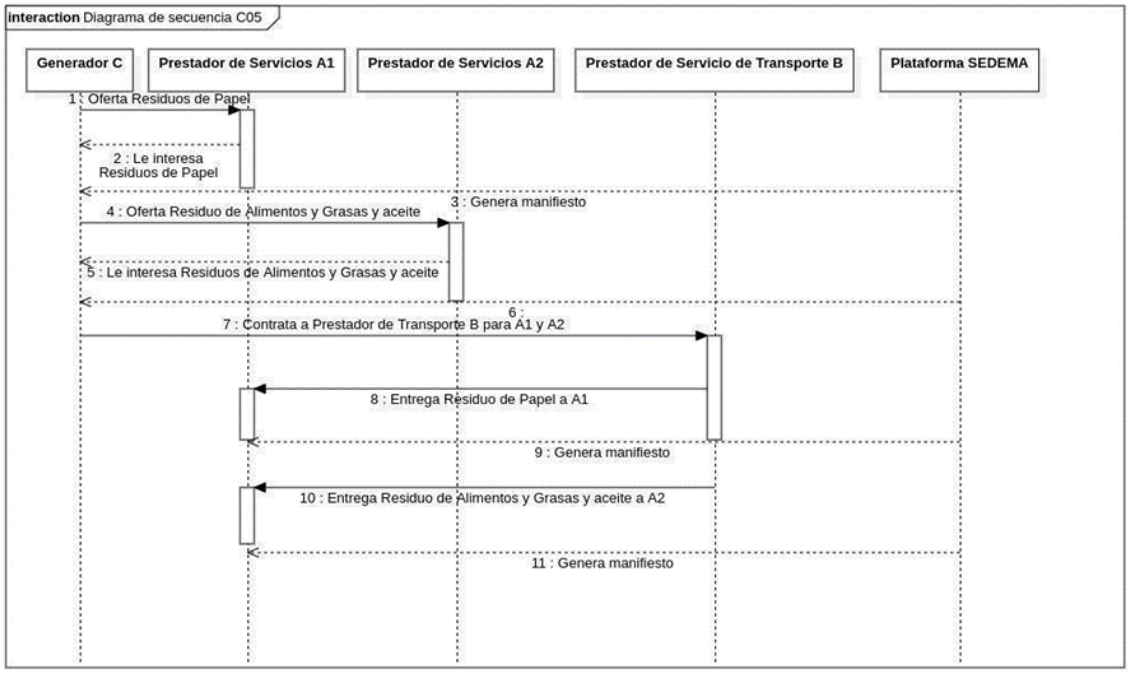

Figura 31: Diagrama de secuencia para el caso común 5.

3.2.4. Casos de uso especiales

Caso 1: Vinculación incompleta

Actores participantes:

1. Generador (Empresa A).
2. Prestador de servicios para la recolección y transporte (Empresa B).

Descripción:

1. Debido a que la Empresa A únicamente contacta el servicio de la Empresa B, no se considera como una vinculación completa. No hay necesidad de llenar el manifiesto de entrega-recepción, ya que solo entró a consultar las ofertas y demandas.

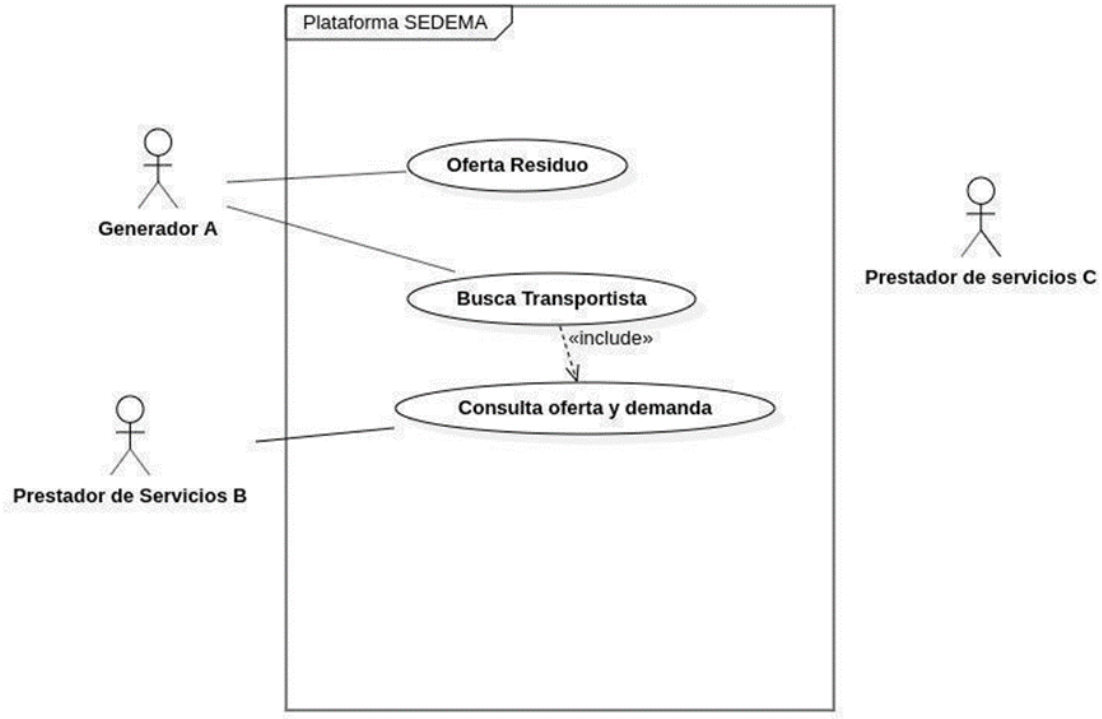

Figura 32: Caso de uso especial 1 (vinculación incompleta).

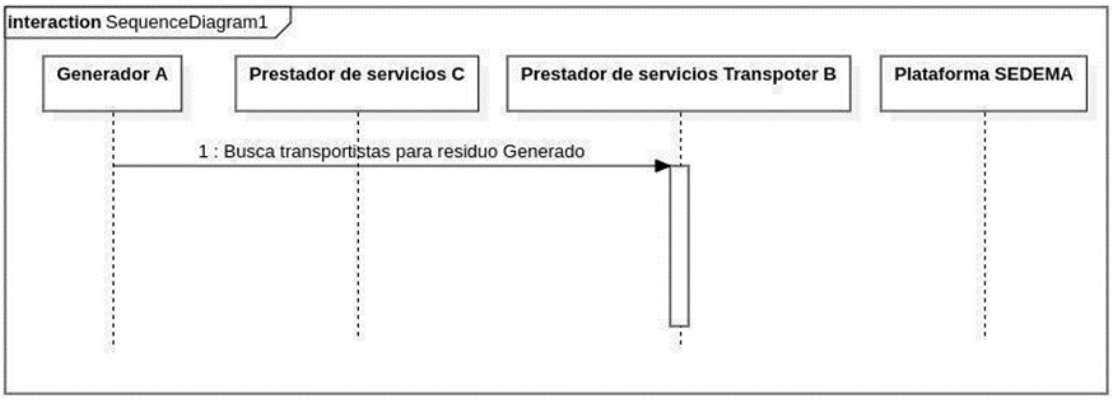

Figura 33: Diagrama de secuencia para el caso especial 1.

Caso 2: Múltiples envíos

Actores participantes:

1. Generador (Empresa A).
2. Prestador de servicios para la recolección y transporte (Empresa B).
3. Prestador de servicios 1* (Empresa C1).
4. Prestador de servicios 2* (Empresa C2).

Descripción:

1. Empresa A genera y oferta residuos de papel, alimentos, grasas y aceites.
2. A la Empresa C1 le interesa los residuos de papel.
3. A la Empresa C2 le interesa residuos de alimentos y de grasas y aceite.
4. La Empresa A contrata a la Empresa B para las Empresas C1 y C2.
5. Se cierra vinculación con Empresa C1, generando su manifiesto, considerando únicamente el residuo de papel
6. Se cierra vinculación con Empresa C2, generando su manifiesto, considerando ambos residuos (alimentos y de grasas y aceite).
7. Se generan dos manifiestos.

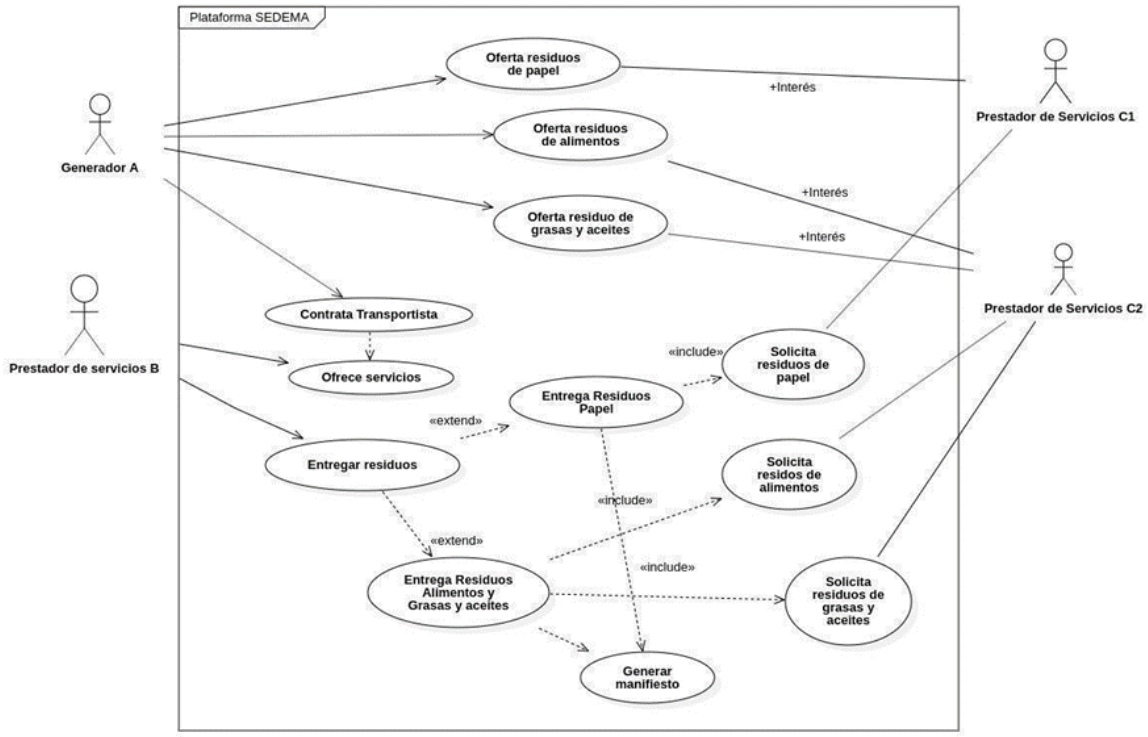

Figura 34: Caso de uso especial 2 (múltiples envíos).

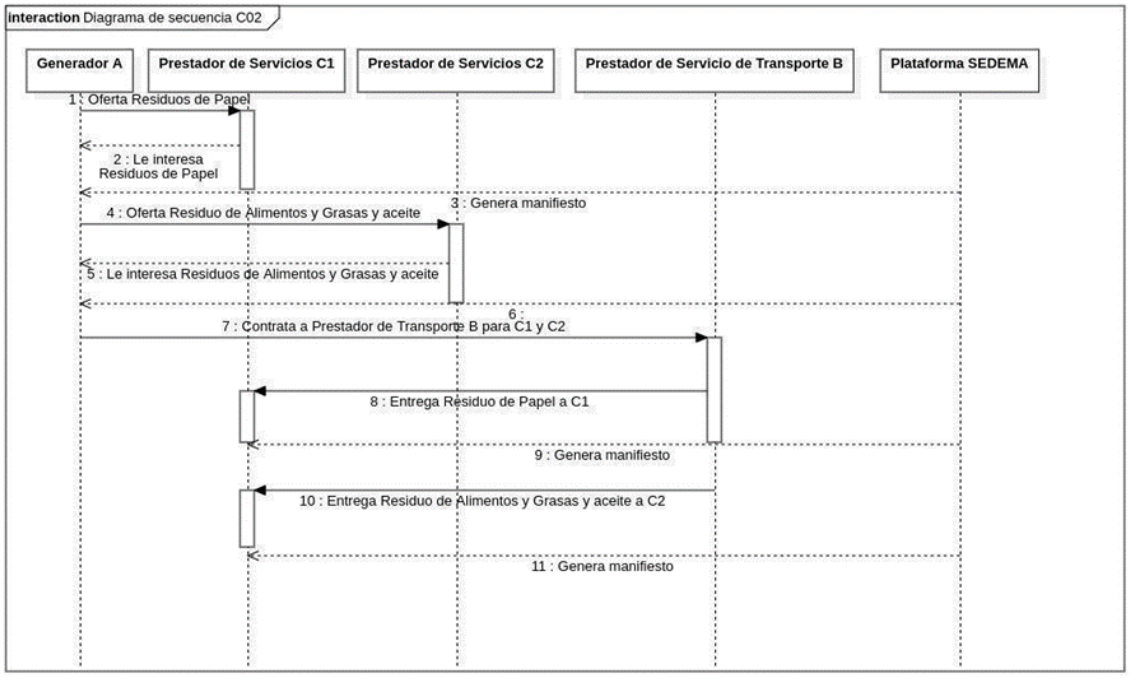

Figura 35: Diagrama de secuencia para el caso especial 2.

Caso 3: Anexos de residuos

Actores participantes:

1. No aplica.

Descripción:

1. Se pueden presentar situaciones en las que en la recolección de electrónicos, al manifiesto de entrega-recepción se le anexan listados de los equipos recolectados, que en la práctica pueden llegar a ser más de un artículo.

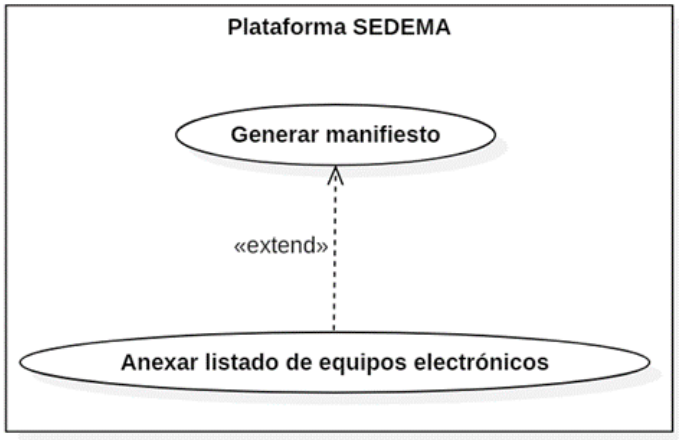

Figura 36: Caso de uso especial 3 (anexos de residuos).

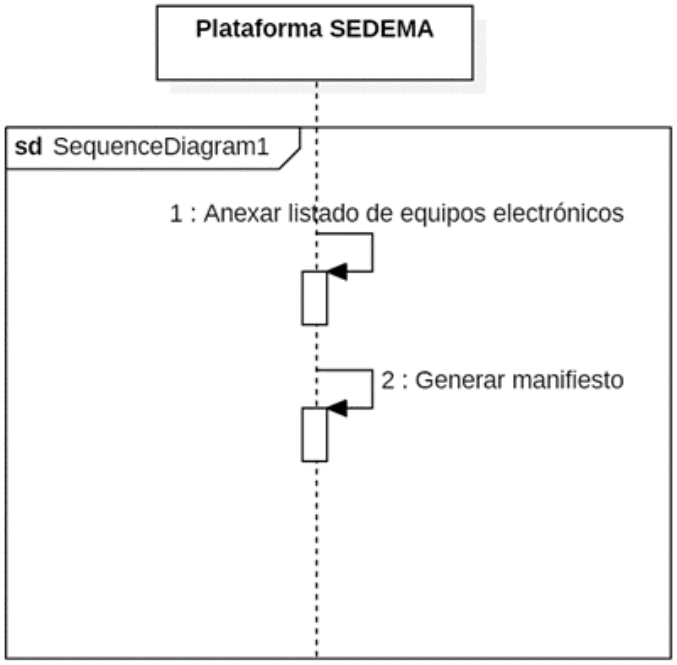

Figura 37: Diagrama de secuencia para el caso especial 3.

