

---

# Aplicación móvil para la gestión de residuos y reducción de la huella de carbono

---

*Backend, gestión de base de datos y análisis de datos*

---

Oscar Fernando López Barrios



UNIVERSIDAD DEL VALLE DE GUATEMALA  
Facultad de Ingeniería



**Aplicación móvil para la gestión de residuos y  
reducción de la huella de carbono**

*Backend, gestión de base de datos y análisis de datos*

Trabajo de graduación en modalidad de Tesis presentado por  
Oscar Fernando López Barrios

Para optar al grado académico de Licenciado en Ingeniería en Ciencias  
de la Computación y Tecnologías de la Información

Guatemala, noviembre del 2024

Vo.Bo.:

(f) \_\_\_\_\_  
Ing. Héctor Hurtarte

Tribunal Examinador:

(f) \_\_\_\_\_  
Ing. Héctor Hurtarte

(f) \_\_\_\_\_  
Ing. Mario Barrientos

(f) \_\_\_\_\_  
Ing. Eddy Castro

Fecha de aprobación: Guatemala, 28 de noviembre de 2024.

---

## Prefacio

---

Siendo originario del departamento de Retalhuleu y habiéndome mudado a la Ciudad de Guatemala, para mí ha sido evidente el contraste en el entorno ambiental en el que vivimos.

Esta situación resulta preocupante, ya que, con el paso del tiempo, la problemática de la gestión de residuos en una ciudad densamente poblada representa un gran desafío que debe abordarse, además de la necesidad de generar mayor conciencia entre los ciudadanos. Esto es especialmente relevante, dado que experimenté esta problemática de forma cotidiana en mi ciudad natal, donde la menor densidad de población hacía que el problema de la basura fuera a menudo pasado por alto.

Este contexto preocupante es el que nos ha inspirado, como grupo de Megaproyecto, a presentar una solución adecuada que permita a las personas tener un mejor control sobre los desechos que generan. A través de esta aplicación, los usuarios podrán acceder a información sobre la clasificación correcta de residuos y comprender el impacto ambiental de sus acciones responsables mediante el análisis de sus datos.

Al finalizar este Megaproyecto Tecnológico y haber realizado mi aporte de primera mano en el módulo de *Backend*, gestión de bases de datos y análisis de datos, me siento orgulloso de los logros alcanzados. Espero que esta iniciativa, presentada a través de este proyecto, sea utilizada para generar conciencia ambiental y que constituya una contribución importante para lograr un impacto positivo en el medio ambiente.

Agradezco a la Municipalidad de Guatemala por el apoyo brindado durante la realización de este proyecto. En especial, quiero expresar mi gratitud a la Unidad de Reciclaje y al Centro de Educación Ambiental por su orientación y dedicación en la presentación de esta solución para los habitantes de la ciudad.

Por último, quiero expresar mi más sincero agradecimiento al Ing. Héctor Hurtarte, mi asesor, por su invaluable apoyo y guía a lo largo de la realización de este proyecto. Su dedicación y conocimientos han sido fundamentales para el desarrollo de cada etapa, brindándome orientación tanto técnica como profesional. Agradezco especialmente su paciencia y confianza en mi trabajo.

---

## Agradecimientos

---

Este trabajo de graduación está dedicado a todas las personas que han estado conmigo a lo largo de mi vida y carrera universitaria.

Agradezco de todo corazón a cada una de las personas que me han acompañado en este arduo camino, especialmente a mis padres, Oscar y Norma, a mi hermana Angélica, a mis abuelitos y a toda mi familia. Cada uno de ellos ocupa un lugar especial en mi corazón y en mi memoria. También agradezco a quienes ya no se encuentran entre nosotros, pero que, a través de sus enseñanzas, dejaron una huella imborrable en mi vida. Durante mi proceso de crecimiento, cada una de estas personas ha contribuido a formar la persona y los valores que represento hoy en día.

Agradezco profundamente a Dios, quien siempre me ha brindado sabiduría y fortaleza en el camino de mi carrera universitaria, la cual ha sido una gran aventura llena de alegrías y desafíos.

También quiero expresar mi gratitud a cada uno de los amigos que he encontrado en cada etapa de mi vida. Agradezco su apoyo y aliento para alcanzar mi meta, y los llevo en el corazón: mis amigos de Retalhuleu, mis amigos de la Resi, mi novia y, en especial, mis amigos de la Universidad, con quienes compartimos muchas experiencias gratas. Agradezco de manera especial a Santiago, Pedro, Yong y Gabo; gracias a ustedes y a sus ideas logramos llevar a cabo este trabajo de graduación. Sin su compañía, estos cinco años de carrera universitaria no habrían sido tan maravillosos como fueron.

Quiero expresar un agradecimiento especial a mi asesor, Ing. Héctor Hurtarte, por la confianza que depositó en mí para la realización de este proyecto, por su guía y apoyo durante todo el proceso, y por sus enseñanzas tanto en la Universidad como en el ámbito profesional. Cada una de ellas ha sido de gran ayuda para mi desarrollo como futuro ingeniero.

Por último, quisiera agradecer a la Fundación Juan Bautista Gutiérrez por confiar en mí desde el primer día y brindarme la gran oportunidad de estudiar en la Universidad del Valle. Agradezco de corazón a cada una de las personas que formaron parte de mi proceso, en especial a mis coordinadoras, quienes me enseñaron importantes lecciones personales que han dejado un gran impacto en mi vida. Agradezco profundamente a doña Isabelita (Q.E.P.D.) por haber confiado en mí y haberme brindado su apoyo.

---

## Índice

---

<b>Prefacio</b>	<b>III</b>
<b>Agradecimientos</b>	<b>IV</b>
<b>Lista de Figuras</b>	<b>IX</b>
<b>Lista de Cuadros</b>	<b>X</b>
<b>Resumen</b>	<b>XI</b>
<b>Abstract</b>	<b>XII</b>
<b>1. Introducción</b>	<b>1</b>
<b>2. Objetivos</b>	<b>2</b>
2.1. Objetivo General . . . . .	2
2.2. Objetivos Específicos . . . . .	2
<b>3. Justificación</b>	<b>3</b>
<b>4. Marco Teórico</b>	<b>5</b>
4.1. Backend . . . . .	5
4.1.1. ¿Qué es un Backend? . . . . .	5
4.1.2. Componentes de un Backend . . . . .	5
4.1.3. Backends más Comunes . . . . .	5
4.2. Tecnologías para el Desarrollo de Aplicaciones Escalables . . . . .	6
4.2.1. Introducción a la Escalabilidad en Aplicaciones Móviles . . . . .	6
4.2.2. Tecnologías Actuales para el Desarrollo de Aplicaciones Escalables . . . . .	6
4.2.3. Factores para la Selección de Tecnologías Adecuadas en Proyectos Móviles . . . . .	8
4.3. Bases de Datos en la Gestión de Residuos y Clasificación de Desechos . . . . .	8
4.3.1. Diseño de Bases de Datos para Aplicaciones de Gestión de Residuos . . . . .	8
4.3.2. Normalización de Bases de Datos: Asegurando la Eficiencia y Consistencia . . . . .	9
4.3.3. Gestión de Transacciones: Principios ACID y su Aplicación . . . . .	10
4.4. Desarrollo de una API . . . . .	11
4.4.1. Introducción a la Arquitectura Cliente-Servidor . . . . .	11
4.4.2. Principios de Diseño de una API Restful para Aplicaciones Móviles . . . . .	11
4.4.3. Integridad de los Datos Transmitidos: Mecanismos de Validación y Seguridad	12
4.5. Análisis de Datos Generados por los Usuarios . . . . .	12