Caso 4: Múltiples vinculaciones

Actores participantes:

1. Generador 1 (Empresa A1).
2. Prestador de servicios (Empresa C).
3. Generador 2 (Empresa A2).
4. Prestador de servicios para la recolección y transporte (Empresa B).

Descripción:

1. Empresa A1 genera y oferta residuos eléctricos y electrónicos.
2. Empresa C da tratamiento a residuos eléctricos y electrónicos para la obtención de cobre.
3. Empresa A2 le interesa residuos de cobre.
4. Empresa A1 contrata a un transportista (Empresa B) para que recolecte su residuo y lo lleve a la Empresa C para su tratamiento.
5. Se cierra la vinculación con Empresa C, generando su manifiesto.
6. Empresa A2 solicita a Empresa C los residuos de cobre derivados del tratamiento.
7. Empresa C contrata a Empresa B para que recolecte los residuos y los lleve a Empresa A2.
8. Se cierra la vinculación con Empresa A2, generando su manifiesto.
9. Se generan dos manifiestos. Considerar que para la valorización de un residuo se pueden dar una cantidad "n"de tratamientos. Los manifiestos se cierran siempre que el residuo se entregue a un destino y participe un transporte autorizado.

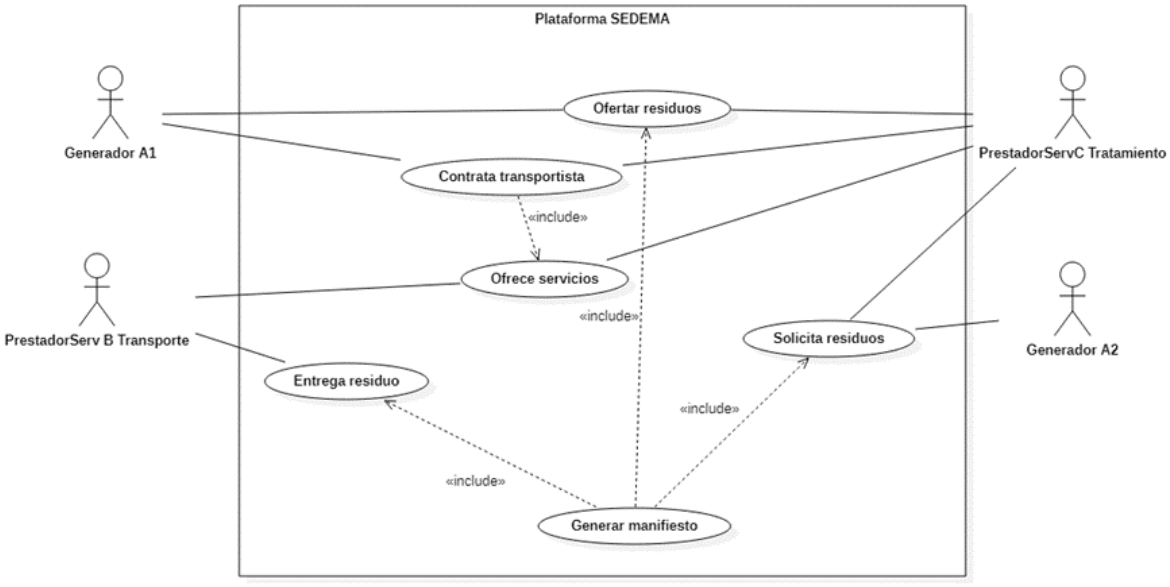

Figura 38: Caso de uso especial 4 (múltiples vinculaciones).

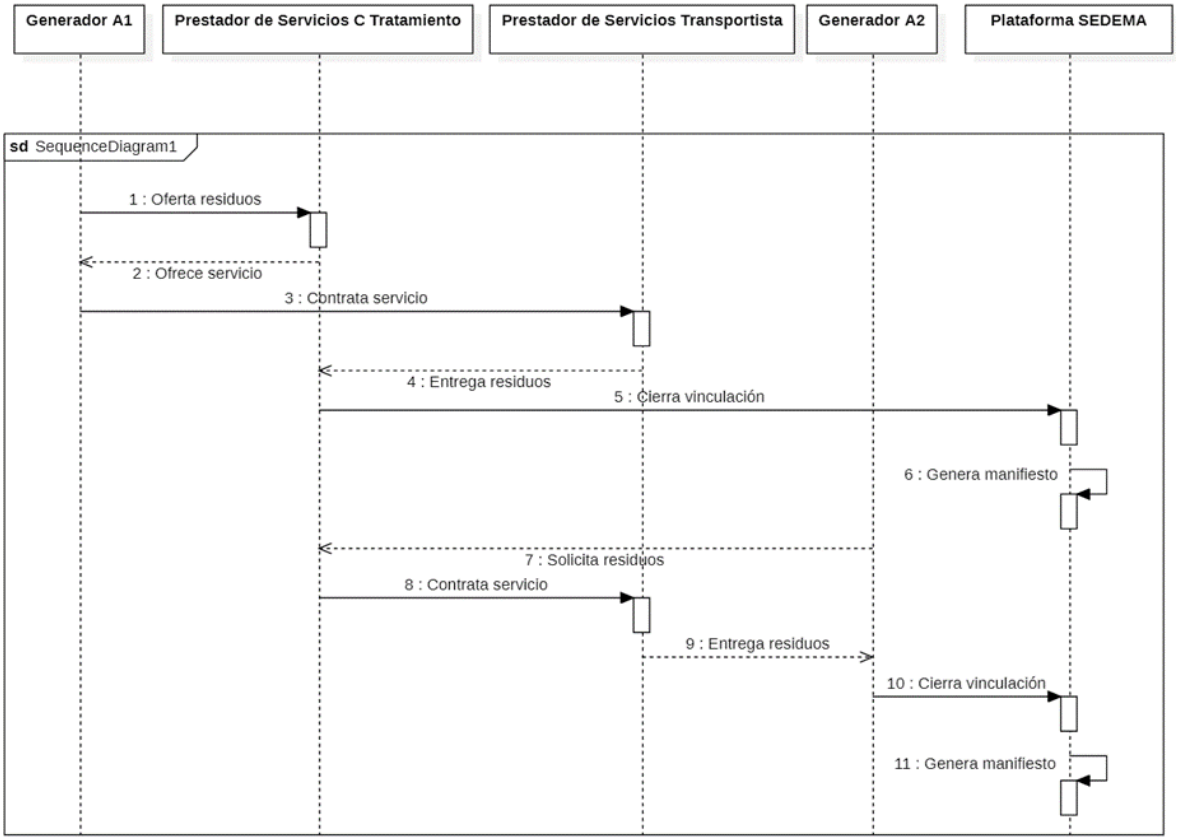

Figura 39: Diagrama de secuencia para el caso especial 4.

3.3 Desarrollo

3.3.1. Base de datos

Los diferentes catálogos definidos para el funcionamiento de la plataforma Web fueron integrados en una base de datos relacional, los cuales fueron agrupados de acuerdo con la función para la cual fueron diseñados, como: a) catálogos generales prellenados, b) catálogos de procesos que requieren una interfaz gráfica de usuario, y c) catálogos de procesos de referencia.

a) Catálogos generales prellenados (cg). Se refiere a información existente que es posible obtener de algún repositorio. Por ejemplo, los códigos postales que se encuentran almacenados en el Catálogo Nacional de Códigos Postales, elaborado por el Servicio Postal Mexicano (SEPOMEX), que se obtiene de forma gratuita. La Tabla 3.1 muestra la lista de catálogos pertenecientes a este grupo.

Catálogos generales prellenados
cg_usuarios_tipo
cg_estados
cg_colonias
cg_alcaldia_municipios
cg_calles
cg_codigos_postales

Cuadro 3.1: Catálogos generales prellenados.

b) Catálogos de procesos que requieren interfaz de usuario (cpi). Se requieren para llevar a cabo acciones como Crear, Consultar, Actualizar o Eliminar información de la plataforma Web. Estas operaciones comúnmente son conocidas como CRUD, que provienen del acrónimo en inglés Create, Read, Update, and Delete. La Tabla 3.2 muestra la lista de catálogos de esta categoría.

Catálogos de procesos que requieren interfaz de usuario
cpi_domicilios
cpi_empresas
cpi_empresas_domicilios
cpi_empresas_identificadores
cpi_residuos_autorizados_empresas
cpi_servicios_empresas
cpi_transportistas_tipo_vehiculos
cpi_transportistas_vehiculos
cpi_usuarios
cpi_visitantes

Cuadro 3.2: Catálogos de procesos que requieren interfaz de usuario de tipo CRUD.

c) Catálogos de procesos de referencia (cpr). Estos catálogos se utilizan para almacenar información que será consultada para verificar que se lleven a cabo procesos válidos en la plataforma Web, como estatus, residuos autorizados y servicios (Tabla 3.3). Además, se incluyeron tablas en los que se registran información relacionada con procesos para la publicación de ofertas, publicación de solicitudes, elaboración de manifiestos, entre otros (Tabla 3.4).

Catálogos de procesos de referencia
cpr_estatus_procesos
cpr_residuos_autorizados
cpr_servicios

Cuadro 3.3: Catálogos de procesos de referencia.

Tablas para procesos
p1_ofertas
p2_ofertas_solicitantes_candidatos
p1_solicitudes
p2_solicitudes_ofertadores_candidatos
p3_ofertas_solicitudes_vinculadas
p4_contratacion_trasporte
p5_recoleccion_entrega
p6_manifiestos
bitacora_notificaciones

Cuadro 3.4: Tablas para registrar información relacionada con los procesos.

Con base en lo anterior, en la Figura 40 se muestra, a manera de ejemplo, el modelo entidad-relación del diseño de la base de datos, donde se observa la estructura relacional y la lógica definida como parte del desarrollo del proyecto de la plataforma Web.

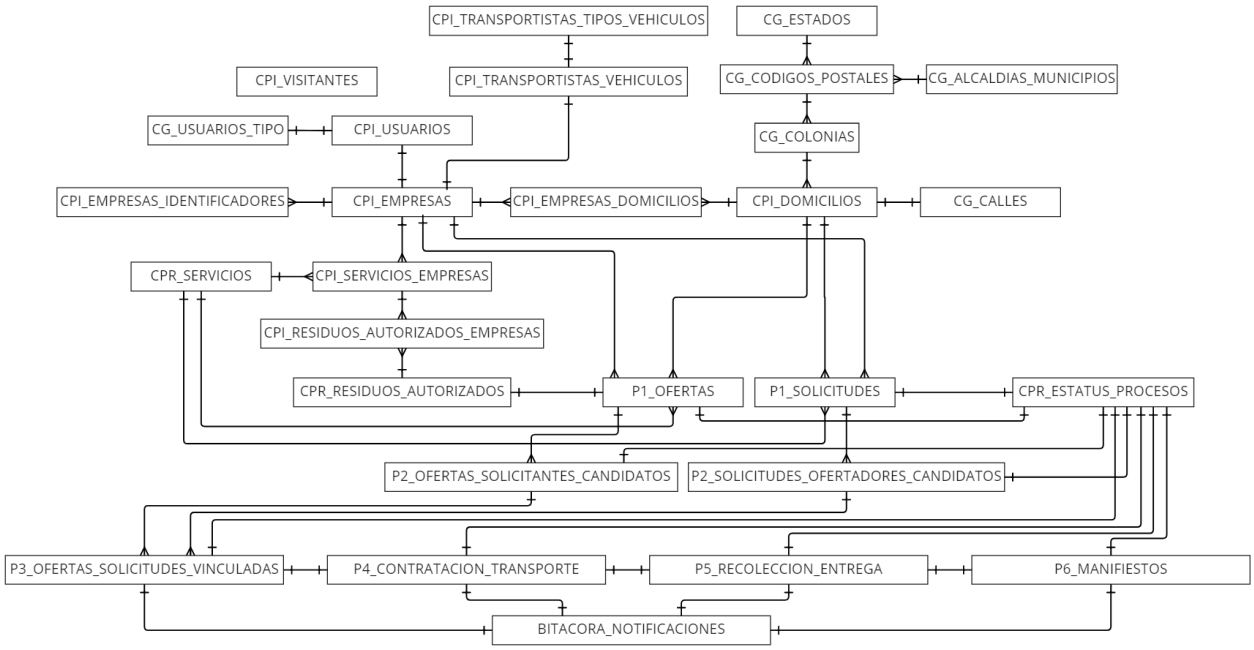

Figura 40: Modelo entidad-relación de la base de datos.

Para información ampliada sobre la descripción de los campos y los tipos de datos de cada uno de los catálogos definidos en el modelo entidad-relación se recomienda al lector revisar el Anexo A.

3.3.2. Back-End

Como se mencionó previamente, para el lado del Back-End se utilizó como lenguaje de programación Go (Goland), el cual es un lenguaje de programación compilado, y una API de GraphQL. Dentro de las bibliotecas existentes en Go, se probaron las APIs para GraphQL: i) *gqlgen* (github.com/99designs/gqlgen), y ii) *graphql-go* (github.com/graphql-go/graphql). En ambos casos se hicieron pruebas para definir la mejor opción en cuanto a la conexión con el Front-End. Así, con base en las pruebas realizadas se decidió utilizar *gqlgen*.

Gqlgen se basa en un lenguaje de definición de esquemas GraphQL. Prioriza la seguridad de los tipos de datos ante cualquier acceso a sus ubicaciones de memoria. Además, permite la generación de código, por lo que, la definición de los tipos se genera automáticamente mediante comandos, agilizando así la estructura inicial y cualquier cambio futuro en el esquema. Por lo tanto, para la configuración e inicialización de *gqlgen* se utilizó el comando siguiente:

```
1 go run github.com/99designs/gqlgen init
```

Código Fuente 6: *Comando para inicializar gqlgen.*

Una vez inicializado *gqlgen*, se crea una carpeta con la estructura para la generación de código mediante el esquema *graphqls*, tal como se muestra en la Figura 41.

Figura 41: Carpeta con la estructura inicial de gqlgen.

Posteriormente, se estableció el esquema dentro de los archivos: *schema.graphqls*, *input.graphqls*, *mutation.graphqls*, *output.graphqls* y *query.graphqls*. Estos archivos fueron creados para mantener un orden, dado a la gran cantidad de tipos, mutaciones y consultas requeridas en el desarrollo de la plataforma Web. A manera de ejemplo, el código siguiente muestra el esquema de calles para demostrar su implementación y funcionalidad. Por lo que, en *schema.graphqls* se declara el tipo cg_calles, donde se encuentra el identificador de la calle, el nombre completo, el nombre corto, y el nombre coloquial; así como una nota de la ubicación.

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

Código Fuente 7: *Esquema de calles.*

En *query.graphqls* se declara la consulta, mediante la cual se obtiene la información del esquema. En el código siguiente se muestra la consulta mediante el identificador único o los nombres.