4.5.1. Introducción al Análisis de Datos en la Gestión de Residuos . . . . .	12
4.5.2. Técnicas para Procesar y Analizar Datos . . . . .	12
4.5.3. Métricas para la Reducción de la Huella de Carbono . . . . .	13
4.6. Escalabilidad y Rendimiento en el Sistema de Gestión de Residuos . . . . .	13
4.6.1. Optimización de Consultas y Operaciones CRUD . . . . .	13
4.6.2. Pruebas de Carga y Rendimiento en el Backend . . . . .	13
4.7. CI/CD: Integración y Entrega Continua . . . . .	14
4.7.1. Pruebas . . . . .	14
4.7.2. Monitoreo . . . . .	14
4.7.3. Automatización . . . . .	14
4.8. Tecnologías Utilizadas en el Proyecto . . . . .	15
4.8.1. Elección de Tecnologías para el Backend . . . . .	15
4.8.2. Integración entre Express.js y PostgreSQL . . . . .	16
4.8.3. Herramientas de Análisis de Datos . . . . .	16
4.9. Impacto Social y Ambiental de la Gestión de Residuos . . . . .	16
4.9.1. La Tecnología como Herramienta de Educación Ambiental . . . . .	16
4.9.2. Reducción de la Huella de Carbono a Través de la Gestión de Residuos . . . . .	17
4.9.3. La Gestión de Residuos como Estrategia para la Reducción del Impacto Ambiental . . . . .	17
4.9.4. Clasificación de Residuos y Educación Ambiental a través de la Tecnología . . . . .	17
<b>5. Metodología</b> . . . . .	<b>19</b>
5.1. Elección de Tecnologías . . . . .	19
5.2. Primera Iteración . . . . .	20
5.2.1. Base de Datos: Diseño versión inicial . . . . .	20
5.2.2. API: Definición de las funciones . . . . .	21
5.3. Segunda Iteración . . . . .	22
5.3.1. Base de Datos: Diseño de tablas para ubicación y medición . . . . .	22
5.3.2. API: Creación de funciones CRUD para ubicaciones y mediciones . . . . .	22
5.4. Tercera Iteración . . . . .	23
5.4.1. Base de Datos: Diseño de tabla para la zona de la ciudad . . . . .	23
5.4.2. API: Creación de modelos y uso de ORM . . . . .	24
5.5. Cuarta Iteración . . . . .	25
5.5.1. Base de Datos: Cambios en mediciones de desechos . . . . .	25
5.5.2. API: Ampliación de controladores . . . . .	26
5.5.3. Análisis de Datos: Creación de métricas iniciales . . . . .	27
5.6. Quinta Iteración . . . . .	28
5.6.1. Base de Datos: Creación de tabla para tipos de desecho . . . . .	28
5.6.2. API: Implementación de autenticación con JWT . . . . .	29
5.6.3. Análisis de Datos: Diseño de funciones principales . . . . .	30
5.7. Sexta Iteración . . . . .	31
5.7.1. Base de Datos: Creación de tabla de bitácora de auditoría . . . . .	31
5.7.2. API: Implementación de registros en bitácora de auditoría . . . . .	32
5.7.3. Análisis de Datos: Diseño de fucniones de análisis . . . . .	33
5.8. Séptima Iteración . . . . .	34
5.8.1. Base de Datos: Optimización y mejora de la estructura . . . . .	34
5.8.2. API: Optimización y mejora del API . . . . .	35
5.8.3. Análisis de Datos: Creación de las rutas de análisis en API . . . . .	36
5.9. Octava Iteración . . . . .	36
5.9.1. Análisis de Datos: Creación de análisis de datos global de los usuarios . . . . .	36
5.10. Implementación de CI/CD . . . . .	38
5.10.1. Pruebas Automatizadas . . . . .	39
5.10.2. Manejo y Monitoreo de Errores con Sentry . . . . .	43
5.10.3. Implementación y Automatización con GitHub Actions . . . . .	45

<b>6. Resultados</b>	<b>47</b>
6.1. Diseño y Estructura de la Base de Datos . . . . .	47
6.2. API . . . . .	48
6.2.1. Resultados de Pruebas de Funcionalidad . . . . .	49
6.2.2. Pruebas Unitarias y de Integración . . . . .	51
6.2.3. Pruebas de Carga y Rendimiento . . . . .	53
6.3. Análisis de Datos . . . . .	63
6.4. Implementación de CI/CD . . . . .	66
6.4.1. Pruebas . . . . .	66
6.4.2. Monitoreo de Errores con Sentry . . . . .	69
6.4.3. Automatización con GitHub Actions . . . . .	71
<b>7. Análisis de Resultados</b>	<b>73</b>
7.1. Elección de las Tecnologías . . . . .	73
7.2. Diseño y Estructura de la Base de Datos . . . . .	73
7.3. API . . . . .	74
7.4. Análisis de Datos . . . . .	74
7.5. Implementación de CI/CD . . . . .	75
<b>8. Conclusiones</b>	<b>76</b>
<b>9. Recomendaciones</b>	<b>78</b>
<b>Bibliografía</b>	<b>82</b>
<b>Anexos</b>	<b>83</b>
<b>A. Pruebas de carga y rendimiento en API de producción</b>	<b>83</b>

---

## Lista de Figuras

---

4.1. Arquitectura cliente-servidor: comunicación entre frontend y backend [18]. . . . .	6
4.2. Diagrama de comunicación REST API. Fuente: [41]. . . . .	8
4.3. Diagrama de la base de datos. Elaboración propia. . . . .	9
4.4. Representación de la Segunda Forma Normal (Second Normal Form). Fuente: [46]. .	10
5.1. Diagrama entidad-relación de la primera iteración. . . . .	21
5.2. Diagrama entidad-relación de la segunda iteración. . . . .	22
5.3. Diagrama entidad-relación de la tercera iteración. . . . .	24
5.4. Diagrama entidad-relación de la cuarta iteración. . . . .	26
5.5. Reunión para Estructurar el Análisis de Datos. . . . .	28
5.6. Diagrama entidad-relación de la quinta iteración. . . . .	29
5.7. Diagrama entidad-relación de la sexta iteración. . . . .	32
5.8. Diagrama entidad-relación de la séptima iteración. . . . .	35
5.9. Ejemplo de solicitudes en origenes de conexión. . . . .	37
5.10. Tablas resultantes de la conexión del API. . . . .	38
5.11. Pipeline de CI/CD [13]. . . . .	38
5.12. Tablero del Proyecto en Sentry. . . . .	44
6.1. Diagrama entidad-relación final. . . . .	48
6.2. Registro de usuarios. . . . .	49
6.3. Login de usuarios. . . . .	50
6.4. Registro en la bitácora de auditoría. . . . .	50
6.5. Intento de acceso no autorizado. . . . .	50
6.6. Consulta de análisis para usuarios autorizados. . . . .	51
6.7. Tests realizados al API. . . . .	52
6.8. Resultado de los tests acertados. . . . .	53
6.9. Resultados prueba de carga del 20 %. . . . .	54
6.10. Resultados prueba de carga del 20 %. . . . .	54
6.11. Resultados prueba de carga del 20 %. . . . .	55
6.12. Resultados prueba de carga del 20 %. . . . .	56
6.13. Resultados prueba de carga del 50 %. . . . .	57
6.14. Resultados prueba de carga del 50 %. . . . .	57
6.15. Resultados prueba de carga del 50 %. . . . .	58
6.16. Resultados prueba de carga del 50 %. . . . .	59
6.17. Resultados de la prueba de carga para el 100 % de la población. . . . .	60
6.18. Resultados de la prueba de carga para el 100 % de la población. . . . .	60
6.19. Resultados de la prueba de carga para el 100 % de la población. . . . .	61

6.20. Resumen de resultados de la prueba de carga para el 100 %.	62
6.21. Resultado top 5 desechos usuario.	63
6.22. Resultado porcentaje comparación usuario.	64
6.23. Resultado porcentaje comparación usuario.	64
6.24. Resultado top 5 desechos generales.	65
6.25. Resultado desechos reciclables generales.	65
6.26. Resultado CO2 y Agua generales.	66
6.27. Resultado de los tests acertados.	67
6.28. Resultado del action de los tests acertados.	67
6.29. Resultado de los tests fallados.	68
6.30. Resultado del action de los tests fallados.	68
6.31. Resultado de los tests fallados en el mail.	69
6.32. Resultado de los errores de Sentry.	69
6.33. Resultado de la descripción de un error en Sentry.	70
6.34. Resultado una notificación de error.	70
6.35. Resultado del apartado de performance de Sentry.	70
6.36. Resultado del detalle de descripción de performance de Sentry.	71
6.37. Resultado de los trazos de Sentry.	71
6.38. Resultado de GitHub Actions.	72
6.39. Resultado del deploy.	72
A.1. Resultados prueba de carga del 50 %.	84
A.2. Resultados prueba de carga del 50 %.	85
A.3. Resultados prueba de carga del 50 %.	85
A.4. Resultados prueba de carga del 50 %.	86
A.5. Resultados prueba de carga del 50 %.	86
A.6. Resultados prueba de carga del 50 %.	87
A.7. Resultados prueba de carga del 50 %.	87
A.8. Resultados prueba de carga del 100 %.	88
A.9. Resultados prueba de carga del 100 %.	89
A.10. Resultados prueba de carga del 100 %.	89
A.11. Resultados prueba de carga del 100 %.	90

---

## Lista de Cuadros

---

4.1. Comparación entre bases de datos relacionales y no relacionales . . . . .	7
6.1. Resultados de pruebas de carga a diferentes tasas de solicitudes . . . . .	62
6.2. Comparación de rendimiento en tres escenarios de carga en API de producción . . .	63

---

## Resumen

---

Este proyecto tiene como finalidad desarrollar el módulo *backend* una aplicación móvil innovadora para mejorar la gestión de residuos en la Ciudad de Guatemala, incentivando a los usuarios a reducir su huella de carbono mediante prácticas sostenibles en sus hogares. La aplicación ofrece a los usuarios herramientas para clasificar correctamente sus desechos y realizar un seguimiento de su impacto ambiental, permitiéndoles adoptar hábitos de consumo más responsables.