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

Código Fuente 8: *Consultas de calles.*

A través de *mutacions.graphqls* se puede codificar operaciones para crear, actualizar y eliminar, tal como se muestra en el código siguiente:

```
type Mutation {
    CreateCalle(input: CalleCreateInput!): ID!
    UpdateCalle(id_calle: ID!, input: CalleUpdateInput!): String!
    DeleteCalle(id_calle: ID!): String!
}
```

Código Fuente 9: *Mutaciones de calles.*

En *input.graphqls* se define la separación de las entradas de creación y actualización, ambos por separado, tal como se muestra en el código siguiente:

```
input CalleCreateInput {
  nombre_calle: String!
  nota_ubicacion: String
}

input CalleUpdateInput {
  nombre_calle: String
  nombre_corto: String
  nombre_coloquial: String
  nota_ubicacion: String
}
```

Código Fuente 10: *Entradas de calles.*

Una vez declarado el esquema de GraphQL, con base en la codificación a través de gqlgen, se ejecuta el siguiente comando para enfocarse en la implementación de cada una de las consultas y mutaciones.

```
go run github.com/99designs/gqlgen generate
```

Código Fuente 11: *Comando para la generación de código.*

Por otro lado, para una conexión segura con la base de datos en PostgreSQL, se utilizó la biblioteca de gorm para Golang (https://gorm.io/docs/), la cual es una herramienta útil en operaciones con bases de datos. El Código Fuente 12, línea 23, muestra la variable que tiene la referencia de la base de datos que fue empleada en el desarrollo del proyecto.

```
package database

import (
        "fmt"
        "gorm.io/driver/postgres"
        "gorm.io/gorm"
)

type Config struct {
```

```
10          Host     string
11          Port     string
12          Password string
13          User     string
14          DBName   string
15          SSLMode  string
16  }
17
18  func NewConnection(config *Config) (*gorm.DB, error) {
19          dsn := fmt.Sprintf(
20                  "host=%s port=%s user=%s password=%s dbname=%s sslmode=%s",
21                  config.Host, config.Port, config.User, config.Password, config.
        DBName, config.SSLMode,
22          )
23          db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{
24                  SkipDefaultTransaction: true,
25          })
26          if err != nil {
27                  return db, err
28          }
29          return db, nil
30  }
```

Código Fuente 12: *Archivo de conexión a la base de datos.*

Asimismo, como parte del desarrollo fue necesario incluir la migración del modelo con el fin de interactuar con la base de datos. Este proceso, a manera de ejemplo, se muestra el archivo que contiene la tabla de *cg_calles*, utilizando los tipos de datos similares a los que se tienen en Go, para esto se ingresaron, a manera de ejemplo, valores nulos en la base de datos, tal como se muestra en el código siguiente:

```
1  package models
2
3  import "github.com/guregu/null"
4
5  type Cg_Calles struct {
6          Id_Calle         int         `json:"id_calle" gorm:"primary_key"`
7          Nombre_Calle     string      `json:"nombre_calle"`
8          Nombre_Corto     null.String `json:"nombre_corto"`
9          Nombre_Coloquial null.String `json:"nombre_coloquial"`
10         Nota_Ubicacion   null.String `json:"nota_ubicacion"`
11 }
```

Código Fuente 13: *Migración del modelo de cg_calles.*

3.3.2.1. Repositorio y funcionalidades CRUD

Para las funciones del repositorio, se crearon nuevos archivos, a manera de ejemplo, para el caso de la tabla anterior (cg_calles), se incluyeron cuatro componentes: i) CalleRepository, ii) CalleService, iii) NewCalleService, iv) operadores CRUD.

i) CalleRepository. Contiene los operadores CRUD, tal como se muestra en el código siguiente:

```
type CalleRepository interface {
        CreateCalle(input *model.CalleCreateInput, tx *gorm.DB) (*models.
    Cg_Calles, error)
        GetCalle(id_calle int, nombre_calle string, nombre_corto string,
    nombre_coloquial string) ([]*model.CgCalles, error)
        UpdateCalle(input *model.CalleUpdateInput, id int) error
        DeleteCalle(id int) error
}
```

Código Fuente 14: *Repositorio del catálogo Calles.*

ii) CalleService. Es una estructura para contener la instancia de la base de datos, tal como se observa en el código siguiente:

```
type CalleService struct {
        DB *gorm.DB
}

var _ CalleRepository = &CalleService{}
```

Código Fuente 15: *Servicio del catálogo Calles.*

iii) NewCalleService. Es una función que toma como argumento la instancia de la base de datos y devuelve un apuntador a la estructura *CalleService*, tal como se muestra en el código siguiente:

```
func NewCalleService(db *gorm.DB) *CalleService {
        return &CalleService{DB: db}
}
```

Código Fuente 16: *Constructor del servicio del catálogo Calles.*

iv) Operadores CRUD. La primera es la función *create* que recibe como argumentos una referencia a una transacción y un apuntador al modelo *CreateCalleInput*. Para luego devolver un puntero al modelo *cg_calle* y una excepción de error, tal como se muestra en el código siguiente:

```
func (c *CalleService) CreateCalle(input *model.CalleCreateInput, tx *gorm.DB)
    (*models.Cg_Calles, error) {
        calle := &models.Cg_Calles{
                Nombre_Calle:   input.NombreCalle,
                Nota_Ubicacion: null.StringFromPtr(input.NotaUbicacion),
        }

```

```
7         err := tx.Create(&calle).Error
8
9         if err != nil {
10                return nil, errors.New("Error en la creacion de la calle. " +
    err.Error())
11        }
12
13        return calle, err
14 }
```

Código Fuente 17: *Función create en el repositorio.*

La otra función es la lectura (*get*), que toma el ID (identificador) de la calle a obtener y devuelve un apuntador al modelo de *cg_calles* y un mensaje de error (excepción), tal como se muestra en el código siguiente:

```
1 func (c *CalleService) GetCalle(id_calle int, nombre_calle string, nombre_corto
     string, nombre_coloquial string) ([]*model.CgCalles, error) {
2         calle := []*model.CgCalles{}
3
4         err := c.DB.Where(
5                 "id_calle = ?", id_calle).Or(
6                 "nombre_calle = ?", nombre_calle).Or(
7                 "nombre_corto = ?", nombre_corto).Or(
8                 "nombre_coloquial = ?", nombre_coloquial,
9         ).Find(&calle).Error
10
11        return calle, err
12 }
```

Código Fuente 18: *Función de consulta en el repositorio.*

La función de actualización (*update*) toma como argumento el ID (identificador) de la calle a actualizar y un apuntador al modelo *CalleUpdateInput* y un mensaje de error (excepción), tal como se muestra en el código siguiente:

```
1 func (c *CalleService) UpdateCalle(input *model.CalleUpdateInput, id int) error
     {
2     // Se crea una variable para guardar la informaci n del registro anterior
3         calle := models.Cg_Calles{}
4         err := c.DB.First(&calle, id).Error
5
6         if err != nil {
7                 return err
8         }
9
10        if input.NombreCalle != nil {
11                calle.Nombre_Calle = *input.NombreCalle
12        }
13        if input.NombreCorto != nil {
14                calle.Nombre_Corto = null.StringFromPtr(input.NombreCorto)
```

```
15          }
16          if input.NombreColoquial != nil {
17                  calle.Nombre_Coloquial = null.StringFromPtr(input.
    NombreColoquial)
18          }
19          if input.NotaUbicacion != nil {
20                  calle.Nota_Ubicacion = null.StringFromPtr(input.NotaUbicacion)
21          }
22
23          err = c.DB.Model(&calle).Where("id_calle = ?", id).Updates(calle).Error
24          return err
25  }
```

Código Fuente 19: *Función de actualización en el repositorio.*

Finalmente, la función de borrado (*delete*) toma como argumento el ID ((identificador)) de la calle a borrar y devuelve un mensaje de error (excepción), tal como se muestra en el código siguiente:

```
1  func (c *CalleService) DeleteCalle(id int) error {
2          calle := &models.Cg_Calles{}
3          err := c.DB.Delete(calle, id).Error
4          return err
5  }
```

Código Fuente 20: *Función de eliminación en el repositorio.*

3.3.2.2. Dependencias de implementación (resolutores)

Gqlgen genera algunos resolutores (*resolver.go*), o resolución de paquetes, que contiene una estructura para mantener todas las dependencias de la aplicación que se usen en el desarrollo. Es una buena práctica mantener las implementaciones separadas, para cuando realmente se las necesite. Esto ayuda a la reutilización del código, dado que permite el uso de diferentes implementaciones para una función determinada y permite probarlas de manera fácil. Además, hace que el código sea mucho más legible. Para este caso, a modo de ejemplo, se muestra en el resolutor del catálogo Calles (código 21):

```
1  type Resolver struct {
2          Calle repository.CalleRepository
3  }
```

Código Fuente 21: *Ardchivo resolver.go del catálogo Calles.*

Por lo que, a partir de la generación del código, se crearon dos archivos: a) *query.go*, que contiene todas las declaraciones de las consultas dentro de *query.graphqls*; y b) *muta-*

tion.resolvers.go, que contiene las declaraciones de todas las mutaciones dentro de *mutation.graphqls*. Así, la función del catálogo Calles quedaría de la siguiente manera (código 22):

```
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 }
```

Código Fuente 22: *Consulta del catálogo Calles.*

Mientras que para el caso de las mutaciones de catálogo Calles, como: creación, actualización y eliminación, estas fueron declaradas a través de los códigos siguientes: 23, 24, y 25, respectivamente.

```
1  func (r *mutationResolver) CreateCalle(ctx context.Context, input model.
       CalleCreateInput) (int, error) {
2          var idCalle int
3          err := r.DB.Transaction(func(tx *gorm.DB) error {
4                  calle, err := r.Calle.CreateCalle(&input, tx)
5                  if err != nil {
6                          return err
7                  }
8                  idCalle = calle.Id_Calle
9                  return nil
10         })
11         if err != nil {
12                 return 0, err
13         }
14         return idCalle, nil
15 }
```

Código Fuente 23: *Mutación para la creación de una calle.*

```
1  func (r *mutationResolver) UpdateCalle(ctx context.Context, idCalle int, input
       model.CalleUpdateInput) (string, error) {
2          err := r.Calle.UpdateCalle(&input, idCalle)
3          if err != nil {
4                  return "nil", err
5          }
6          successMessage := "successfully updated"
```

```
7
8         return successMessage, nil
9 }
```

Código Fuente 24: *Mutación para la actualización de una calle.*

```
1 func (r *mutationResolver) DeleteCalle(ctx context.Context, idCalle int) (
      string, error) {
2         err := r.Calle.DeleteCalle(idCalle)
3         if err != nil {
4                 return "", err
5         }
6         successMessage := "successfully deleted"
7         return successMessage, nil
8 }
```

Código Fuente 25: *Mutación para la eliminación de una calle.*

3.3.2.3. Creación de la función principal

Por último, dentro del archivo *server.go* se creó la función principal. En la primera parte de esta función se importaron las bibliotecas utilizadas, luego funciones que fueron implementadas, y posteriormente la declaración del puerto utilizado.

```
1 package main
2
3 import (
4         "apiCatalogosGenerales/app/database"
5         "apiCatalogosGenerales/app/repository"
6         "apiCatalogosGenerales/graph"
7         "apiCatalogosGenerales/graph/generated"
8         "log"
9         "net/http"
10        "os"
11
12        "github.com/99designs/gqlgen/graphql/handler"
13        "github.com/99designs/gqlgen/graphql/handler/transport"
14        "github.com/99designs/gqlgen/graphql/playground"
15        "github.com/go-chi/chi/v5"
16        "github.com/gorilla/websocket"
17        "github.com/joho/godotenv"
18        "github.com/rs/cors"
19 )
20
21 const defaultPort = "XXXX"
```

Código Fuente 26: *Función principal en el archivo server.go.*

Luego, se cargaron las variables de entorno, las cuales contienen la información necesaria para instanciar la referencia a la base de datos, y se declaró el puerto de escucha, tal como se muestra en el código siguiente:

```
// Obtenci n de variables del sistema de bases de datos guardado en el archivo .env
godotenv.Load()
config := &database.Config{
        Host:     os.Getenv("DB_HOST"),
        Port:     os.Getenv("DB_PORT"),
        Password: os.Getenv("DB_PASSWORD"),
        User:     os.Getenv("DB_USER"),
        SSLMode:  os.Getenv("DB_SSLMODE"),
        DBName:   os.Getenv("DB_NAME"),
}
// Conexion a la base de datos
db, err := database.NewConnection(config)
if err != nil {
        panic(err)
}
// Definici n del puerto de escucha
port := os.Getenv("PORT")
if port == "" {
        port = defaultPort
}

router := chi.NewRouter()

// Add CORS middleware around every request
// See https://github.com/rs/cors for full option listing
router.Use(cors.New(cors.Options{
        AllowedOrigins:   []string{"http://localhost:*"}, // Acceso a trav s de todos los puertos en el front end
        AllowCredentials: true,
        Debug:            true,
}).Handler)
```

Código Fuente 27: *Creación a la instancia de la base de datos y el puerto de escucha.*

Adicionalmente, a modo de ejemplo, se inicializó el repositorio de Calles, y posteriormente se creó el servidor de Graphql, configurando la ruta de acceso, la cual se nombró como *catalogosGenerales*, tal como se muestra en el código siguiente:

```
repoCalle := repository.NewCalleService(db)

srv := handler.NewDefaultServer(generated.NewExecutableSchema(generated.Config{
    Resolvers: &graph.Resolver{
    Calle:                              repoCalle,
}}))

srv.AddTransport(&transport.Websocket{
        Upgrader: websocket.Upgrader{
```

```
                CheckOrigin: func(r *http.Request) bool {
                        // Check against your desired domains here
                        return r.Host == ""
                },
                ReadBufferSize:  1024,
                WriteBufferSize: 1024,
        },
})
// Ruta de acceso al playground
router.Handle("/", playground.Handler("GraphQL playground", "/
    catalogosGenerales"))
// Ruta de acceso para el frontend
router.Handle("/catalogosGenerales", srv)

err = http.ListenAndServe(":XXXX", router)
if err != nil {
        panic(err)
}

log.Printf("connect to http://localhost:%s/ for GraphQL playground", port)
log.Fatal(http.ListenAndServe(":"+port, nil))
```

Código Fuente 28: *Inicialización del repositorio de Calles.*

Es importante destacar que con respecto a la seguridad en el intercambio de datos, estas APIs desarrolladas, al actuar como intermediario entre la aplicación y el servidor, proveen un nivel de seguridad adicional, esto debido a que en el punto final de la API se desvincula la aplicacioín que consume la infraestructura que proporciona el servicio. Dado que Gqlgen, como se mencionó previamente, prioriza la seguridad de los tipos de datos ante cualquier acceso a sus ubicaciones de memoria. Además, se incluyeron credenciales de autorizacioín para reducir el riesgo de ataques al servidor y limitar el acceso para minimizar las amenazas a la seguridad.

3.4 Síntesis

En este capítulo se presentó la propuesta de solución, basado en las primeras etapas de una metodología de desarrollo ágil. Esta estrategia de desarrollo permitió tener una solución dinámica y adapta al cambio. Durante la etapa de diagnóstico se llevaron a cabo diversas reuniones de trabajo con el equipo de desarrollo y los usuarios finales, esto con el fin de generar una lista de requerimientos funcionales y no funcionales. Durante la etapa de diseño se elaboraron los casos de uso, los diagramas de secuencia y los diagramas de flujo, los cuales fueron la base y necesarios para una construcción eficiente de la aplicación Web. Por otro lado, durante el desarrollo de la aplicación se creó también la base de datos en PostgreSQL, la cual a lo largo del desarrollo del proyecto tuvo varias modificaciones con base en las ne-

cesidades del usuario final, en este caso la SEDEMA. Además, se realizó la implementación de los catálogos a través de la API de GraphQL en el lenguaje de programación GoLang, para esto se utilizó la biblioteca gqlgen. De igual manera, se hicieron pruebas para comprobar la funcionalidad de las implementaciones realizadas, comprobando así la eficiencia de los operadores y funcionalidades desarrolladas.

CAPÍTULO 4

Resultados

En el capítulo anterior se presentó la propuesta de solución basada en una secuencia de las etapas principales de una metodología ágil de desarrollo de software (diagnóstico, diseño y desarrollo). Esto permitió llevar a cabo un desarrollo de la solución de manera adecuada, logrando así implementar las funcionalidades requeridas del lado del Back-End para su uso en el Front-End de la plataforma SEDEMA.

En este capítulo se presentan los resultados obtenidos con base a los alcances de la propuesta de solución, siendo importante destacar el trabajo realizado desde el diseño hasta el desarrollo de las APIs para los diversos catálogos requeridos en el funcionamiento del encadenamiento productivo de los residuos sólidos urbanos de la Ciudad de México. Por lo que, se hace un desglose de las pruebas efectuadas en las consultas y mutaciones que fueron desarrolladas para esta aplicación Web.

4.1 APIs desarrolladas

Con base en el diseño y desarrollo descrito en el capítulo anterior, donde se crearon APIs con diferentes operaciones para la consulta y manipulación de datos que son utilizadas desde el lado del cliente (Front-End), a continuación se presenta un resumen, en forma de tablas, de las consultas y mutaciones de los catálogos generales implementados (Tabla 4.1); consultas de los catálogos de referencia (Tabla 4.2); funciones adicionales (Tabla 4.3); consultas y mutaciones de los catálogos de interfaz (Tabla 4.4); y consultas y mutaciones de los catálogos de procesos (Tabla 4.5).

Consultas de catálogos generales	
Usuario_tipos	GetAllUsuarios
	GetUsuarioTipo
Estados	GetAllEstados
	GetEstado
Colonias	GetColonia
	GetInfoCodigoPostal
Alcaldia_municipios	GetAllAlcaldiasMunicipios
	GetAlcaldiaMunicipio
Calles	GetCalle
Codigos_postales	GetCodigoPostal
	GetCodigoPostalPorStr
	GetAllCodigosPostales
Mutaciones de catálogos generales	
Calles	CreateCalle
	UpdateCalle
	DeleteCalle

Cuadro 4.1: Consultas y mutaciones de catálogos generales.