Para lograr estos objetivos, se diseñó un módulo de backend robusto, encargado de la gestión, almacenamiento y análisis de los datos generados por los usuarios. La base de datos se estructura para almacenar información detallada sobre los tipos de residuos, su frecuencia de desecho y otros datos relevantes que permiten un análisis exhaustivo tanto a nivel individual como colectivo. Este diseño no solo facilita la administración de los registros, sino que también optimiza el rendimiento del sistema al permitir consultas rápidas y eficientes, incluso bajo una alta demanda de usuarios.

La seguridad y la integridad de los datos son pilares fundamentales del proyecto. Para proteger la información de los usuarios, se emplea autenticación JWT (JSON Web Token), que garantiza que solo personas autorizadas tengan acceso a la aplicación. Asimismo, se implementa una bitácora de auditoría que registra cualquier modificación en la información, asegurando la transparencia y permitiendo rastrear cambios. Las funcionalidades de la API, además, están diseñadas para una integración fluida con el frontend, permitiendo una comunicación segura y eficaz entre ambas partes.

En términos de análisis de datos, el sistema permite a los usuarios visualizar métricas personalizadas sobre sus hábitos de gestión de residuos, como la reducción de CO<sub>2</sub> y el ahorro de agua. Este análisis no solo proporciona retroalimentación útil para cada usuario, sino que también permite generar informes a nivel global, ofreciendo una visión amplia del impacto colectivo y apoyando a los administradores en la formulación de estrategias ambientales.

Para asegurar la calidad y escalabilidad del sistema, se implementó un pipeline de CI/CD que facilita un ciclo de desarrollo continuo, permitiendo que nuevas funcionalidades y mejoras se desplieguen rápidamente y sin interrupciones. Este enfoque de desarrollo también incluye pruebas automatizadas de funcionalidad, rendimiento y carga, lo que garantiza que la aplicación mantenga su eficiencia y estabilidad a largo plazo.

Finalmente, el monitoreo constante mediante la herramienta Sentry permite responder de forma ágil a cualquier problema en producción y realizar ajustes que optimicen el rendimiento del API. Con esta infraestructura, el sistema está preparado para futuras expansiones, como la incorporación de nuevas métricas ambientales y funcionalidades para usuarios administrativos, asegurando que el proyecto evolucione de acuerdo con las necesidades de los usuarios y las demandas de sostenibilidad.

---

## Abstract

---

This project aims to develop the *backend* module, an innovative mobile application to improve waste management in Guatemala City, encouraging users to reduce their carbon footprint through sustainable practices in their homes. The application provides users with tools to correctly classify their waste and track their environmental impact, enabling them to adopt more responsible consumption habits.

To achieve these objectives, a robust backend module was designed to manage, store and analyze the data generated by users. The database is structured to store detailed information on the types of waste, their frequency of disposal and other relevant data that allows for a comprehensive analysis at both the individual and collective level. This design not only facilitates record management, but also optimizes system performance by enabling fast and efficient queries, even under high user demand.

Security and data integrity are fundamental pillars of the project. To protect user information, JWT (JSON Web Token) authentication is used to ensure that only authorized persons have access to the application. An audit log is also implemented to record any modification to the information, ensuring transparency and allowing changes to be tracked. The API functionalities are also designed for seamless integration with the frontend, allowing secure and efficient communication between both parties.

In terms of data analytics, the system allows users to visualize customized metrics on their waste management habits, such as CO reduction and water savings. This analysis not only provides useful feedback for each user, but also enables reporting at a global level, providing a broad view of the collective impact and supporting managers in formulating environmental strategies.

To ensure the quality and scalability of the system, a CI/CD pipeline was implemented that facilitates a continuous development cycle, allowing new features and enhancements to be deployed quickly and without interruption. This development approach also includes automated functionality, performance and load testing, ensuring that the application maintains its efficiency and stability over the long term.

Finally, constant monitoring using the Sentry tool enables agile response to any issues in production and adjustments to optimize API performance. With this infrastructure in place, the system is prepared for future expansions, such as the incorporation of new environmental metrics and functionalities for administrative users, ensuring that the project evolves according to user needs and sustainability demands.

# CAPÍTULO 1

---

## Introducción

---

Actualmente, el cambio climático y el manejo inadecuado de los residuos son problemas que afectan globalmente. En el caso de Guatemala, esta problemática se manifiesta en la destrucción de los recursos hídricos y naturales a lo largo del país, principalmente debido a la incorrecta gestión de los desechos y las altas tasas de contaminación en muchas regiones [47].

Conscientes de esta situación, el Ministerio de Ambiente y Recursos Naturales, en colaboración con diversas entidades gubernamentales, ha propuesto una ley cuyo objetivo principal es fomentar una adecuada distribución de los residuos en cada hogar guatemalteco [40]. Sin embargo, esto plantea un nuevo desafío: lograr que las personas comprendan cómo clasificar correctamente los diferentes tipos de desechos.

Por esta razón, surge la idea del presente un megaproyecto que consista en desarrollar una Aplicación Móvil para la Gestión de Residuos y la Reducción de la Huella de Carbono. Esta aplicación, diseñada para ser de uso sencillo, tiene como objetivo facilitar a los habitantes de la Ciudad de Guatemala la correcta clasificación y gestión diaria de los desechos generados en sus hogares, así como llevar un registro de dichos datos. Este proyecto es parte de dicha iniciativa.

En lo que respecta al módulo de *Backend*, Gestión de Base de Datos y Análisis de Datos, se busca un diseño adecuado para el almacenamiento de la información recabada durante el uso de la aplicación. Estos datos son fundamentales, ya que proporcionarán métricas útiles a los usuarios.

Además, es necesario establecer un esquema de datos y relaciones que permita que la información almacenada pueda ser consultada de forma rápida y precisa. En cuanto a las transacciones, este módulo se encargará de gestionar los datos de los usuarios, asegurando que la información clave sea almacenada adecuadamente. A través de las diferentes transacciones que se construyan, se podrá ofrecer información precisa y útil a cada usuario.

# CAPÍTULO 2

---

## Objetivos

---

### 2.1. Objetivo General

Desarrollar un módulo de *Backend*, Gestión de Base de Datos y Análisis de Datos que permita la gestión eficiente de la información relacionada con la clasificación de residuos y gestión de desechos en la aplicación, asegurando su escalabilidad y capacidad para generar métricas valiosas.

### 2.2. Objetivos Específicos

- Investigar y seleccionar las tecnologías más adecuadas para el desarrollo de la aplicación, considerando escalabilidad y compatibilidad con dispositivos móviles.
- Crear una estructura sólida para almacenar datos sobre los tipos de residuos y la frecuencia de desecho de los usuarios.
- Desarrollar un sistema de base de datos que permita almacenar y gestionar la información sobre los desechos clasificados por los usuarios y los datos relacionados con la gestión de residuos.
- Desarrollar una *API* que permita la comunicación entre el frontend y la base de datos, garantizando la integridad de los datos transmitidos.
- Procesar y analizar los datos generados por los usuarios para obtener métricas y tendencias útiles que los usuarios puedan utilizar para mejorar sus prácticas de gestión de residuos.

# CAPÍTULO 3

---

## Justificación

---

La implementación del módulo de *Backend*, Gestión de Base de Datos y Análisis de Datos es esencial para garantizar el éxito del proyecto de la aplicación para la Gestión de Residuos y la Reducción de la Huella de Carbono. Este módulo cumple varias funciones críticas que justifican su desarrollo y existencia.

Primero, la aplicación generará una gran cantidad de datos, desde la clasificación de residuos hasta el seguimiento de la huella de carbono de los usuarios. Un módulo de *backend* bien diseñado permitirá organizar y almacenar estos datos de manera eficiente, asegurando que sean accesibles y utilizables en tiempo real [52]. Sin una estructura de datos adecuada, la aplicación enfrentaría problemas de rendimiento y escalabilidad, comprometiendo su efectividad.

En segundo lugar, en un sistema que maneja datos sensibles, como los comportamientos de los usuarios y su impacto ambiental, es crucial mantener la integridad y seguridad de la información. El módulo de gestión de base de datos garantizará que todas las operaciones realizadas sean consistentes y seguras, aplicando principios ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) para proteger la información contra errores y accesos no autorizados [7].

Además, la recopilación y análisis de datos son fundamentales para medir el éxito del proyecto y proporcionar retroalimentación útil a los usuarios. El módulo de análisis de datos permitirá procesar grandes volúmenes de información, generando métricas y reportes que ayudarán a los usuarios a comprender mejor sus hábitos de gestión de residuos y su huella de carbono [20]. Estos análisis proporcionarán resultados valiosos para futuras mejoras de la aplicación y la formulación de estrategias de educación ambiental.

A medida que más usuarios adopten la aplicación, la cantidad de datos y transacciones aumentará significativamente. Una gestión adecuada de la base de datos permitirá escalar el sistema eficientemente, asegurando que pueda manejar un mayor volumen de datos sin pérdida de rendimiento [8]. Esto es vital para la sostenibilidad a largo plazo del proyecto.