Consultas de catálogos de referencia	
Servicios	GetAllServicios
	GetServicio
Residuos_autorizados	GetResiduoAutorizado
	GetAllResiduosTipo
Estatus_procesos	GetEstatusProceso

Cuadro 4.2: Consultas de catálogos de referencia.

Funciones adicionales
InicioSesión
UpdatePassword
CreateBitácoraNotificacion
GetBitacoraNotificacion
UpdateBitacoraNotificacion
DeleteBitacoranotificacion

Cuadro 4.3: Funciones adicionales implementadas.

Consultas de catálogos de interfaz	
Empresas	GetEmpresa
Ususarios	GetUsuarioByID
	GetUsuarioByEmpresa
	GetUsuarioByTipoUsuario
Domicilios	GetDomicilio
Empresas_domicilios	GetEmpresaDomicilio
	GetEmpresaDomicliosPorEmpresa
Empresas_identificadores	GetEmpresaIdentificador
	GetAllEmpresaIdentificadorPorEmpresa
Servicios_empresas	GetServiciosEmpresas
	GetTransportistasTipoVehiculos
Transportistas_vehiculos	GetTransportistasVehiculos
Residuos_autorizados_empresas	GetResiduoAutorizadoEmpresa
	GetResiduoAutorizadoEmpresaPorNormativa
Visitantes	GetVisitante
	GetVisitanteByEmail
	GetAllVisitantes
Mutaciones de catálogos de interfaz	
Empresas	CreateEmpresas
	UpdateEmpresa
	DeleteEmpresa
Ususarios	CreateUsuario
	UpdateUsuario
	DeleteUsuario
Domicilios	CreateDomicilio
	UpdateDomicilio
	DeleteDomicilio
Empresas_domicilios	UpdateEmpresaDomicilio
	DeleteEmpresaDomicilio
Empresas_identificadores	CreateEmpresaIdentificador
	UpdateEmpresaIdentificador
	DeleteEmpresaIdentificador
Servicios_empresas	CreateServiciosEmpresas
	UpdateServicioEmpresa
	DeleteServicioEmpresa
Transportistas_tipo_vehículo	CreateTransportistasTipoVehiculos
	UpdateTransportistasTipoVehiculos
	DeleteTransportistasTipoVehiculos
Transportistas_vehiculos	CreateTransportistasVehiculos
	UpdateTransportistasVehiculos
	DeleteTransportistasVehiculos
	CreateResiduosAutorizadosEmpresas
	UpdateResiduosAutorizadosEmpresas
	DeleteResiduosAutorizadosEmpresas
Visitantes	CreateVisitante
	UpdateVisitante
	DeleteVisitante

Cuadro 4.4: Consultas y mutaciones de catálogos de interfaz.

Consultas de catálogos de procesos	
Ofertas	GetOfertas
	GetOfertaPorResiduo
	GetOfertaPorServicio
Solicitudes	Get
	GetSolicitudByResiduo
	GetSolicitudByServicio
Ofertas_solicitantes_candidatos	GetOfertaSolicitanteCandidato
	GetOfertaSolicitanteCAndidatoPorSolicitud
Solicitudes_ofertatntes_candidatos	GetSolicitudOfertadorCandidato
	GetSolicitudOfertadorCandidatoPorSolicitud
Ofertas_solicitudes_vinculadas	GetOfertasSolicitudesVinculadas
	GetContratacionTransporte
	GetRecoleccionEntrega
Manifiestos	GetManifiesto
	GetManifiestoPorEmpresa
Mutaciones de catálogos de procesos	
Ofertas	CreatePOferta
	UpdatePOferta
	DeletePOferta
Solicitudes	CreateSolicitud
	UpdateSolicitud
	DeleteSolicitud
Ofertras_solicitantes_candidatos	CreateOfertaSolicitanteCandidato
	UpdateOfertaSolicitanteCandidato
	DeleteOfertaSolicitanteCandidato
Solicitudes_ofertatntes_candidatos	CreateSolicitudOfertanteCandidato
	UpdateSolicitudOfertanteCandidato
	DeleteSolicitudOfertanteCandidato
Ofertas_solicitudes_vinculadas	CreateOfertaSolicitudVinculada
	UpdateOfertaSolicitudVinculada
	DeleteOfertaSolicitudVinculada
Contratacion_transporte	CreateContratacionTransporte
	UpdateContratacionTransporte
	DeleteContratacionTransporte
Recoleccion_entrega	CreateRecoleccionEntrega
	UpdateRecoleccionEntrega
	DeleteRecoleccionEntrega
Manifiestos	CreateManifiesto
	UpdateManifiesto
	DeleteManifiesto

Cuadro 4.5: Consultas y mutaciones de catálogos de procesos.

Como se mencionó previamente, los catálogos generales (cg) son información existente, previamente almacenados; tal es el caso de los códigos postales, que se encuentran registrados en una determinada tabla. Mientras que los catálogos de procesos que requieren interfaz de usuario (cpi) conllevan operaciones de creación, consulta, actualización y eliminación de información en la plataforma web. Por su parte, los catálogos de procesos de referencia (cpr)

se utilizaron para almacenar información de consulta y verificar que se lleven a cabo los procesos válidos en la plataforma Web, como estatus, residuos autorizados, servicios, procesos para la publicación de ofertas, publicación de solicitudes, elaboración de manifiestos, entre otros.

4.2 Aseguramiento de la calidad

Para comprobar el funcionamiento de las consultas y mutaciones asociadas a los catálogos desarrollados, se hicieron pruebas mediante las sintaxis asociadas a estas. Para esto se hicieron pruebas de consulta y funcionalidad a través de Postman, la cual, como se mencionó en la sección 3.2.1 (Herramientas de desarrollo utilizadas), fue útil para probar las APIs desarrolladas. A modo de ejemplo, en las subsecciones siguientes se muestran las pruebas de funcionamiento realizadas en los catálogos: a) calles, de catálogos generales; y b) domicilio, de catálogos de interfaz. Las pruebas realizadas con los otros catálogos desarrollados se presenta, a manera de información ampliada, en el Anexo B.

4.2.1. Catálogo de calles

En este sentido, para el catálogo de calles (*cg_calles*) se hicieron las consultas basadas en su sintaxis, tal como se muestra en el código 29. Además, se aseguró su correcto funcionamiento, tal como se muestra en la Figura 42.

```
query(
    $id_calle: ID,
    $nombre_calle: String,
    $nombre_corto: String,
    $nombre_coloquial: String
){
    GetCalle(
        id_calle: $id_calle,
        nombre_calle: $nombre_calle,
        nombre_corto: $nombre_corto,
        nombre_coloquial: $nombre_coloquial
    ){
        id_calle
        nombre_calle
        nombre_corto
        nombre_coloquial
        nota_ubicacion
    }
}
```

Código Fuente 29: *Sintaxis de la consulta de GetCalle.*

Figura 42: Consulta de GetCalle en Postman.

De igual manera, se probó la creación de una calle. En este caso se puede apreciar en el código siguiente, 30, donde se declararon los parámetros necesarios y se enviaron las variables en una estructura JSON (ver Figura 43). Además, dentro de la base de datos en PostgreSQL, se creó un nuevo registro con la información que fue enviada, tal como se muestra en la Figura 44.

```
mutation(
    $nombre_calle: String!,
    $nota_ubicacion: String
){
    CreateCalle(
        input: {
            nombre_calle: $nombre_calle,
            nota_ubicacion: $nota_ubicacion
        }
    )
}
```

Código Fuente 30: *Sintaxis de la mutación de CreateCalle.*

Figura 43: Consulta de GetCalle en Postman.

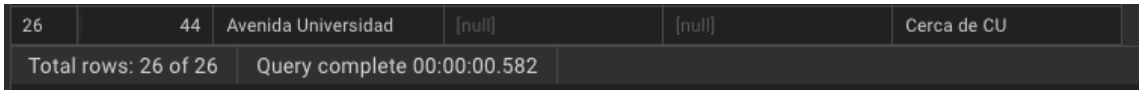

Figura 44: Registro de la información enviada a la base de datos en PosgreSQL.

Por otra parte, se realizaron pruebas de las mutaciones para actualizar y eliminar un registro. Tal como se observa en los códigos siguientes: a) actualización de una calle (Código 31); y eliminación de una calle (Código 32).

```
mutation(
    $id_calle: ID!,
    $nombre_calle: String,
    $nombre_corto: String,
    $nombre_coloquial: String,
    $nota_ubicacion: String
){
    UpdateCalle(
        id_calle: $id_calle,
        input:{
            nombre_calle: $nombre_calle
            nombre_corto: $nombre_corto
            nombre_coloquial: $nombre_coloquial
            nota_ubicacion: $nota_ubicacion
        }
    )
}
```

Código Fuente 31: *Sintaxis de la mutación de UpdateCalle.*

```
mutation(
    $id_calle: ID!
){
    DeleteCalle(
    id_calle: $id_calle,
    )
}
```

Código Fuente 32: *Sintaxis de la mutación de DeleteCalle.*

Las Figuras 45 y 46 muestran las pruebas realizadas en Postman, donde en la primera se observa la actualización de una calle y en la otra la modificación de una calle, respectivamente. Asimismo, se muestra la ejecución de actualización y modificación en la base de datos en PostgreSQL (Figuras 47 y 48).

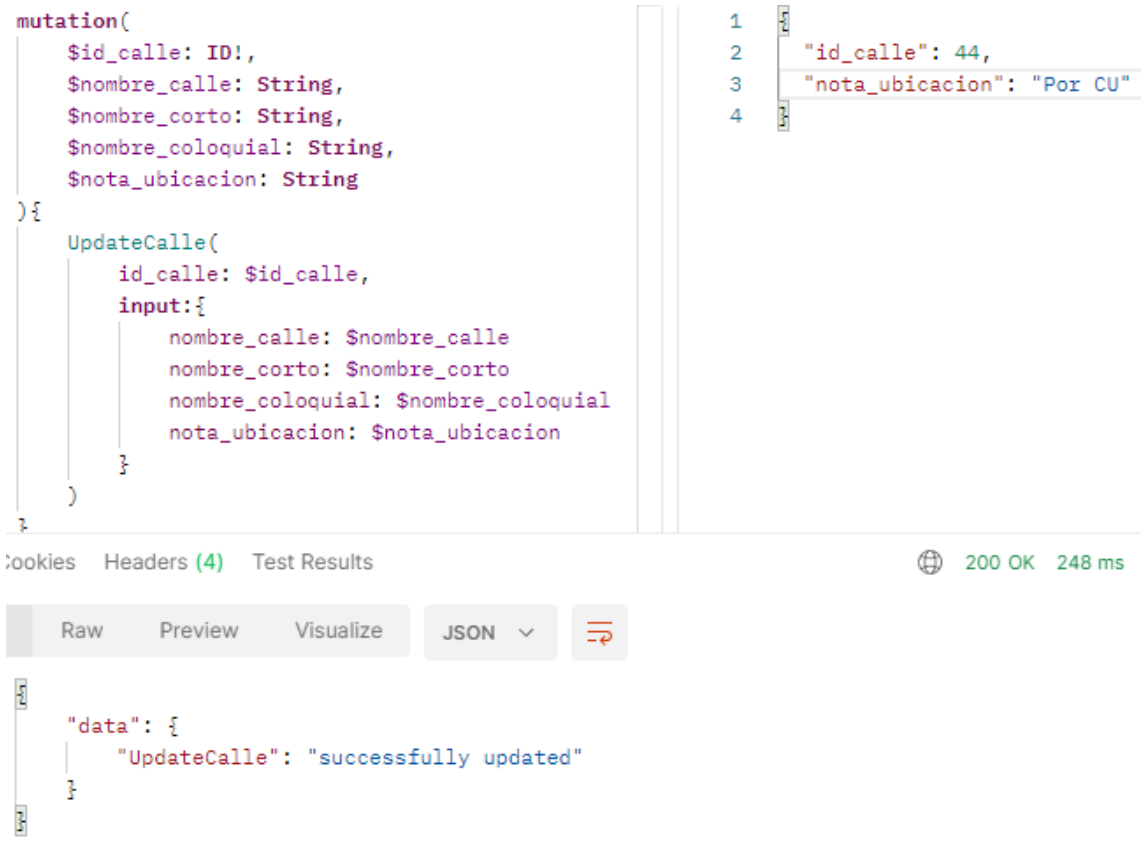

Figura 45: Actualización de una calle.

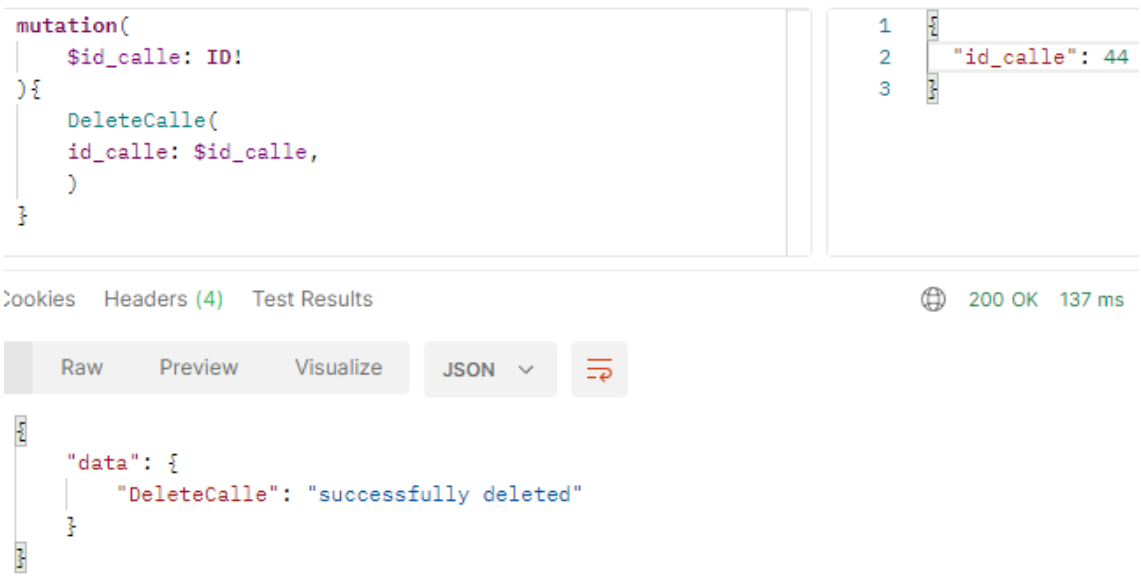

Figura 46: Eliminación de una calle.

Figura 47: Actualización de una calle en la base de datos en PostgueSQL.

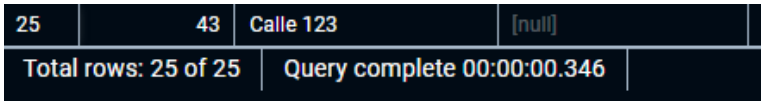

Figura 48: Eliminación de una calle en la base de datos en PostgueSQL.

4.2.2. Catálogo de domicilios

Para el catálogo de domicilios (*cpi_domicilios*), que tiene dependencia con otros catálogos, el proceso es similar al caso anterior, con la diferencia de que en estos casos dentro de las consultas se retorna el esquema completo de la referencia, aprovechando así al máximo la arquitectura de GraphQL. En este sentido, de modo demostrativo se utiliza el esquema de domicilios.

En este sentido, dentro del archivo *schema.graphqls* fue necesario realizar los esquemas de estado de código postal, alcaldia_municipio y colonia; que fueron implementados de manera similar al catálogo de calles. Por lo que, con base en los catálogos anteriores, se declaró el esquema de domicilio, tal como se puede apreciar en el código siguiente:

```
type Cpi_domicilios {
  id_domicilio: ID!
  colonia: Cg_colonias!
  calle: Cg_calles!
  numero_exterior: String!
  numero_interior: String
  referencia_domicilio: String!
  nota_domicilio: String  <
  nota_descripcion: String
  latitud: String
  longitud: String
  url_googlemaps: String
  ctl_es_registro_activo: Boolean
  ctl_fecha_registro: Timestamp
  ctl_usuario_registro: String
  ctl_fecha_modificacion: Timestamp
  ctl_usuario_modificacion: Int
}
```

Código Fuente 33: *Esquema del catálogo de domicilios.*

Posteriormente, para la ejecución del código se hizo la declaración de los archivos: i) *query.graphqls*, ii) *mutation.graphqls*, y iii) *input.graphqls*, tal como se muestra en los códigos 34, 35 y 36, que representan la consulta de domicilios, las mutaciones de domicilios, y las entradas de domicilios, respectivamente.

```
GetDomicilio(id_domicilio: ID!): Cpi_domicilios!
```

Código Fuente 34: *Consultas de domicilios.*

```
CreateDomicilio(input: DomicilioCreateInput!): String!
UpdateDomicilio(id_domicilio: ID!, input: DomicilioUpdateInput!): String!
DeleteDomicilio(id_domicilio: ID!): String!
```

Código Fuente 35: *Mutaciones de domicilios.*

```
input DomicilioCreateInput {
  id_colonia: ID!
  nombre_calle: String!
  id_calle: ID
  nota_ubicacion: String
  numero_exterior: String!
  numero_interior: String
  referencia_domicilio: String
  nota_domicilio: String
  nota_descripcion: String
  latitud: String
  longitud: String
  url_googlemaps: String
}
```

```
15
16 input DomicilioUpdateInput {
17   id_colonia: ID
18   numero_exterior: String
19   numero_interior: String
20   referencia_domicilio: String
21   nota_domicilio: String
22   nota_descripcion: String
23   latitud: String
24   longitud: String
25   url_googlemaps: String
26 }
```

Código Fuente 36: *Entradas de domicilios.*

Para manipular la base de datos, al igual que el caso anterior, se creó una consulta que requiere instanciar los objetos: CalleService y ColoniaService, mediante los cuales se tiene acceso a las funciones de consulta y obtención de estructuras que almacenan los datos. En el código 37, específicamente en las líneas 12 y 17 se crean estos objetos, y en las líneas 14 y 19 se obtienen las estructuras por medio del identificador único (Id).

```
1  func (d DomicilioService) GetDomicilio(id_domicilio int) (*model.CpiDomicilios,
       error) {
2          domicilioSearch := &models.Cpi_domicilios{}
3          err := d.DB.Where(
4                  "id_domicilio = ?", id_domicilio,
5          ).First(&domicilioSearch).Error
6
7          if err != nil {
8                  return &model.CpiDomicilios{}, err
9          }
10
11         // Creaci n de un "objeto" coloniaService para poder utilizar sus
       metodos
12         coloniaService := ColoniaService{DB: d.DB}
13         // Se obtiene la informaci n de la colonia
14         colonia, _ := coloniaService.GetColonia(domicilioSearch.Id_Colonia, "")
15
16         // Creaci n de un "objeto" coloniaService para poder utilizar sus
       metodos
17         calleService := CalleService{DB: d.DB}
18         // Se obtiene la informaci n de la colonia
19         calle, _ := calleService.GetCalle(domicilioSearch.Id_Calle, "", "", "")
20
21         // Conversi n de fechas a string
22         ctlFechaRegistro := domicilioSearch.Ctl_Fecha_Registro.String()
23         ctlFechaModificacion := domicilioSearch.Ctl_Fecha_Modificacion.String()
24
25         numeroInterior := null.StringFrom(domicilioSearch.Numero_Interior.
       String).String
26         referenciaDomicilio := null.StringFrom(domicilioSearch.
       Referencia_Domicilio.String).String
```

```
27          notaDomicilio := null.StringFrom(domicilioSearch.Nota_Domicilio.String)
    .String
28          notaDescripcion := null.StringFrom(domicilioSearch.Nota_Descripcion.
    String).String
29          latitud := null.StringFrom(domicilioSearch.Latitud.String).String
30          longitud := null.StringFrom(domicilioSearch.Longitud.String).String
31          urlGooglemaps := null.StringFrom(domicilioSearch.URL_Googlemaps.String)
    .String
32
33          domicilio := &model.CpiDomicilios{
34                  IDDomicilio:            domicilioSearch.Id_Domicilio,
35                  Colonia:                colonia[0],
36                  Calle:                  calle[0],
37                  NumeroExterior:         domicilioSearch.Numero_Exterior,
38                  NumeroInterior:         &numeroInterior,
39                  ReferenciaDomicilio:    referenciaDomicilio,
40                  NotaDomicilio:          &notaDomicilio,
41                  NotaDescripcion:        &notaDescripcion,
42                  Latitud:                &latitud,
43                  Longitud:               &longitud,
44                  URLGooglemaps:          &urlGooglemaps,
45                  CtlEsRegistroActivo:    &domicilioSearch.
    Ctl_Es_Resgistro_Activo,
46                  CtlFechaRegistro:       &ctlFechaRegistro,
47                  CtlUsuarioRegistro:     &domicilioSearch.Ctl_Usuario_Registro,
48                  CtlFechaModificacion:   &ctlFechaModificacion,
49                  CtlUsuarioModificacion: &domicilioSearch.
    Ctl_Usuario_Modificacion,
50          }
51
52          return domicilio, err
53  }
```

Código Fuente 37: *Función de GetDomicilio en el repositorio.*

Por otra parte, tal como el caso anterior, se aseguró el correcto funcionamiento de la consulta de un domicilio mediante su ejecución en Postman, tal como se muestra en la Figura 49. Logrando así asegurar la calidad de la codificación de todos los catálogos desarrollados para la plaraforma Web.

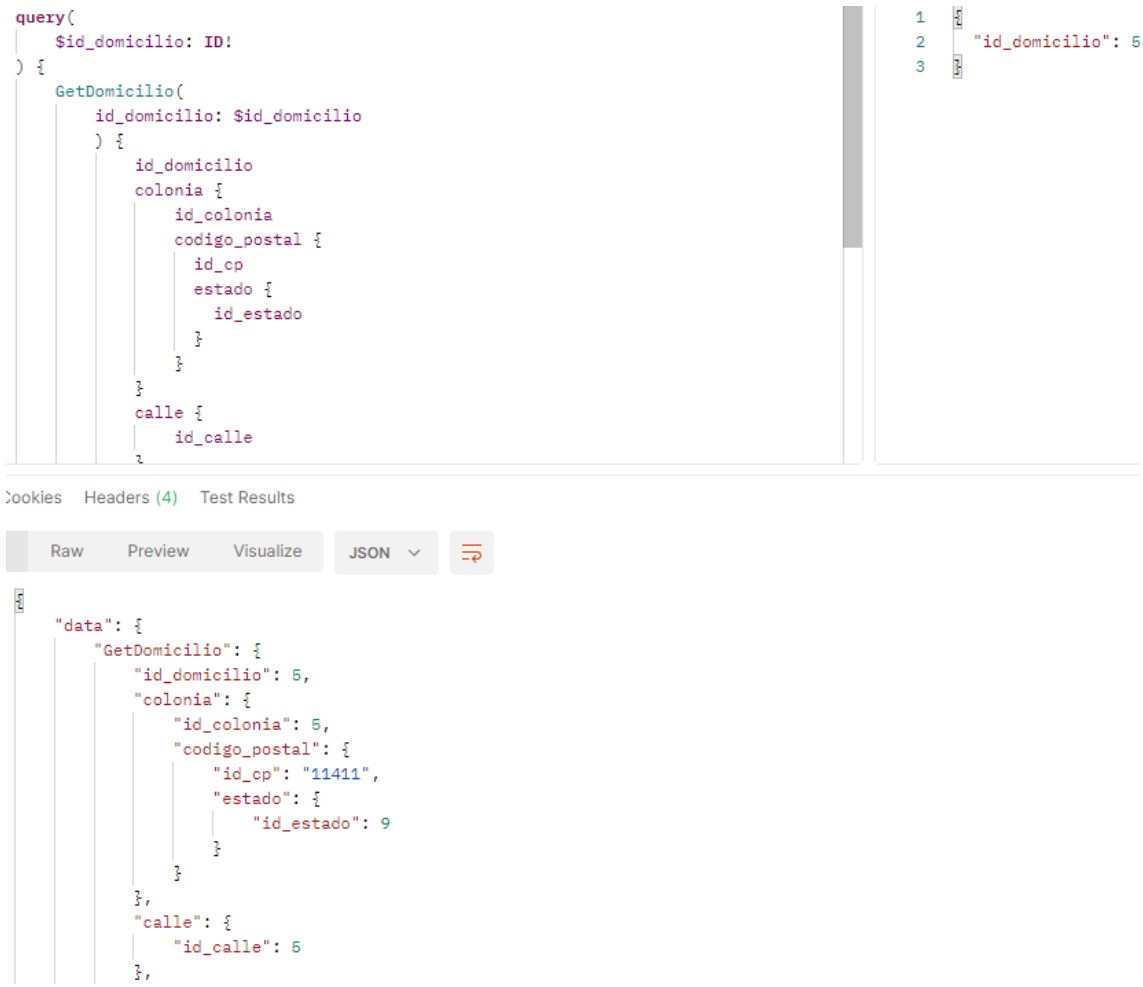

Figura 49: Consulta del domicilio en Postman.

4.3 Desplegado

La página de inicio de la plataforma SEDEMA ofrece cuatro opciones en su menú: i) Inicio, ii) Iniciar Sesión, iii) Registro, y iv) Acerca de. En la Figura 50 se observa la interfaz gráfica de la página web de inicio. La SEDEMA proporcionó la información que se despliega en la página. De manera general, cada una de estas opciones se describen a continuación:

- Inicio. Esta opción es para regresar a la página principal, donde se encuentra un carrusel con información de la plataforma.
- Iniciar sesión. Si el usuario se encuentra registrado, puede directamente seleccionar esta opción e ingresar a la plataforma con sus datos de registro.
- Registro. En esta opción el usuario puede iniciar el registro en la plataforma SEDEMA en caso de que no forme parte de esta.

- Acerca de. Se presenta información relacionada con la plataforma web.

Figura 50: Página web de inicio.

4.3.1. Ingreso a la plataforma (Iniciar sesión)

La Figura 51 muestra la página web implementada para el inicio de sesión, donde se solicitan los siguientes datos:

- Usuario.
- Contraseña.
- Recuperar la contraseña mediante la opción “Olvidé mi contraseña”.
- Si el usuario seleccionó “Iniciar sesión” y no está registrado, puede hacerlo directamente en la opción “REGÍSTRATE”.
- Si el usuario solo quiere conocer la plataforma puede ingresar como “VISITANTE”, sin embargo, esta opción no le permitirá realizar transacciones.

Figura 51: Interfaz gráfica de usuario del inicio de sesión.

4.3.2. Aplicación web para el registro del usuario visitante

Para tener estadísticas de los usuarios visitantes, al acceder a la plataforma se les solicita un registro sencillo, como se observa en la Figura 52, donde se presenta la página web del registro del usuario Visitante.

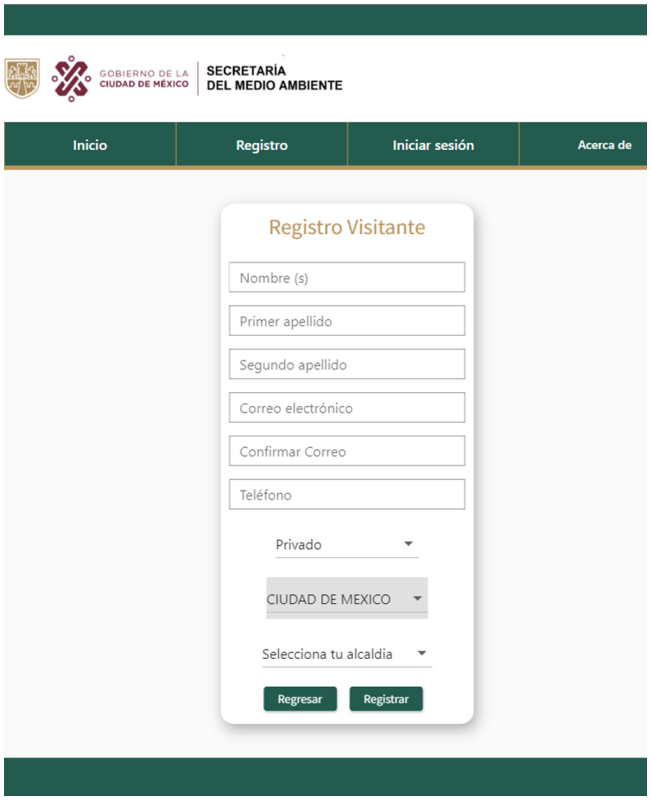

Figura 52: Aplicación web para el registro del usuario visitante.

4.3.3. Registro de nuevos usuarios

Para el registro de prestadores de servicios se tiene una serie de formularios que son necesarios llenar, como: a) datos generales; b) domicilio y datos de la empresa; y c) instrumentos regulatorios y autorizaciones con las que cuenta. La Figura 53 muestra la captura de los datos generales para el prestador de servicios. Se solicitan los datos de la persona, ya sea Física o Moral, incluyendo correo electrónico, teléfonos, puesto de la empresa o giro y contraseña con su confirmación.

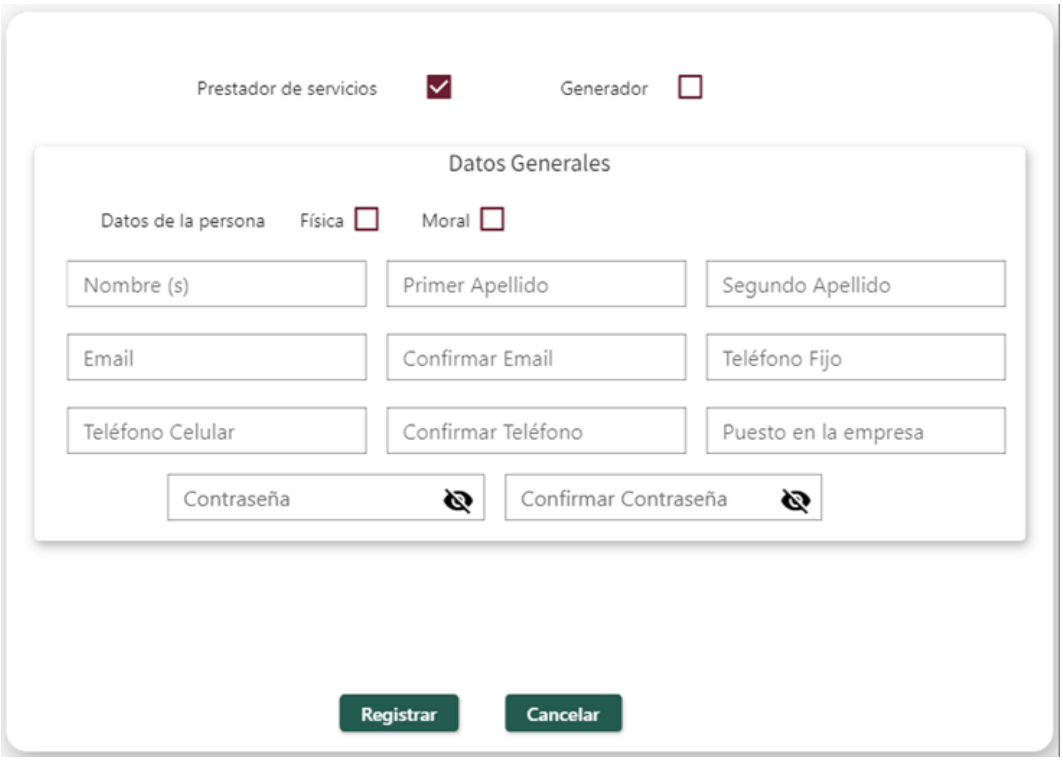

Figura 53: Registro de datos generales para el prestador de servicios.

Por su parte, la Figura 54 muestra el formulario para capturar los datos del domicilio de la empresa, así como la información fiscal de la misma: razón social, registro federal de contribuyentes (RFC), si es matriz o sucursal, y otros campos que conforman el registro.