Finalmente, el diseño flexible del módulo permitirá la integración de nuevas funcionalidades y la adaptación a las necesidades emergentes de los usuarios y reguladores. La capacidad de realizar actualizaciones y mejoras de manera eficiente es crucial para mantener la relevancia y efectividad de la aplicación a lo largo del tiempo [15].

En conclusión, el módulo de *Backend*, Gestión de Base de Datos y Análisis de Datos no solo es un componente técnico fundamental, sino también una pieza estratégica que garantizará la eficiencia

y sostenibilidad del proyecto. Su implementación contribuirá significativamente al éxito de la aplicación, permitiendo a los usuarios gestionar adecuadamente sus residuos, reducir su huella de carbono y participar activamente en la protección del medio ambiente.

# CAPÍTULO 4

---

Marco Teórico

---

## 4.1. Backend

### 4.1.1. ¿Qué es un Backend?

El backend se refiere a la parte de una aplicación o sistema que se encarga de la lógica de negocio, el procesamiento de datos y la interacción con la base de datos, es decir, el lado del servidor. A diferencia del frontend, que es la interfaz visible para el usuario, el backend maneja las solicitudes y respuestas que posibilitan las funcionalidades internas de la aplicación [34]. En términos generales, el backend actúa como el puente que conecta el usuario final con la base de datos y otros servicios de la aplicación [44].

### 4.1.2. Componentes de un Backend

El backend de una aplicación se compone de varios elementos fundamentales: el servidor, la base de datos y el propio código de la aplicación. El servidor es la infraestructura donde se ejecutan las solicitudes de los clientes; las bases de datos almacenan y gestionan los datos de la aplicación; y el código de backend define la lógica de negocio y los procesos de la aplicación [31]. Además, los backends modernos suelen incluir APIs que facilitan la comunicación entre el frontend y otros servicios externos [57].

### 4.1.3. Backends más Comunes

Existen diferentes tecnologías y frameworks utilizados comúnmente para desarrollar backends, entre ellos *Node.js*, que destaca por su naturaleza asíncrona y capacidad para manejar múltiples solicitudes simultáneamente [54], *Django*, un framework basado en Python conocido por su robustez y seguridad [25], y *Ruby on Rails*, que simplifica el desarrollo al proporcionar una estructura organizada y preconfigurada para aplicaciones web [24]. Estos frameworks han ganado popularidad debido a su facilidad de uso, eficiencia y adaptabilidad a diferentes tipos de aplicaciones.

## 4.2. Tecnologías para el Desarrollo de Aplicaciones Escalables

### 4.2.1. Introducción a la Escalabilidad en Aplicaciones Móviles

La escalabilidad en el desarrollo de aplicaciones móviles es la capacidad de un sistema para manejar un número creciente de usuarios, datos y transacciones sin comprometer el rendimiento. Esto implica que el sistema puede crecer y adaptarse a demandas cada vez mayores sin necesidad de realizar cambios estructurales significativos. En el contexto de aplicaciones para la gestión de residuos, la escalabilidad es esencial porque se espera que el volumen de datos crezca significativamente a medida que más usuarios adopten la plataforma. Según [54], la escalabilidad no solo se refiere al hardware, sino también a cómo se diseñan las aplicaciones para garantizar que puedan distribuir la carga de manera eficiente a lo largo del tiempo. Esto incluye aspectos como el diseño de la base de datos y la capacidad de la API para manejar múltiples solicitudes simultáneas.

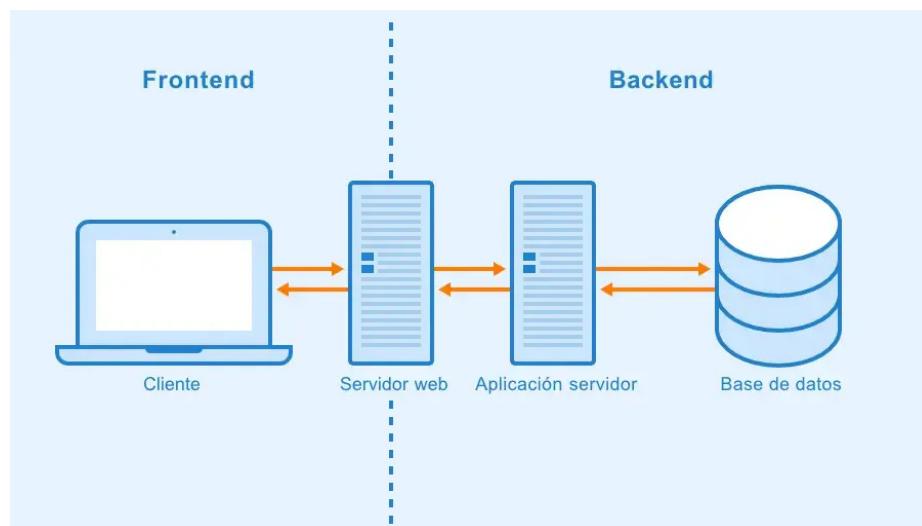


Figura 4.1: Arquitectura cliente-servidor: comunicación entre frontend y backend [18].

### 4.2.2. Tecnologías Actuales para el Desarrollo de Aplicaciones Escalables

Existen diversas tecnologías que permiten que las aplicaciones móviles sean escalables. Estas tecnologías incluyen tipos de bases de datos y el uso de frameworks que permiten la construcción de APIs RESTful para la comunicación entre sistemas. La elección de la tecnología adecuada depende del tamaño de la aplicación, el tipo de datos que se manejará y los requisitos de rendimiento.

#### Plataformas para creación de APIs (Node.js, Django, Flask, etc.)

Las plataformas para la creación de APIs son fundamentales para el desarrollo de aplicaciones escalables. *Node.js* es una opción ampliamente utilizada debido a su naturaleza asíncrona y su capacidad para manejar múltiples solicitudes de manera eficiente. Esto la convierte en una herramienta ideal para construir APIs en tiempo real, como las de gestión de residuos [54]. Por otro lado, *Django*, un framework basado en Python, es conocido por su robustez y capacidad para gestionar aplicaciones complejas de manera eficiente. Django incluye un ORM (Object-Relational Mapping) que facilita la interacción con bases de datos relacionales, lo que lo hace adecuado para APIs que requieren seguridad y escalabilidad [25]. Finalmente, *Flask*, también basado en Python, es una alternativa más ligera

que se puede utilizar cuando se necesita una mayor flexibilidad y personalización en el desarrollo de la API.

### Bases de Datos Relacionales vs No Relacionales (SQL vs NoSQL)

La elección de la base de datos es crítica para la escalabilidad de una aplicación. Las bases de datos *SQL*, como *PostgreSQL* y *MySQL*, son bases de datos relacionales que permiten una estructura clara y bien definida de los datos. Estas bases de datos son adecuadas para aplicaciones que requieren transacciones complejas y consistencia de los datos [51]. En cambio, las bases de datos *NoSQL*, como *MongoDB* o *Cassandra*, son más adecuadas cuando se manejan grandes volúmenes de datos no estructurados o cuando la disponibilidad es prioritaria sobre la consistencia [23]. Las bases de datos NoSQL permiten la escalabilidad horizontal, lo que significa que es posible distribuir los datos a través de múltiples servidores, lo que es ideal para aplicaciones que crecen rápidamente.

Características	Bases de datos relacionales	Bases de datos no relacionales
<b>Modelo de datos</b>	Basado en tablas con filas y columnas, usa relaciones (joins).	Basado en documentos, grafos, pares clave-valor, o modelos de columnas.
<b>Estructura</b>	Esquema fijo y predefinido (estructurado).	Esquema flexible (puede ser no estructurado).
<b>Integridad de los datos</b>	Alta integridad mediante claves primarias y foráneas.	Baja o variable integridad; depende de la aplicación.
<b>Escalabilidad</b>	Vertical (aumentar recursos del servidor).	Horizontal (añadir más servidores/nodos).
<b>Consultas</b>	Lenguaje SQL, ideal para datos estructurados y relaciones complejas.	Varía (NoSQL), más eficiente para grandes volúmenes y consultas rápidas.
<b>Transacciones</b>	Soporte completo para ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad).	Puede no soportar ACID completamente, prioriza la disponibilidad y escalabilidad (BASE).
<b>Ejemplos</b>	MySQL, PostgreSQL, Oracle.	MongoDB, Cassandra, Redis.

Tabla 4.1: Comparación entre bases de datos relacionales y no relacionales

### APIs Restful y Protocolos de Comunicación

Las *APIs RESTful* son la arquitectura preferida para el desarrollo de aplicaciones escalables debido a su simplicidad y eficiencia. REST (*Representational State Transfer*) utiliza los métodos HTTP estándar como GET, POST, PUT y DELETE para interactuar con los recursos del sistema [14]. Una API bien diseñada garantiza que la comunicación entre el frontend y la base de datos sea rápida y segura. Además, REST permite que las aplicaciones escalen al distribuir las solicitudes de manera eficiente entre múltiples servidores, lo que es crucial en una aplicación de gestión de residuos donde los usuarios envían datos de manera continua.