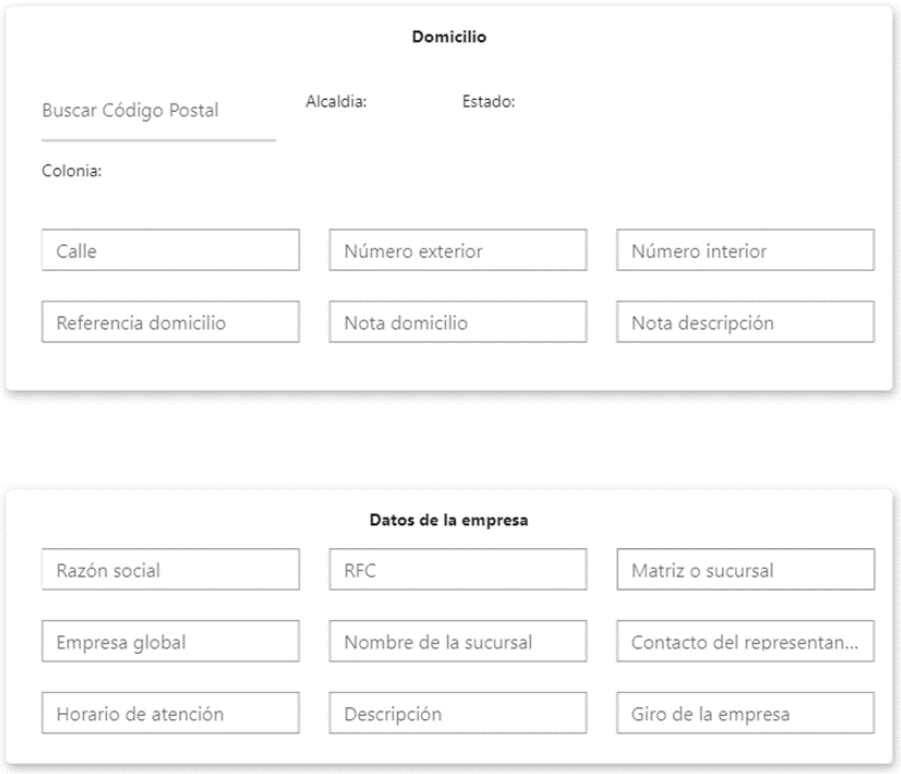

Figura 54: Registro del domicilio y datos de la empresa del prestador de servicios.

El prestador de servicios captura también los datos de los instrumentos regulatorios con los que cuenta sobre una determinada autorización. En la Figura 55 se muestra el formulario antes de seleccionar algún instrumento regulatorio.

Figura 55: Formulario de registro de instrumentos regulatorios.

En el formulario para el registro de la Licencia Ambiental Única, mostrado en la Figu-

ra 56, se aprecia que se solicita el número de permiso o autorización, fecha y vigencia. Los residuos están categorizados como: orgánicos, inorgánicos y de manejo especial. Cada categoría de residuos tiene su propio listado de residuos, de los cuales se pueden seleccionar. Una vez seleccionados los residuos de las distintas categorías, se presenta la opción de residuos seleccionados, tal como se aprecia en la Figura 57.

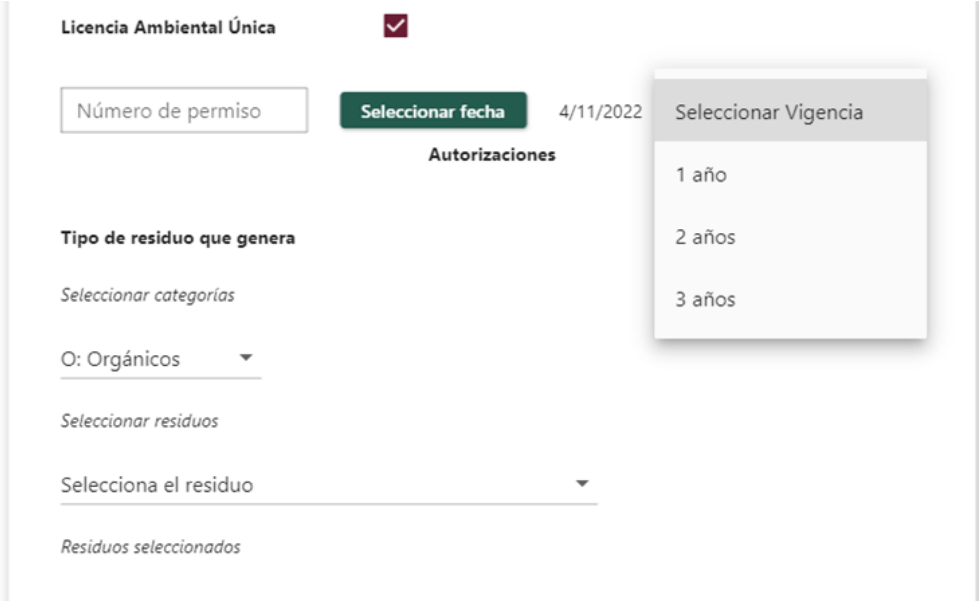

Figura 56: Formulario para el registro de la Licencia Ambiental Única.

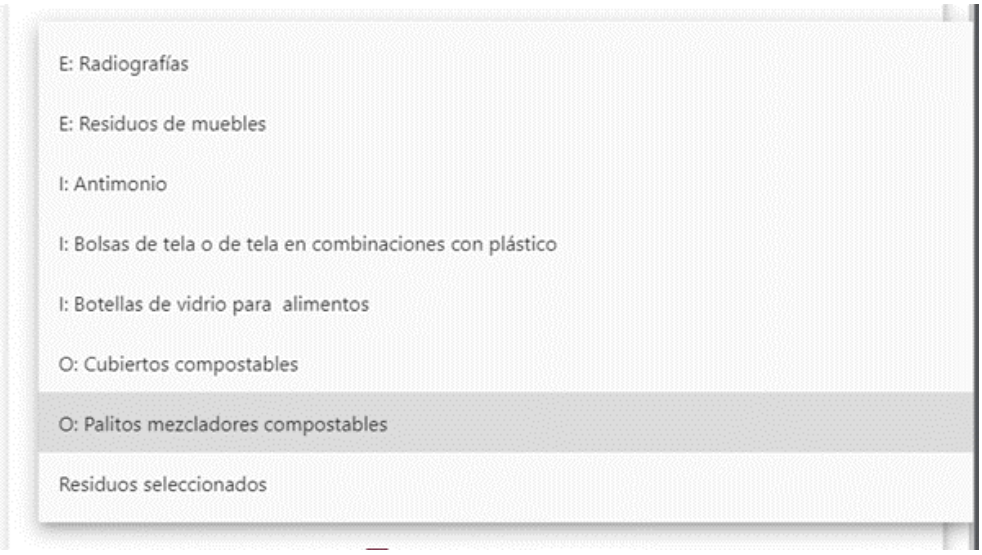

Figura 57: Lista de residuos seleccionados en la Licencia Ambiental Única.

4.3.4. Aplicación web para los usuarios ofertadores

La Figura 58 muestra la aplicación web para el registro de las ofertas de residuos por parte del generador (ofertante). En dicha aplicación se solicita definir la fecha de inicio de la oferta, fecha de término, dirección en general, e información adicional.

Figura 58: Aplicación para el registro de ofertas de residuos por parte del generador.

La Figura 59 muestra las ofertas ya registradas por parte del usuario generador. Posteriormente, estas pueden ser consultadas por otros usuarios, obteniendo todas las ofertas disponibles.

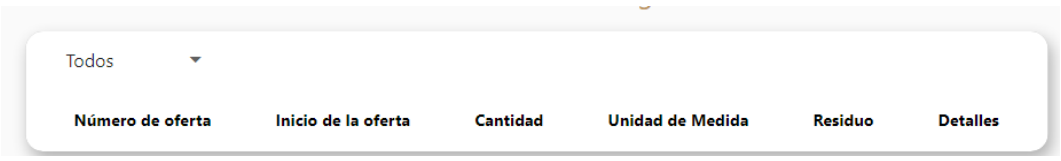

Figura 59: Consulta de las ofertas de residuos establecidos por el usuario generador.

La Figura 60 muestra la vista donde el usuario puede consultar los detalles de cada una de las ofertas y realizar cambios en caso de ser necesario.

Figura 60: Consulta de las ofertas de residuos sólidos ingresados por el usuario generador.

La Figura 61 muestra todas las ofertas registradas. Se consideró que el usuario pueda filtrar la vista por tipo de residuo para facilitar el uso de la misma. A manera de ejemplo, en la Figura 62, se muestra el detalle de la oferta seleccionada. En esta parte de la aplicación se permite hacer las modificaciones necesarias sobre la oferta.

Figura 61: Todas las ofertas registradas por el usuario generador.

Figura 62: Detalle de cada oferta registrada por el usuario generador.

Por otra parte, con respecto al registro de las ofertas de servicios por parte del prestador de servicios, se presenta una interfaz gráfica similar a la oferta de residuos, con los respectivos tipos de servicio que se desea ofertar. La Figura 63 muestra dicho registro de ofertas de servicios.

Figura 63: Aplicación web para el registro de ofertas de servicios.

Caso similar al anterior, el prestador de servicios tiene una vista de todas las ofertas de servicios ingresados. La Figura 64 corresponde a esta vista general.

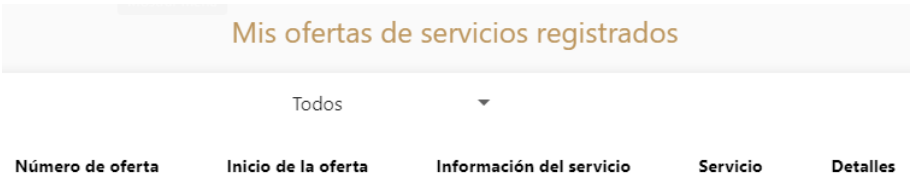

Figura 64: Consulta de las ofertas de servicios.

4.4 Síntesis

En este capítulo se presentaron los resultados obtenidos a través de la propuesta de solución. Para esto se utilizó el lenguaje de programación de Go utilizando varias librerías. Principalmente, gqlgen para el manejo de la API de GraphQL, y gorm para interactuar con la base de datos. Mediante el desglose de las APIs se presentó la variedad de opciones para la manipulación de datos obtenidos desde la base de datos en PostgreSQL, y las consultas realizadas desde el lado del cliente, desde la aplicación web construida en Flutter. A su vez, se presentaron algunas de las consultas y mutaciones, implementadas y su consumo del lado del cliente mediante la interfaz gráfica de usuario.

CAPÍTULO 5

Conclusiones y trabajo futuro

En el capítulo anterior se presentaron los resultados alcanzados, desglosando las API implementadas a través de GraphQL, y se presentaron algunas interfaces de usuario como parte del despliegue de la operación de la plataforma Web. En este capítulo se presentan las conclusiones generales y particulares; así como el trabajo futuro, asociadas a nuevas actividades que se podrían realizar.

5.1 Conclusiones generales

Es un hecho que la cantidad de residuos sólidos generados crece día con día; y se prevé que continúe aumentando como consecuencia del crecimiento demográfico. En la actualidad, el principal reto de la gestión de residuos sólidos urbanos es su reducción, reutilización y reciclaje. Actualmente, la mayor parte de los residuos sólidos urbanos son depositados en vertederos que provocan daños medioambientales, como la contaminación del suelo, el aire y el agua. Para disminuir este impacto ambiental existen variados esfuerzos, algunos aislados, que tienen como propósito mejorar la eficacia de la gestión de los residuos sólidos urbanos y la idoneidad de la recuperación energética.

En la Ciudad de México se tiene normativas que regulan el manejo, transporte, disposición y reciclaje de los residuos sólidos urbanos, como la Licencia Ambiental Única, RAMIR, Plan de Manejo, entre otros. Sin embargo, varias de estas normativas no son conocidas. Por otro lado, se ha visto que las tecnologías de gestión de residuos sólidos son rentables, ecológicas y socialmente aceptables. Por lo que, la gestión de residuos tiene varios beneficios, como la reducción de las emisiones de gases de efecto invernadero, la disminución de los residuos, la generación de ingresos por la venta de energía y la reutilización de los materiales de desecho.

Aunando en lo anterior, es importante fomentar utilizar este tipo de tecnologías. El trabajo de tesis realizado forma parte del desarrollo de una aplicación web, denominada plataforma SEDEMA, la cual permite vincular a los actores en la gestión de los residuos sólidos urbanos de la Ciudad de México, esto es, los generadores, los transportistas y los prestadores de servicios para el tratamiento y disposición de los RSU. En la actualidad, es importante seguir tomando medidas para mitigar la contaminación ambiental, y tratar en lo posible de salvaguardar el medio ambiente y la salud de las personas, antes de seguir sufriendo consecuencias que pueden ser nocivas en la población.

Desde el punto de vista computacional, a través de este trabajo de tesis se logró implementar unas APIs basadas en GraphQL para la consulta y manipulación de datos de la aplicación web. Se logró conectar de manera exitosa la capa de la base de datos, el servidor (Back-End) desarrollado en Go y el cliente (Front-End) implementado en Flutter, con base en las necesidades y especificaciones por parte del usuario SEDEMA.

5.2 Conclusiones particulares

El desarrollo de las APIs basabas en GraphQL permitió implementar de manera exitosa funciones de consulta y manipulación de datos de los catálogos generales, de procesos, de referencia y de interfaz que forman parte de la aplicación web. Con base en esto se proporcionó una importante variedad de funciones útiles para el despliegue y llenado de formularios que desde el cliente (Front-End) se pueden manejar.

Por otra parte, mediante la implementación del servidor de GraphQL, se logró agilizar las solicitudes de la gestión de datos del lado del cliente, dado que mediante una sola solicitud, la aplicación web es capaz de traer varios datos de diferentes tablas, consiguiendo aligerar la obtención de datos almacenados en la base de datos en PostgreSQL.

El uso de gqlgen, que es una biblioteca de Go para construir servidores GraphQL, permitió la generación de esquemas en la implementación de las consultas y mutaciones de los catálogos desarrollados, logrando así cumplir con el objetivo de crear funcionalidades, del lado del servidor (Back-End) dentro de la aplicación web, diseñada para la gestión de los residuos sólidos urbanos en la Ciudad de México.

En lo particular, con respecto a la codificación de las funcionalidades de las APIs, esta labor implicó importantes desafíos, como acortar la curva de aprendizaje para el uso de las nuevas tecnologías en el desarrollo web, como Go, GhaphQL, y Flutter. Además, se tuvie-

ron que explorar nuevas bibliotecas y frameworks, como gqlgen y gorm, para su uso en el desarrollo de la aplicación web.

Otro de los retos, a nivel equipo de desarrollo, fue estabilizar la implementación de la base de datos en PostgreSQL. En primera instancia el diseño y desarrollo fue rápido. Sin embargo, posterior a su implementación en el servidor, se tuvieron que hacer diversos cambios y ajustes debido a los requerimientos y necesidades de software solicitados por el equipo de la SEDEMA, lo que implicó que el trabajo sea dinámico entre los equipos de desarrollo dedicados a los dos servicios: Back-End y Front-End.

Finalmente, realizar este tipo de trabajos de tesis originó una importante motivación debido a: i) la importancia de tener un sistema de gestión de los RSU en la Ciudad de México; ii) la necesidad de dar a los generadores opciones para cumplir con las normas regulativas con el ambiente; iii) apoyar en el inventario y estado actual de los RSU de la Ciudad de México; y iv) tratar de aprovechar al máximo los materiales para ser reutilizados por otras empresas.

5.3 Trabajo futuro

Si bien los resultados obtenidos fueron provechosos, las funciones que se presentaron fueron útiles para su uso del Front-End; sin embargo, existen posibles mejoras que se podrían hacer en la aplicación web como parte del trabajo futuro, entre los que destacan:

- Hacer mejoras en la base de datos para incluir evidencia fotográfica u otros archivos como parte de la generación de los reportes (manifiestos), en el que se encadenan los actores: generador, transportista y reciclador. Para esto se propone que cada archivo o imagen sea almacenado con previo encriptado de su información, y que la referencia a su ubicación sea guardada en forma de campos de caracteres en la base de datos.
- Integrar nuevas funciones que no se hayan contemplado hasta la fecha, dado que al ser un sistema inédito para la SEDEMA, es probable que se requieran mejoras, posterior a su utilización por los usuarios finales.
- De la mano del desarrollo de nuevas funciones, se puede seguir optimizando el código desarrollado para mejorar la eficiencia de las consultas y mutaciones existentes. Esto permitirá tener un sistema confiable y con tiempos de respuesta cada vez menor.

APÉNDICE A

A continuación se desglosan los campos y tipos de datos para cada una de las tablas de la base de datos.

A.1 Catálogos generales

CG_USUARIOS_TIPO	id_tipo_usuario	integer
	tipo_usuario	varchar(100)
	descripcion	varchar(250)
CG_ESTADOS	id_estado	integer
	nombre_estado	varchar(100)
	clave_estado	text
CG_COLONIAS	id_colonia	integer
	nombre	varchar(250)
	nombre_corto	text
	nombre_coloquial	text
	id_cp_interno	integer
	cp_str	varchar(5)
CG_ALCALDIA_MUNICIPIOS	id_alcaldia_municipio	integer
	nombre	varchar(250)
	nombre_corto	varchar(50)
	nombre_coloquial	varchar(250)
CG_CALLES	id_calle	integer
	nombre_calle	varchar(250)
	nombre_corto	varchar(50)
	nombre_coloquial	varchar(100)
	nota_ubicacion	varchar(250)
CG_CODIGOS_POSTALES	id_estado	bigint
	id_alcaldia_municipio	bigint
	id_cp_interno	integer
	id_cp	varchar(5)

A.2 Catálogos de referencia

SERVICIOS	id_servicio	integer
	descripcion_servicio	varchar(250)
	tipo_servicio	varchar(100)
	requiere_permiso	boolean
	ctl_es_registro_activo	boolean
	ctl_fecha_registro	timestamp without time zone
	ctl_usuario_registro	varchar(100)
	ctl_fecha_modificacion	timestamp without time zone
	ctl_usuario_modificacion	integer
RESIDUOS_AUTORIZADOS	id_residuo_autorizado	integer
	nombre_residuo	varchar(100)
	unidad_residuo	varchar(100)
	cantidad_residuo	numeric
	referencia_norma_lau	varchar(250)
	nota_descripcion	varchar(250)
	clave_residuo_autorizado	varchar(100)
	ctl_es_registro_activo	boolean
	ctl_fecha_registro	timestamp without time zone
	ctl_usuario_registro	varchar(100)
	ctl_fecha_modificacion	timestamp without time zone
	ctl_usuario_modificacion	integer
	tiene_lau	boolean
	tiene_ramir	boolean
	tiene_planmanejo	boolean
	tiene_planmanejobienes	boolean
	tiene_otro	boolean
ESTATUS_PROCESOS	id_estatus	varchar(20)
	descripcion	varchar(250)
	codigo_estatus	integer

A.3 Catálogos de interfaz

EMPRESAS	id_empresa	integer
	razon_social	varchar(250)
	rfc_empresa	varchar(100)
	es_matriz_o_surcursal	integer
	id_empresa_global	integer
	nombre_empresa_sucursal	varchar(250)
	represente_gerente_contacto	varchar(250)
	horario_atencion	varchar(100)
	notas_descripcion	varchar(250)
	numreg_lau_cmdx	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
	ctl_fecha_modificacion	timestamp
	ctl_usuario_modificacion	integer
	rolempresa_es_solicitante	boolean
	rolempresa_es_generador	boolean
	rolempresa_es_prestadorservicios	boolean
	tipoempresa_es_personafisica	boolean
	tipoempresa_es_personamoral	boolean
USUARIOS	id_usuario	integer
	id_tipo_usuario	integer
	id_empresa	integer
	puesto_rol_usarlo_empresa	varchar(100)
	nombres	varchar(250)
	apellido1	varchar(250)
	apellido2	varchar(250)
	telefono_fijo	varchar(100)
	telefono_celular	varchar(50)
	email	varchar(100)
	password	varchar(256)
	email_confirmado	boolean
	telefono_confirmado	boolean
	ctl_es_registro_activo	boolean
	ctl_fecha_registro	timestamp
	ctl_usuario_registro	varchar(100)
	ctl_fecha_modificacion	timestamp
	ctl_usuario_modificacion	integer

DOMICILIOS	id_domicilio	integer
	id_colonia	integer
	id_calle	integer
	numero_exterior	varchar(50)
	numero_interior	varchar(50)
	referencia_domicilio	varchar(250)
	nota_domicilio	varchar(250)
	nota_descripcion	varchar(250)
	latitud	varchar(50)
	longitud	varchar(50)
	url_googlemaps	varchar(250)
	ctl_es_registro_activo	boolean
	ctl_fecha_registro	timestamp
	ctl_usuario_registro	varchar(100)
	ctl_fecha_modificacion	timestamp
	ctl_usuario_modificacion	integer
EMPRESAS_DOMICILIOS	id_empresas_domicilios	integer
	id_empresa	integer
	id_domicilio	integer
	tipo_domicilio	varchar(100)
	nota_descripcion	varchar(250)
	ctl_es_registro_activo	boolean
	ctl_fecha_registro	timestamp
	ctl_usuario_registro	varchar(100)
	ctl_fecha_modificacion	timestamp
	ctl_usuario_modificacion	integer
EMPRESAS_IDENTIFICADORES	id_empresa_identificador	integer
	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

SERVICIOS_EMPRESAS	id_servicios_empresa	integer
	id_empresa	integer
	id_servicio	integer
	num_tarjeton_ambiental	varchar(100)
	num_tarjeton_ambiental_fechavigencia	date
	transportista_ramir_vigencia	date
	ctl_es_registro_activo	boolean
	ctl_fecha_registro	timestamp
	ctl_usuario_registro	varchar(100)
	ctl_fecha_modificacion	timestamp
	ctl_usuario_modificacion	integer
	normativa	varchar(50)
TRANSPORTISTAS_TIPO_VEHICULOS	id_tipo_vehiculo	integer
	tipo_vehiculo	varchar(100)
	descripcion	varchar(250)
	ctl_es_registro_activo	boolean
	ctl_fecha_registro	timestamp
	ctl_usuario_registro	varchar(100)
	ctl_fecha_modificacion	timestamp
	ctl_usuario_modificacion	integer
TRANSPORTISTAS_VEHICULOS	id_vehiculo	integer
	id_transportista	integer
	id_tipo_vehiculo	integer
	num_placa	varchar(50)
	marca	varchar(100)
	modelo	varchar(100)
	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
RESIDUOS_AUTORIZADOS_EMPRESAS	id_residuo_autorizado	integer
	nota_descripcion	varchar(250)
	ctl_es_registro_activo	boolean
	ctl_fecha_registro	timestamp
	ctl_usuario_registro	varchar(100)
	ctl_fecha_modificacion	timestamp
	ctl_usuario_modificacion	integer
	nombre_especifico_residuo	varchar(150)
	id_servicio_empresa	integer

VISITANTES	id_visitante	integer
	nombre	varchar(100)
	primer_apellido	varchar(75)
	segundo_apellido	varchar(75)
	email	varchar(50)
	telefono	varchar(25)
	sector	varchar(25)
	estado	varchar(25)
	alcaldia	varchar(50)
	fecha_registro	timestamp

APÉNDICE B

A modo demostrativa se incluyen las operaciones creadas para otros catálogos de la API de GraphQL.

B.1 Catálogos generales

B.1.1. cg_alcaldia_municipios

(GetAllAlcaldiasMunicipios, GetAlcaldiaMunicipio). APIs codificadas para las consultas de alcaldías o municipios, según sea el caso, donde mediante GetAllAlcaldiasMunicipios (Figura 65) se obtiene la lista de alcaldías relacionadas con una determinada entidad federativa, en este caso Ciudad de México. Mientras que con GetAlcaldiaMunicipio (Figura 66) se recupera a través del identificador (ID) o el nombre de una determinada alcaldía, por ejemplo, (90007) Iztapalapa.

Figura 65: API para la consulta de alcaldías o municipios.

Figura 66: API para la consulta de una determinada alcaldía o municipio.

B.1.2. cg_colonias

(GetColonia y GetInfoCodigoPostal). APIs codificadas para la consulta de colonias, tal como se observa en la (Figura 67), donde por medio del nombre de la colonia '10 de Abril' se obtiene la información de todas las colonias con ese nombre. En este caso al código postal 11411, ubicado en la alcaldía Miguel Hidalgo (ID 90016) de la Ciudad de México. Mientras que con GetInfoCodigoPostal se obtienen todas las colonias asociadas con un mismo código postal.

Figura 67: API para la consulta de colonias.

B.2 Catálogos de referencia

B.2.1. cpr_servicios

(GetAllServicios, GetServicio). API utilizada para la consulta de servicios ofrecidos por los prestadores de servicios. En la Figura 68 se observa que se utiliza la consulta de GetAll-Servicios para obtener la información de todos los servicios. La revisión se basó en la unión de roles y servicios en una misma tabla.

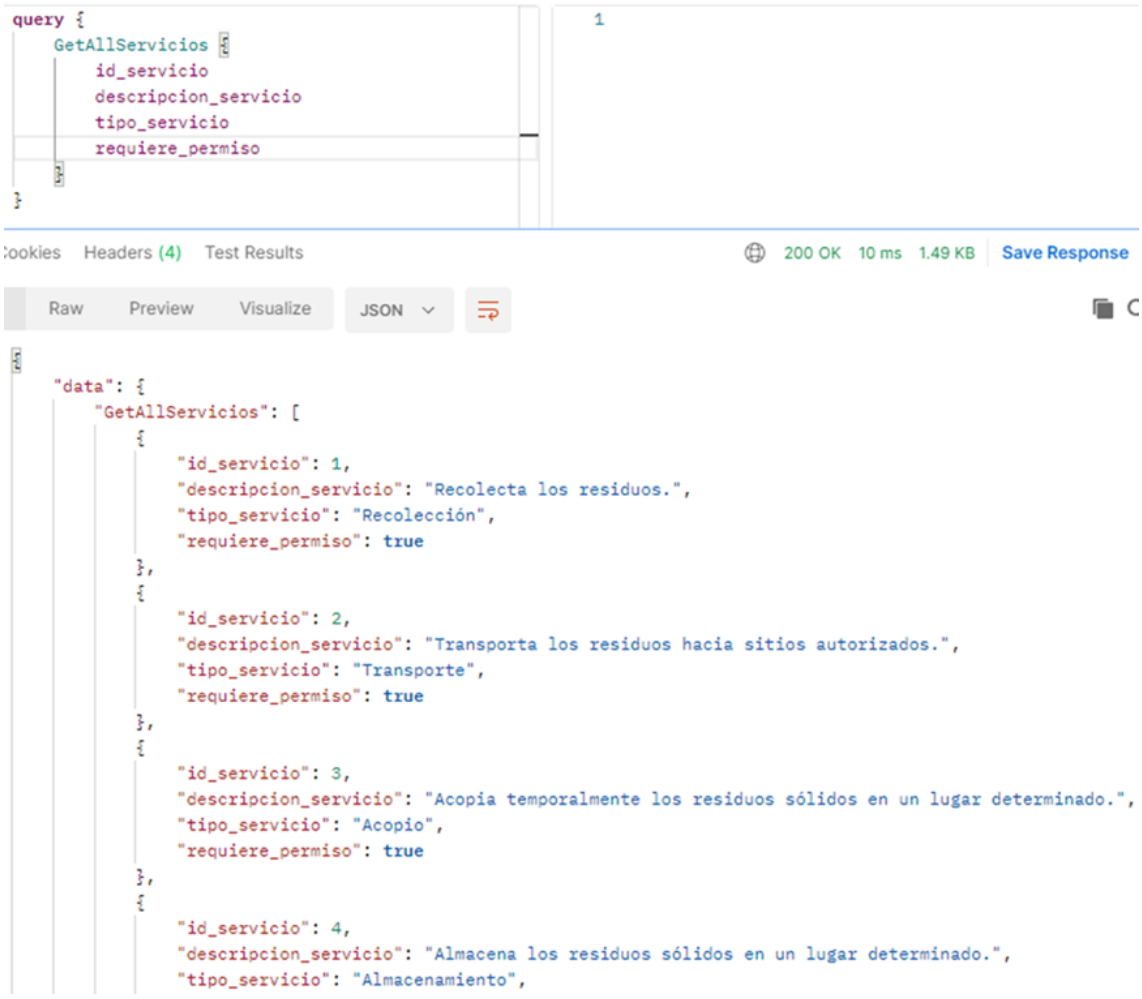

Figura 68: API utilizada para la consulta de servicios ofrecidos por los prestadores de servicios.

B.2.2. cpr_residuos_autorizados

(GetResiduosAutorizados, GetAllResiduosAutorizados). API utilizada para la consulta de residuos autorizados de acuerdo al listado proporcionado. La revisión consta de la inclusión de cuatro campos que indican si el residuo está listado para cada una de las normativas o está excluido de alguno. De igual manera, se indica el tipo de residuo. La Figura 69 muestra que se requieren todos los residuos que pertenezcan a RAMIR.

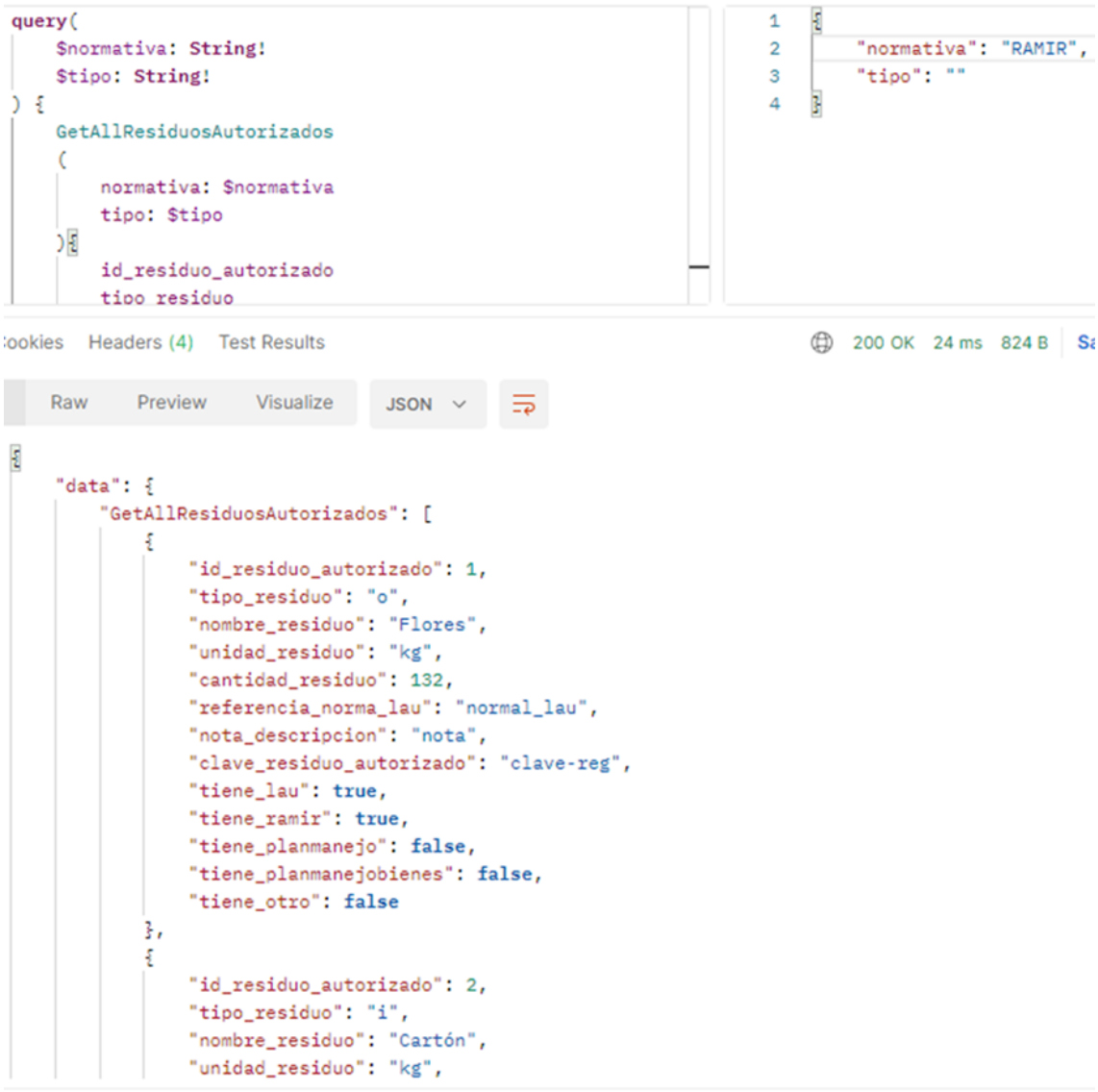

Figura 69: API utilizada para la consulta de residuos autorizados de acuerdo al listado proporcionado.