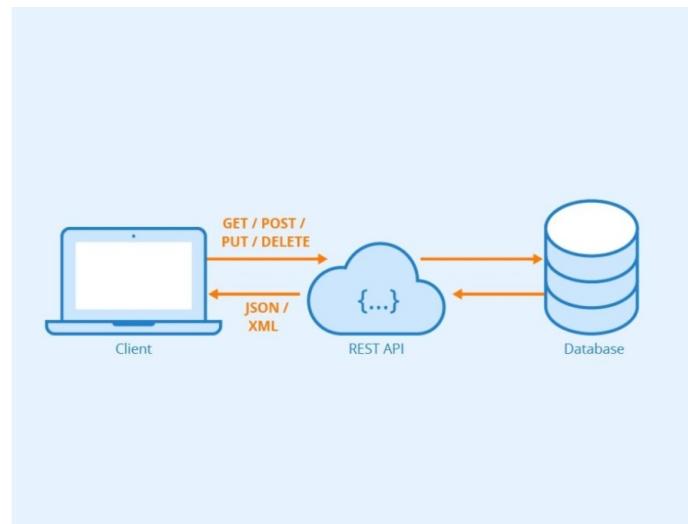


Figura 4.2: Diagrama de comunicación REST API. Fuente: [41].

#### 4.2.3. Factores para la Selección de Tecnologías Adecuadas en Proyectos Móviles

Seleccionar las tecnologías adecuadas para el desarrollo de aplicaciones móviles implica considerar varios factores, como la escalabilidad, el rendimiento, la facilidad de uso y los costos de mantenimiento. Según [23], las plataformas y las bases de datos deben seleccionarse con base en su capacidad para manejar el volumen esperado de datos y usuarios, así como en su capacidad para adaptarse a futuras necesidades de la aplicación. También es importante considerar la comunidad de soporte y la documentación disponible para cada tecnología, lo que facilita su implementación y resolución de problemas a lo largo del desarrollo del proyecto.

### 4.3. Bases de Datos en la Gestión de Residuos y Clasificación de Desechos

#### 4.3.1. Diseño de Bases de Datos para Aplicaciones de Gestión de Residuos

El diseño de la base de datos es fundamental para cualquier sistema de gestión de residuos, ya que los datos deben almacenarse de manera eficiente para garantizar que la aplicación funcione correctamente. Una base de datos bien estructurada permite que la información de los usuarios, los residuos y las transacciones se organice de manera que sea fácilmente accesible y procesable. El diseño de una base de datos para este tipo de aplicaciones implica la creación de un modelo de datos que refleje la realidad del problema que se está abordando, como la clasificación de residuos y la frecuencia con que los usuarios generan estos desechos.

#### Modelos Entidad-Relación (E-R) Aplicados a la Gestión de Residuos

El *modelo Entidad-Relación (E-R)* es una herramienta útil para el diseño de bases de datos, ya que permite identificar las entidades principales del sistema, como los usuarios, los tipos de residuos y

las transacciones, y las relaciones entre estas entidades [10]. En una aplicación de gestión de residuos, las relaciones entre los usuarios y los tipos de residuos son clave para garantizar que los datos se organicen de manera lógica y eficiente. Además, el modelo E-R facilita la identificación de atributos clave para cada entidad, como la frecuencia de desecho de cada tipo de residuo por parte de los usuarios, lo que es esencial para generar métricas útiles.

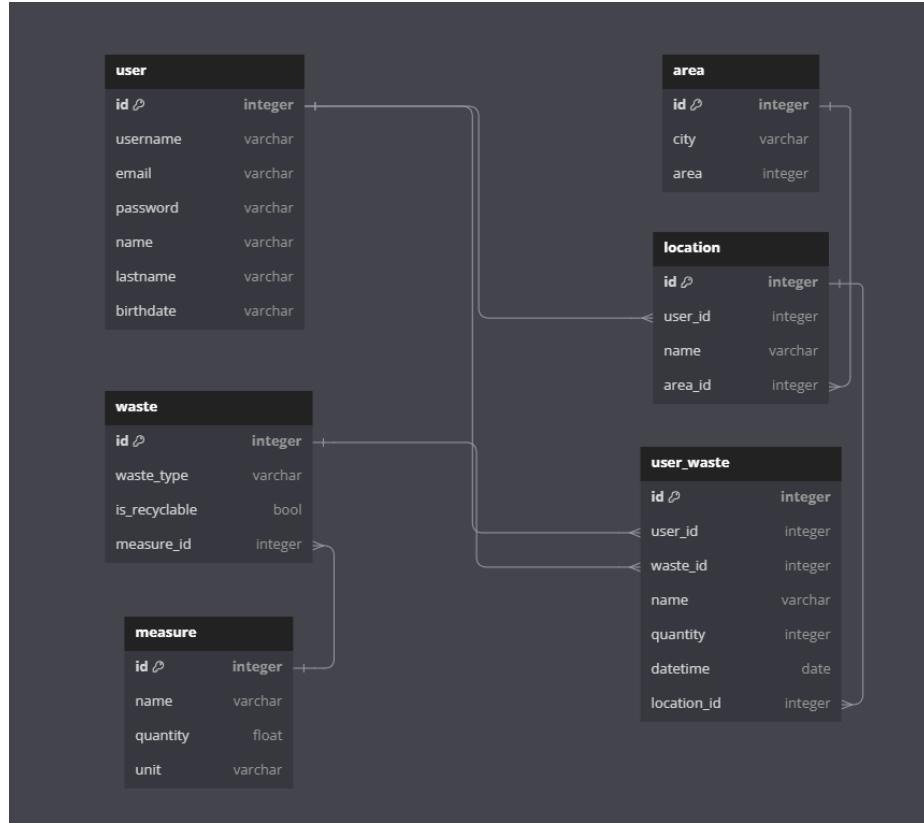


Figura 4.3: Diagrama de la base de datos. Elaboración propia.

### Estructuración de Datos: Tipos de Residuos y Frecuencia de Desecho

La estructuración de datos es un aspecto crítico en cualquier base de datos. En el caso de la gestión de residuos, es importante almacenar información detallada sobre los tipos de residuos que se generan y la frecuencia con que los usuarios realizan el desecho de estos. Esta información permite no solo el seguimiento de las prácticas de los usuarios, sino también la generación de informes que pueden ayudar a mejorar la eficiencia del sistema. De acuerdo con [5], una base de datos bien estructurada no solo facilita el almacenamiento de los datos, sino que también mejora la capacidad del sistema para generar consultas eficientes y precisas.

#### 4.3.2. Normalización de Bases de Datos: Asegurando la Eficiencia y Consistencia

La *normalización* de bases de datos es un proceso que tiene como objetivo eliminar redundancias y asegurar que los datos estén organizados de manera eficiente. Esto se logra dividiendo las tablas de la base de datos en entidades más pequeñas y asegurando que cada entidad tenga un único propósito. Según [5], la normalización se lleva a cabo en diferentes etapas, conocidas como formas normales.

### Primeras, Segundas y Terceras Formas Normales

La primera forma normal (1NF) asegura que los datos en cada tabla estén en formato tabular, es decir, que no existan grupos repetidos de datos. La segunda forma normal (2NF) elimina la dependencia parcial de las claves primarias, mientras que la tercera forma normal (3NF) elimina dependencias transitivas [10]. En un sistema de gestión de residuos, la normalización asegura que cada tipo de residuo y cada usuario se gestionen de manera independiente, lo que facilita el análisis posterior de los datos.

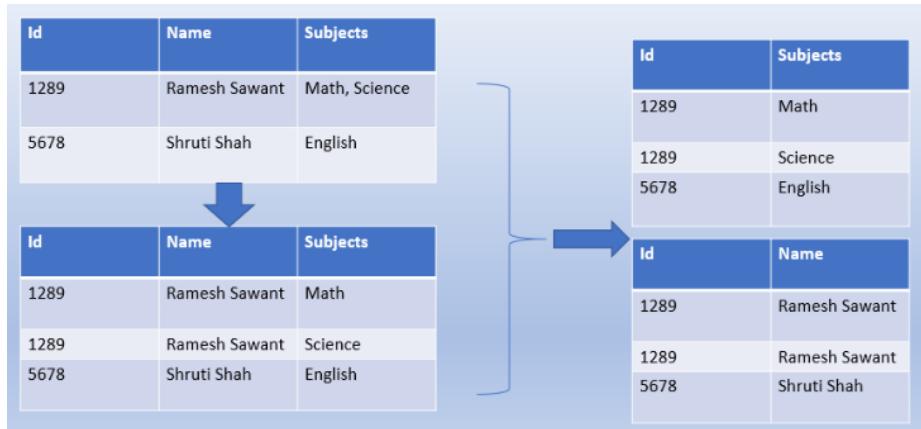


Figura 4.4: Representación de la Segunda Forma Normal (Second Normal Form). Fuente: [46].

### Ventajas y Desventajas de la Normalización en Sistemas Escalables

La normalización tiene varias ventajas, incluida la eliminación de redundancias, lo que reduce el tamaño de la base de datos y mejora su eficiencia. Sin embargo, también tiene algunas desventajas, como el aumento en la complejidad de las consultas, especialmente en sistemas grandes y escalables. Según [10], en algunos casos puede ser beneficioso desnormalizar partes de la base de datos para mejorar el rendimiento, aunque esto debe hacerse con precaución para evitar inconsistencias.

#### 4.3.3. Gestión de Transacciones: Principios ACID y su Aplicación

El manejo de transacciones en bases de datos es esencial para garantizar que las operaciones se realicen de manera segura y consistente. Los *principios ACID* (Atomicidad, Consistencia, Aislamiento y Durabilidad) aseguran que las transacciones en la base de datos sean correctas y que los datos permanezcan consistentes, incluso en situaciones de fallo [17]. En el caso de una aplicación de gestión de residuos, estos principios son fundamentales para garantizar que la información sobre los residuos clasificados por los usuarios se almacene correctamente y que los datos puedan recuperarse sin errores.

#### Atomicidad, Consistencia, Aislamiento y Durabilidad

La atomicidad asegura que todas las operaciones dentro de una transacción se completen correctamente, o no se realicen en absoluto. La consistencia garantiza que la base de datos pase de un estado válido a otro, mientras que el aislamiento asegura que las transacciones concurrentes no interfieran entre sí. Finalmente, la durabilidad garantiza que, una vez que una transacción se completa, los datos no se perderán incluso si el sistema falla [17]. Estos principios son críticos para asegurar

que los datos de los usuarios sobre la clasificación de residuos y otros comportamientos se almacenen y procesen correctamente.

## 4.4. Desarrollo de una API

### 4.4.1. Introducción a la Arquitectura Cliente-Servidor

El diseño de una *API* (*Application Programming Interface*) es crucial para la comunicación entre el frontend (la interfaz de usuario) y la base de datos de una aplicación móvil. En el caso de una aplicación de gestión de residuos, la API actúa como un intermediario que permite que los datos del usuario se envíen al servidor y se almacenen en la base de datos, y que las respuestas se envíen de vuelta al usuario en tiempo real. Según [48], una API bien diseñada permite que la aplicación se comunique de manera eficiente y segura, facilitando la escalabilidad del sistema.

### 4.4.2. Principios de Diseño de una API Restful para Aplicaciones Móviles

El diseño de una *API RESTful* implica el uso de los principios de REST (*Representational State Transfer*), que utiliza los métodos HTTP (GET, POST, PUT, DELETE) para interactuar con los recursos del sistema [14]. Las APIs RESTful son ligeras y fáciles de implementar, lo que las convierte en una opción popular para aplicaciones móviles que requieren comunicación en tiempo real. Además, REST es escalable y flexible, permitiendo que los desarrolladores añadan nuevas funcionalidades sin tener que reescribir grandes partes del código existente.

#### Endpoints, Métodos HTTP (GET, POST, PUT, DELETE)

En una API RESTful, los *endpoints* son los puntos de acceso a los recursos del sistema. Cada endpoint representa una entidad específica, como los datos del usuario o la información sobre los tipos de residuos. Los métodos HTTP (GET, POST, PUT, DELETE) permiten realizar operaciones sobre estos recursos. Por ejemplo, GET se utiliza para recuperar información, mientras que POST se utiliza para crear nuevos registros. [48] destacan que una API bien estructurada debe seguir los principios REST para asegurar que la comunicación entre el frontend y la base de datos sea eficiente y fácil de escalar.

#### Seguridad en la Transmisión de Datos (autenticación, encriptación)

La seguridad es un aspecto fundamental en el diseño de una API, especialmente cuando se manejan datos sensibles, como el comportamiento de los usuarios y su huella de carbono. Para garantizar la seguridad en la transmisión de datos, se utilizan técnicas como la *autenticación* de usuarios, que asegura que solo usuarios autorizados puedan acceder a los recursos. Además, la encriptación de los datos mediante HTTPS es esencial para evitar que la información sea interceptada durante la transmisión [22]. Esto garantiza que los datos de los usuarios estén protegidos contra accesos no autorizados y ataques de intermediarios.

#### 4.4.3. Integridad de los Datos Transmitidos: Mecanismos de Validación y Seguridad

La integridad de los datos es crucial para asegurar que la información que se transmite entre el frontend y la base de datos sea precisa y esté completa. Los mecanismos de validación de datos aseguran que los datos enviados por el usuario sean correctos antes de que se almacenen en la base de datos. Esto incluye la validación de los tipos de datos, la longitud de los campos y la estructura del contenido [27]. Además, es importante implementar técnicas de seguridad como el uso de tokens de autenticación (por ejemplo, JWT) para proteger los datos contra accesos no autorizados [22].

### 4.5. Análisis de Datos Generados por los Usuarios

#### 4.5.1. Introducción al Análisis de Datos en la Gestión de Residuos

El análisis de datos es fundamental para extraer valor de la información generada por los usuarios en una aplicación de gestión de residuos. Los datos, como la clasificación de residuos y la frecuencia de desecho, pueden analizarse para identificar patrones y proporcionar recomendaciones a los usuarios. Según [21], el análisis de datos permite descubrir tendencias en el comportamiento de los usuarios que pueden ayudar a mejorar la eficiencia del sistema y fomentar prácticas más sostenibles entre los usuarios.

#### 4.5.2. Técnicas para Procesar y Analizar Datos

Existen varias técnicas de análisis de datos que son útiles en el contexto de la gestión de residuos. Entre las más comunes se encuentran la minería de datos, que permite extraer patrones a partir de grandes volúmenes de datos, y el análisis predictivo, que puede anticipar el comportamiento futuro de los usuarios basado en datos históricos. De acuerdo con [21], estas técnicas son útiles para identificar qué tipos de residuos se generan con mayor frecuencia y cómo pueden mejorarse los hábitos de desecho de los usuarios.

##### Minería de Datos y Análisis de Tendencias

La minería de datos es un proceso que utiliza algoritmos avanzados para analizar grandes volúmenes de datos y descubrir patrones ocultos. En una aplicación de gestión de residuos, la minería de datos puede ayudar a identificar tendencias en el comportamiento de los usuarios, como la frecuencia con la que clasifican ciertos tipos de residuos o los días de la semana en que generan más desechos. Estas tendencias pueden ser utilizadas para proporcionar recomendaciones personalizadas a los usuarios y para mejorar el diseño de la aplicación [21].

##### Herramientas de Análisis (Python, R, SQL)

Las herramientas de análisis de datos, como Python, R y SQL, son ampliamente utilizadas para procesar y analizar grandes volúmenes de datos. Python es particularmente útil debido a sus bibliotecas de análisis de datos, como Pandas y NumPy, que permiten manipular y analizar datos de manera eficiente [37]. R, por su parte, es popular en el análisis estadístico y visualización de datos, mientras que SQL se utiliza para consultar y extraer datos directamente desde bases de datos relacionales. Estas herramientas facilitan el procesamiento y análisis de los datos generados por los usuarios en tiempo real.

### 4.5.3. Métricas para la Reducción de la Huella de Carbono

Una de las funciones clave de la aplicación es la capacidad de generar métricas relacionadas con la huella de carbono de los usuarios. Estas métricas pueden incluir información sobre la cantidad de residuos reciclados por cada usuario, la frecuencia con la que desechan ciertos tipos de residuos y el impacto ambiental de sus acciones. Según [37], las herramientas de análisis de datos permiten generar estas métricas de manera automática, proporcionando retroalimentación útil para los usuarios y ayudándolos a reducir su huella de carbono.

## 4.6. Escalabilidad y Rendimiento en el Sistema de Gestión de Residuos

En el contexto de la gestión de residuos, la escalabilidad y el rendimiento de la base de datos y el API son fundamentales, ya que un sistema mal diseñado puede fallar a medida que la cantidad de usuarios y transacciones crece. La escalabilidad se refiere a la capacidad del sistema para aumentar su rendimiento y recursos conforme crece la demanda, mientras que el rendimiento se relaciona con la rapidez y eficiencia del sistema en la ejecución de sus operaciones.

### 4.6.1. Optimización de Consultas y Operaciones CRUD

Las operaciones CRUD (*Create, Read, Update, Delete*) son las operaciones básicas que se realizan sobre una base de datos. Optimizar estas operaciones es esencial para mejorar el rendimiento general del sistema y asegurar que las consultas se ejecuten de manera eficiente, especialmente cuando se trabaja con grandes volúmenes de datos.

#### Optimización de Queries y Consultas SQL

La optimización de queries SQL es otra técnica crucial para mejorar el rendimiento de las consultas en una base de datos. Esto implica escribir consultas eficientes que utilicen los índices correctamente y eviten operaciones costosas, como los *joins* innecesarios o las subconsultas complejas. Según [10], una buena práctica es analizar las consultas utilizando herramientas de análisis de rendimiento (como el EXPLAIN en PostgreSQL) para identificar cuellos de botella y optimizar las partes que consumen más tiempo. Las consultas optimizadas reducen la carga sobre la base de datos y permiten que el sistema maneje un mayor número de transacciones sin sacrificar el rendimiento.