B.3 Catálogos de interfaz

B.3.1. cpi_usuarios

(UpdatePassword, RegistroUsuario). APIs utilizadas para las operaciones CRUD (Creación, Lectura, Actualización y Eliminación) de la tabla cpi_usuarios. La Figura 70 muestra la creación de un usuario de nombre Alejandro Martínez. Mientras que Figura 71 muestra la actualización de la contraseña de un usuario.

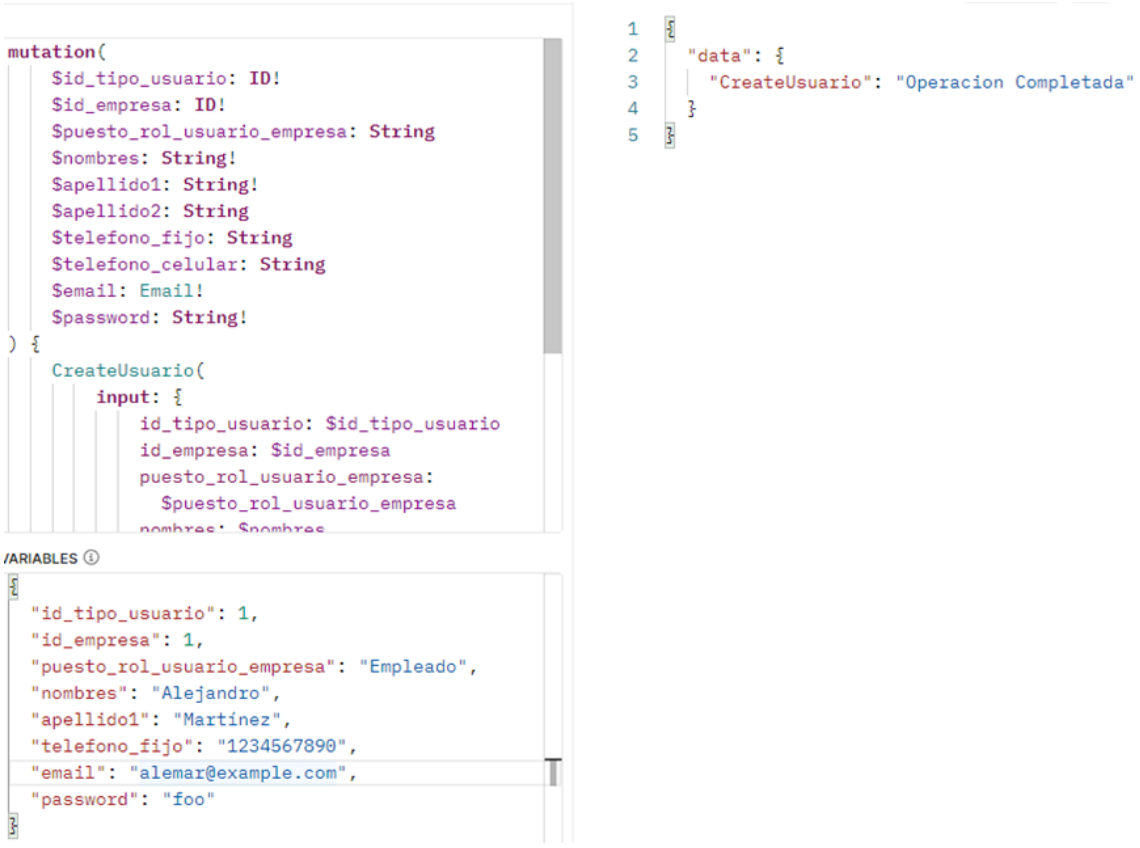

Figura 70: API para la creación de un usuario.

Figura 71: API para la actualización de la contraseña de un usuario.

B.4 Catálogos de procesos

B.4.1. p1_solicitudes

(CreateSolicitud, GetSolicitud, GetSolicitubByResiduo, GetSolicitudByServicio, UpdateSolicitud, DeleteSolicitud). APIs utilizadas para la consulta y creación de solicitudes. En la Figura **??** se observa la creación de una solicitud de cartón de 5000 kilos. De igual manera, en la Figura 73 se solicita la información de una solicitud previa, que tiene como identificador único '12'.

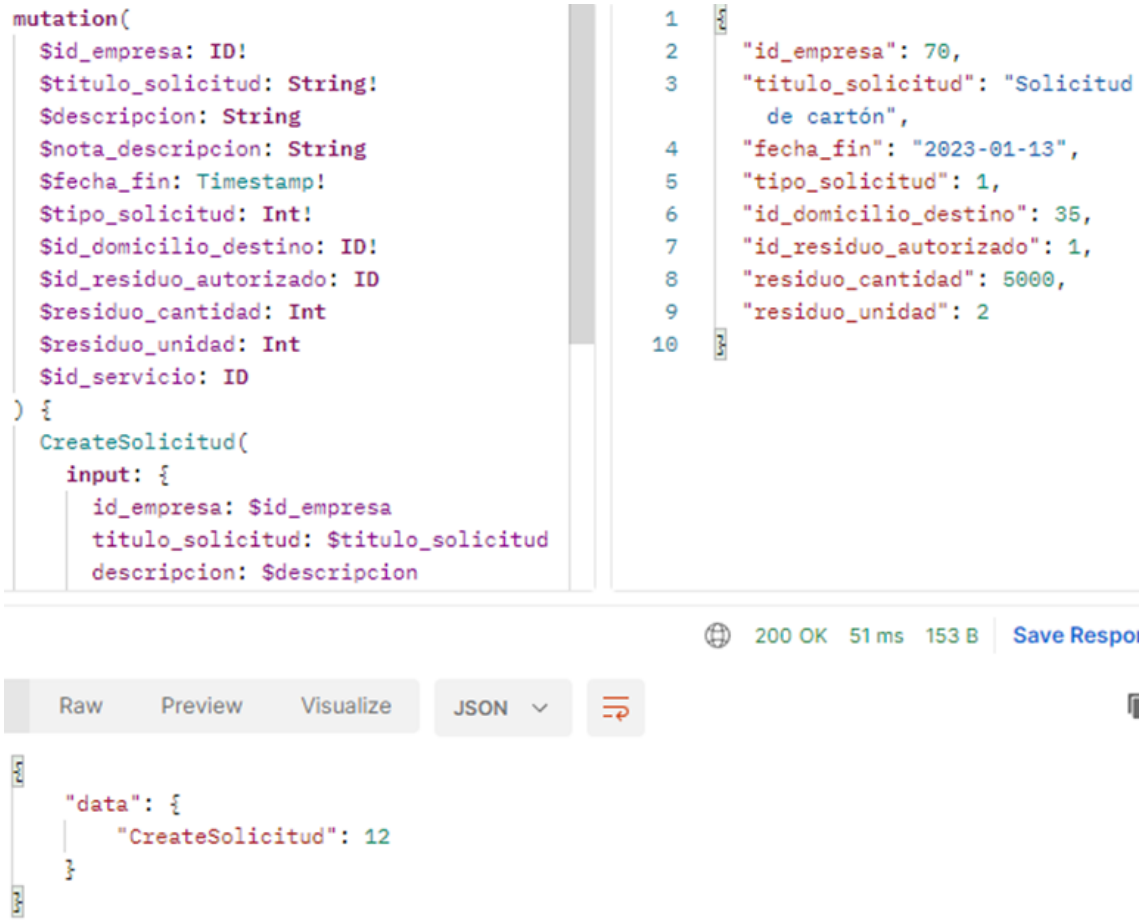

Figura 72: API para la creación de una solicitud.

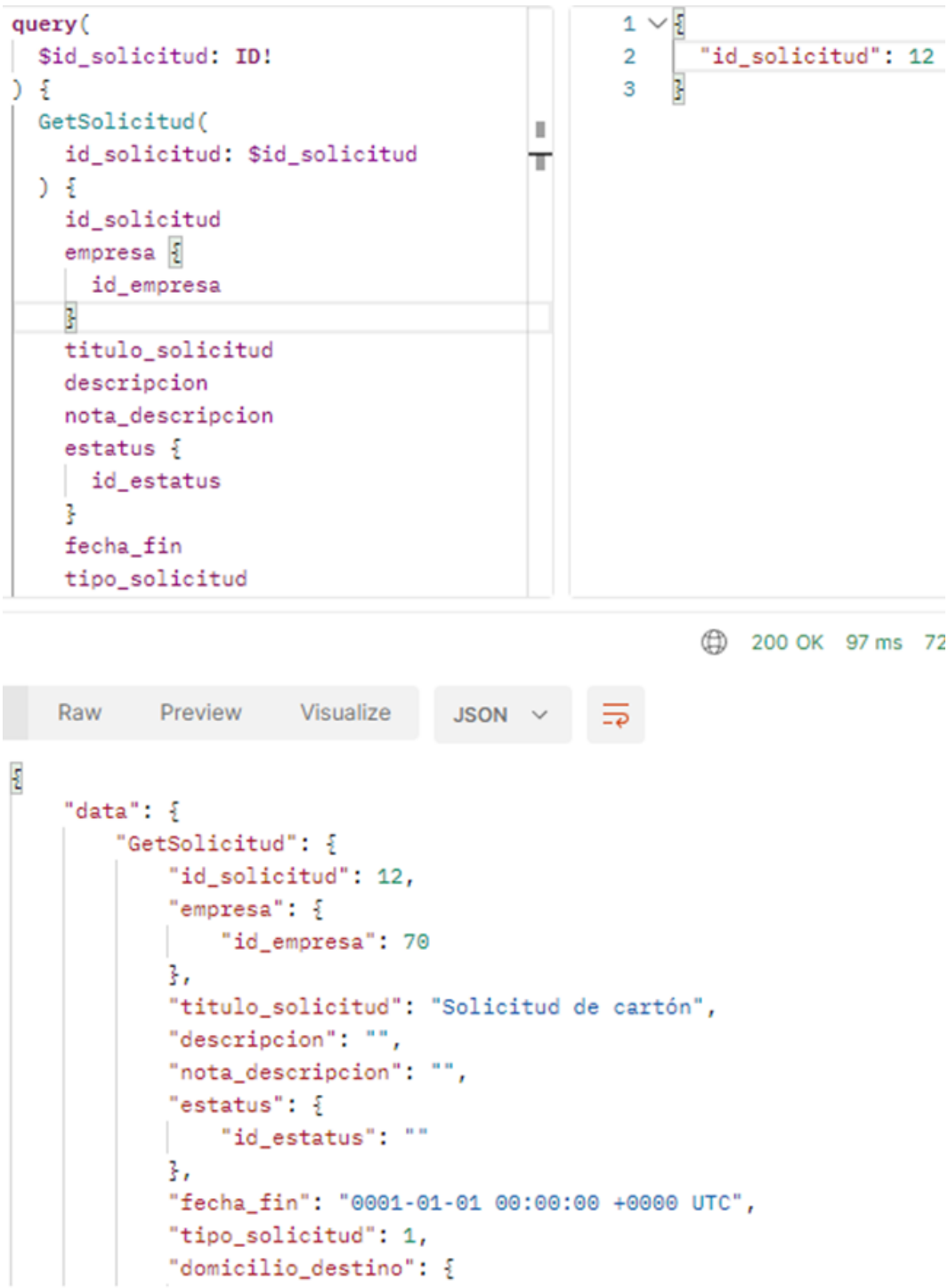

Figura 73: API para la solicitud de información de una solicitud previa.

Bibliografía

Agenda, I. (2016). The New Plastics Economy Rethinking the future of plastics. *World Economic Forum*, 36 (vid. pág. 12).

AIR, E., EDISTRIBUTE, R. & REFURBISH, R. (2016). Circular economy in Europe (vid. pág. 21).

Arias, P., Bellouin, N., Coppola, E., Jones, R., Krinner, G., Marotzke, J., Naik, V., Palmer, M., Plattner, G.-K., Rogelj, J. et al. (2021). Climate Change 2021: The Physical Science Basis. Contribution of Working Group14 I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change; Technical Summary (vid. pág. 10).

Botello-Álvarez, J. E., Rivas-Garcíia, P., Fausto-Castro, L., Estrada-Baltazar, A. & Gomez-Gonzalez, R. (2018). Informal collection, recycling and export of valuable waste as transcendent factor in the municipal solid waste management: A Latin-American reality. *Journal of cleaner production*, *182*, 485-495 (vid. pág. 34).

Cantú Martínez, P. C. (2014). Cambio climático: sus repercusiones para la sustentabilidad. *Ciencia UANL*, *17*(67), 31-36 (vid. pág. 10).

Cerdá, E. & Khalilova, A. (2016). Economía circular. *Economía industrial*, *401*(3), 11-20 (vid. pág. 12).

Choudhury, N. (2014). World wide web and its journey from web 1.0 to web 4.0. *International Journal of Computer Science and Information Technologies*, *5*(6), 8096-8100 (vid. pág. 22).

DE, C. Ú. O. Y. Á. (2018). Ley general para la prevención y gestión integral de los residuos (vid. pág. 11).

de la Ciudad de México, G. (2021). *Manual de Identidad Institucional 2021-2024*. Consultado el 20 de julio de 2022, desde https://jefaturadegobierno.cdmx.gob.mx/storage/app/uploads/public/611/1a8/ad4/6111a8ad425fd683582497.pdf. (Vid. pág. 38)

Díaz Cordero, G. (2012). El cambio climático. *Ciencia y sociedad* (vid. pág. 10).

Docker. (2022). *What is a Container? - Docker*. Consultado el 4 de mayo de 2022, desde https://www.docker.com/resources/what-container/. (Vid. págs. 26, 27)

Documentation, F. (2022). Flutter widget index. (Vid. pág. 25).

Engineering, I. (2016). Instagration Pt. 2: Scaling our infrastructure to multiple data centers. (Vid. pág. 24).

Foundation, E. M. (2015). Towards a circular economy: Business rationale for an accelerated transition. (Vid. pág. 12).

Gutierrez Galicia, F., Coria Paez, A. L. & Tejeida Padilla, R. (2019). A study and factor identification of municipal solid waste management in mexico city. *Sustainability*, *11*(22), 6305 (vid. págs. 33, 34).

Hotspot, H. C. (2021). Waste Management Country Report: Mexico. *Holland Circular Hotspot* (vid. pág. 34).

Liu, D., Guo, X. & Xiao, B. (2019). What causes growth of global greenhouse gas emissions? Evidence from 40 countries. *Science of The Total Environment*, *661*, 750-766 (vid. pág. 14).

Napoli, M. L. (2019). *Beginning Flutter: A Hands On Guide To App Development*. John Wiley & Sons. (Vid. pág. 25).

Nath, K., Dhar, S. & Basishtha, S. (2014). Web 1.0 to Web 3.0-Evolution of the Web and its various challenges. *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, 86-89 (vid. págs. 22, 23).

ONU. (2019). Se alcanzan niveles récord de concentración de gases de efecto invernadero en la atmósfera. (Vid. pág. 16).

Quintero Fernandez, A. E. (2018). *DISEÑO DEL SISTEMA INFORMÁTICO DE GESTIÓN DE RESIDUOS INSTITUCIONALES SIGRI* (Tesis doctoral). (Vid. págs. 33, 34).

Sandoval, V. P., Jaca, C. & Ormazabal, M. (2017). Economıía circular. *Memoria Investigaciones en Ingenierıía*, (15), 85-95 (vid. págs. 12, 21).

SEDEMA. (2019). Inventario de residuos sólidos de la Ciudad de México. (Vid. págs. 17, 18).

Sedesol. (2012). Informe 2012. (Vid. pág. 19).

Semarnat. (2015). Informe de la Situación del Medio Ambiente en México. Compendio de Estadísticas Ambientales. Indicadores Clave, de Desempeño Ambiental y de Crecimiento Verde. Edición 2015. (vid. pág. 10).

Shimel, A. (2011). Apple ditches mysql in favor of PostgreSQL. (Vid. pág. 24).

Statista Research Department, F. (2022). Most popular database management systems 2022. (Vid. pág. 23).

Zeng, X., Sun, Q., Huo, B., Wan, H. & Jing, C. (2010). Integrated solid waste management under global warming. *The Open waste management journal*, *3*(1) (vid. pág. 11).