### 4.6.2. Pruebas de Carga y Rendimiento en el Backend

Las pruebas de carga y rendimiento son esenciales para garantizar que el backend de la aplicación pueda manejar la demanda esperada de usuarios y transacciones sin degradar el rendimiento.

Las pruebas de rendimiento se utilizan para evaluar cómo el sistema responde bajo diferentes condiciones de carga. Estas pruebas simulan un número creciente de usuarios o transacciones para identificar cuándo el sistema comienza a tener problemas de rendimiento. Según [19], las pruebas de rendimiento deben realizarse en condiciones lo más cercanas posibles a las del entorno de producción para garantizar que los resultados sean representativos. Además, es importante realizar pruebas de estrés, que someten al sistema a una carga superior a la esperada para ver cómo responde ante situaciones extremas.

## 4.7. CI/CD: Integración y Entrega Continua

El desarrollo de software moderno hace uso de CI/CD (Integración Continua/Entrega Continua o Despliegue Continuo) para mejorar la eficiencia, calidad y consistencia de las aplicaciones. CI/CD permite la integración frecuente de código y despliegues automáticos a entornos de producción, reduciendo errores y asegurando que las aplicaciones estén listas para ser lanzadas con mayor rapidez [16, 26, 1]. Este proceso se compone de varios elementos fundamentales: pruebas, monitoreo y automatización.

### 4.7.1. Pruebas

Las pruebas en CI/CD son esenciales para identificar y solucionar errores en etapas tempranas del desarrollo, lo cual es especialmente importante en el contexto de APIs RESTful, donde se espera que cada cambio en el código mantenga la consistencia de los puntos de conexión y sus respuestas. A través de pruebas unitarias, de integración y de rendimiento, los equipos pueden asegurar que la API funcione de acuerdo con sus especificaciones sin afectar negativamente la experiencia del usuario final [12, 39]. Las herramientas para pruebas automáticas incluyen *Mocha* para pruebas unitarias, *Postman* para pruebas de endpoints de la API y *Artillery* para pruebas de carga y rendimiento en entornos de producción [30, 56]. La integración de estas pruebas en pipelines de CI/CD permite validaciones consistentes y rigurosas, contribuyendo a la robustez de la API [9].

### 4.7.2. Monitoreo

El monitoreo continuo es crucial para asegurar que la API RESTful mantenga un rendimiento óptimo y esté libre de errores una vez desplegada en producción. Las herramientas de monitoreo permiten rastrear métricas como tiempos de respuesta, uso de recursos y posibles errores en tiempo real, elementos esenciales para aplicaciones con un alto volumen de solicitudes. Herramientas como *Prometheus* y *Grafana* son comúnmente utilizadas para monitoreo de métricas y visualización, mientras que *Sentry* se utiliza para monitoreo de errores, ofreciendo un seguimiento detallado de excepciones en el código [29, 33, 35]. La combinación de estas herramientas permite una gestión proactiva de la API, ayudando a identificar y corregir problemas antes de que afecten a los usuarios finales [45].

### 4.7.3. Automatización

La automatización en CI/CD facilita la integración y despliegue continuo mediante la eliminación de tareas manuales repetitivas, permitiendo a los desarrolladores concentrarse en mejorar el código en lugar de en procesos de integración y despliegue. Los scripts de automatización configuran la ejecución de pruebas, la compilación y el despliegue de la API de manera eficiente, garantizando una consistencia y velocidad adecuadas al proceso de desarrollo [38, 2]. Herramientas como *Jenkins*, *CircleCI*, *GitLab CI/CD* y *GitHub Actions* permiten implementar pipelines de CI/CD completamente automatizados, los cuales aseguran una integración continua desde la codificación hasta la entrega en producción [49, 32]. *GitHub Actions*, en particular, facilita la configuración de workflows de CI/CD directamente desde el repositorio de código, integrándose de manera nativa con GitHub y permitiendo a los desarrolladores automatizar tareas como pruebas, construcción y despliegue cada vez que se realiza un cambio en el código [28]. Estas herramientas permiten a los desarrolladores mantener un flujo de trabajo constante y eficiente, minimizando errores y reduciendo el tiempo de entrega [50].

## 4.8. Tecnologías Utilizadas en el Proyecto

El desarrollo de una aplicación para la gestión de residuos y reducción de la huella de carbono requiere el uso de diversas tecnologías que permitan no solo el manejo eficiente de los datos, sino también asegurar la escalabilidad, seguridad y facilidad de uso de la aplicación. A continuación, se presentan las principales tecnologías utilizadas en el desarrollo del proyecto, junto con las justificaciones para su selección.

### 4.8.1. Elección de Tecnologías para el Backend

Para la elección de las tecnologías utilizadas en el desarrollo del backend y la gestión de datos en el proyecto, se priorizó la escalabilidad, el rendimiento y la facilidad de uso. Las tecnologías seleccionadas fueron **Express.js** para el API y **PostgreSQL** como motor de base de datos. Estas tecnologías proporcionan un marco robusto que permite manejar grandes volúmenes de datos y usuarios, asegurando que la aplicación se mantenga eficiente conforme crezca el número de usuarios.

#### Express.js para el API

**Express.js** es un framework ligero y flexible basado en **Node.js** que permite la creación rápida y eficiente de APIs RESTful. Este framework fue elegido debido a su naturaleza asíncrona y su capacidad para manejar múltiples solicitudes de manera concurrente, lo que lo convierte en una opción ideal para sistemas que requieren escalabilidad y alto rendimiento [54]. Además, Express.js ofrece una gran modularidad, lo que facilita la adición de middleware y la configuración de rutas para las operaciones CRUD en la base de datos [43].

Express.js destaca en aplicaciones donde la baja latencia es crucial, como en la comunicación entre el frontend y el API en una aplicación de gestión de residuos. Gracias a su arquitectura asíncrona, se pueden procesar múltiples solicitudes sin bloquear el servidor, lo que garantiza tiempos de respuesta rápidos y escalabilidad sin necesidad de aumentar excesivamente los recursos del servidor.

#### PostgreSQL como Motor de Base de Datos

La base de datos seleccionada fue **PostgreSQL**, un sistema de gestión de bases de datos relacional de código abierto que soporta grandes volúmenes de datos y consultas complejas. PostgreSQL es compatible con las propiedades **ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad)**, lo que garantiza la seguridad y consistencia de los datos incluso en situaciones de fallos o concurrencia [51]. Además, PostgreSQL permite una integración eficiente con aplicaciones desarrolladas en Express.js, lo que facilita la gestión de operaciones CRUD y análisis avanzados directamente desde la base de datos [53].

PostgreSQL también soporta la escalabilidad horizontal y vertical, lo que lo convierte en una opción ideal para aplicaciones que manejan un creciente número de usuarios y transacciones [5]. Las capacidades avanzadas de PostgreSQL para indexación, consultas de ventana y funciones analíticas permiten que el sistema realice análisis de datos en tiempo real sin la necesidad de herramientas externas.

En conjunto, estas dos tecnologías proporcionan una base sólida para el desarrollo del sistema, permitiendo una escalabilidad eficiente y un rendimiento alto a medida que la plataforma crece.

#### 4.8.2. Integración entre Express.js y PostgreSQL

La integración entre **Express.js** y **PostgreSQL** es crucial para el correcto funcionamiento de la aplicación. A través de la API RESTful construida con Express.js, las operaciones **CRUD (Crear, Leer, Actualizar, Borrar)** se realizan sobre la base de datos PostgreSQL, asegurando que los datos ingresados por los usuarios se almacenen y gestionen de manera eficiente [54].

El uso combinado de estas tecnologías permite que la aplicación pueda escalar sin problemas, ya que Express.js maneja las solicitudes de los usuarios mientras que PostgreSQL gestiona el almacenamiento y recuperación de datos. Además, la conexión entre ambas herramientas garantiza la integridad de las transacciones y la consistencia de los datos almacenados.

#### 4.8.3. Herramientas de Análisis de Datos

El análisis de los datos generados por los usuarios se realiza completamente dentro de **PostgreSQL**, aprovechando sus capacidades avanzadas para el procesamiento y análisis de datos. PostgreSQL no solo gestiona el almacenamiento de datos estructurados, sino que también facilita consultas complejas y operaciones analíticas que son fundamentales para extraer insights útiles y generar métricas sobre la clasificación de residuos y la frecuencia de desecho de los usuarios.

Gracias a funciones como agregaciones, consultas de ventana, y capacidades de join complejas, PostgreSQL permite realizar análisis detallados que ayudan a entender mejor los patrones de comportamiento de los usuarios y a mejorar la eficiencia del sistema de gestión de residuos. Además, las capacidades de indexación y optimización de consultas de PostgreSQL aseguran que el análisis de datos se realice de manera eficiente, incluso con grandes volúmenes de datos, garantizando tiempos de respuesta rápidos y reduciendo la carga en el sistema [53].

### 4.9. Impacto Social y Ambiental de la Gestión de Residuos

La gestión adecuada de residuos tiene un impacto directo en el medio ambiente y en la calidad de vida de las personas. A través de la implementación de tecnologías móviles, es posible educar a los usuarios y promover prácticas sostenibles que contribuyan a la reducción de residuos y la huella de carbono.

#### 4.9.1. La Tecnología como Herramienta de Educación Ambiental

La tecnología juega un papel clave en la educación ambiental, permitiendo a los usuarios tomar decisiones más informadas sobre cómo gestionar sus residuos. Aplicaciones como la desarrollada en este proyecto permiten a los usuarios ver el impacto directo de sus acciones en términos de reducción de residuos y huella de carbono. A través de notificaciones y reportes personalizados, los usuarios pueden recibir retroalimentación sobre sus hábitos y mejorar su compromiso con el reciclaje y la reducción de residuos [6].

Además, iniciativas similares a nivel internacional han demostrado que el uso de aplicaciones móviles puede aumentar la participación ciudadana en programas de reciclaje y gestión de residuos. Por ejemplo, aplicaciones como *JouleBug* y *MyWaste* han logrado incrementar la conciencia ambiental y la participación de los usuarios mediante incentivos y educación continua [4, 55]. Este proyecto espera replicar esos resultados en el contexto local de Guatemala, adaptando la tecnología a las necesidades específicas de la población.

#### 4.9.2. Reducción de la Huella de Carbono a Través de la Gestión de Residuos

Uno de los principales objetivos de la plataforma es ayudar a los usuarios a reducir su huella de carbono mediante la correcta clasificación de residuos. Según [36], la correcta gestión de residuos puede reducir significativamente las emisiones de gases de efecto invernadero, especialmente en zonas urbanas donde la acumulación de desechos es mayor. Al ofrecer reportes personalizados sobre la cantidad de residuos reciclados y no reciclados, la plataforma motiva a los usuarios a adoptar hábitos más sostenibles y a reducir su impacto ambiental.

#### 4.9.3. La Gestión de Residuos como Estrategia para la Reducción del Impacto Ambiental

La gestión de residuos se ha convertido en una de las principales estrategias para mitigar los efectos negativos del cambio climático y la contaminación ambiental. Según [36], la acumulación de residuos en vertederos y su inadecuada disposición contribuyen significativamente a la emisión de gases de efecto invernadero, como el metano, que es liberado durante la descomposición de residuos orgánicos en condiciones anaeróbicas. Implementar un sistema eficiente de gestión de residuos no solo reduce la cantidad de basura que llega a los vertederos, sino que también promueve la reutilización y el reciclaje de materiales, lo que disminuye la demanda de recursos naturales y reduce las emisiones de carbono.

En Guatemala, al igual que en muchos otros países, la gestión de residuos es un desafío importante debido a la falta de infraestructura adecuada y a la falta de conciencia ambiental entre la población [47]. A través del uso de tecnologías como las aplicaciones móviles, es posible educar a los usuarios sobre la importancia de clasificar adecuadamente sus residuos y reducir su huella de carbono. Esto es particularmente relevante en zonas urbanas donde la acumulación de residuos es mayor y tiene un impacto directo en la salud pública y el entorno.

#### 4.9.4. Clasificación de Residuos y Educación Ambiental a través de la Tecnología

La tecnología puede desempeñar un papel clave en la educación ambiental, permitiendo que los usuarios se informen sobre la correcta clasificación de residuos y los beneficios del reciclaje. Según [6], la educación ambiental es crucial para fomentar cambios en el comportamiento de las personas y promover prácticas sostenibles a largo plazo. Las aplicaciones móviles permiten ofrecer información personalizada a los usuarios sobre cómo clasificar los diferentes tipos de residuos, además de proporcionarles métricas que les muestran el impacto de sus acciones en el medio ambiente.

##### Ejemplos de Aplicaciones Móviles que Promueven la Gestión Ambiental

Existen varios ejemplos de aplicaciones móviles que han sido desarrolladas para promover la gestión ambiental y la reducción de residuos. Una de estas es *JouleBug*, una aplicación que incentiva a los usuarios a adoptar hábitos sostenibles mediante un sistema de recompensas y desafíos. La aplicación ofrece consejos sobre cómo reducir el consumo de energía, agua y residuos, y permite a los usuarios competir entre sí para ver quién adopta más prácticas sostenibles [4].

Otra aplicación es *MyWaste*, que ofrece información sobre cómo clasificar correctamente los residuos y proporciona alertas sobre los días de recolección de basura en diferentes áreas. Esta

aplicación ha sido utilizada por varias municipalidades para mejorar la eficiencia en la recolección de residuos y fomentar la participación ciudadana en el reciclaje [55].

Estas aplicaciones demuestran cómo la tecnología puede ser utilizada para fomentar la participación activa de los ciudadanos en la gestión de residuos y la protección del medio ambiente.

La correcta gestión de residuos tiene un impacto directo en la reducción de la huella de carbono y en la preservación del medio ambiente. Este proyecto no solo busca implementar una solución tecnológica, sino también promover un cambio de comportamiento en los usuarios mediante la concienciación sobre la clasificación adecuada de los residuos y la reducción del impacto ambiental.

# CAPÍTULO 5

---

## Metodología

---

El desarrollo de la aplicación se realizó en diversas etapas clave, cada una orientada a cumplir con los objetivos específicos del proyecto, como la creación del API, la gestión de la base de datos y el análisis de datos para una aplicación de gestión de residuos y reducción de huella de carbono. Durante este proceso, se tomaron decisiones estratégicas en cuanto a la selección de tecnologías, el diseño estructural de la base de datos, la implementación de las funcionalidades en el API, y el desarrollo de análisis de los datos de los usuarios.

Asimismo, a lo largo de la ejecución del proyecto, se tomaron decisiones basadas en buenas prácticas de desarrollo de software para optimizar la arquitectura del API principal del proyecto. Con base en un análisis riguroso, se implementaron procesos de CI/CD fundamentados en la metodología DevOps, con el objetivo de garantizar una entrega continua y de calidad.

Cada iteración representó un paso hacia un sistema más eficiente y adaptable, alineado con las necesidades de los usuarios, los objetivos de sostenibilidad del proyecto y la visión de establecer una base sólida para su futura expansión.

### 5.1. Elección de Tecnologías

La selección de tecnologías fue una de las decisiones más importantes en el desarrollo del proyecto, ya que influiría en la escalabilidad, flexibilidad y sostenibilidad de la aplicación a largo plazo. Se evaluaron diferentes opciones basadas en criterios de rendimiento, facilidad de uso y soporte continuo.

El motor de base de datos elegido fue PostgreSQL, una base de datos de código abierto reconocida por su robustez y cumplimiento de las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo cual la convierte en una excelente opción para gestionar los datos de una aplicación que requiere transacciones seguras y consistentes. PostgreSQL es altamente escalable, permitiendo manejar grandes volúmenes de datos y ejecutar consultas complejas de manera eficiente. Otra ventaja clave es su capacidad para realizar análisis avanzados directamente en la base de datos, eliminando la necesidad de exportar datos a herramientas externas y simplificando la arquitectura del sistema.

Para el desarrollo del API se eligió Express.js, un framework ligero de Node.js. Express.js permite la construcción rápida y flexible de APIs RESTful, fundamental para una comunicación fluida entre el frontend y la base de datos. Su naturaleza asíncrona facilita el manejo de múltiples solicitudes

concurrentes, haciéndolo ideal para una aplicación que necesita un rendimiento ágil y alta disponibilidad. Esta combinación de tecnologías garantiza un sistema eficiente, escalable y fácil de mantener, capaz de adaptarse a las crecientes demandas de la aplicación.

Una vez seleccionadas las tecnologías principales para el API y la estructura de la base de datos, se optó por herramientas complementarias que aseguren la calidad del sistema. Se eligió la herramienta de pruebas de carga Artillery, que permite una conexión eficaz con aplicaciones basadas en Express.js, facilitando la simulación de escenarios de tráfico y evaluando el comportamiento del API bajo diferentes cargas.

Además, Artillery se integra fácilmente con los procesos de CI/CD de GitHub Actions, lo cual permite automatizar estas pruebas y monitorizar el rendimiento del API a lo largo del tiempo. Esto asegura un control continuo del rendimiento y una preparación adecuada para las necesidades futuras de la aplicación.

Para proporcionar un monitoreo continuo de la API, se seleccionó una herramienta de monitoreo de errores: Sentry. Esta decisión se basó en la facilidad de implementación que Sentry ofrece para aplicaciones basadas en Express.js, además de ser una herramienta robusta que proporciona la información necesaria para gestionar y resolver errores de manera eficiente.

## 5.2. Primera Iteración

### 5.2.1. Base de Datos: Diseño versión inicial

En la **primera iteración**, se diseñó la estructura inicial de la base de datos, la cual se centró en tres tablas principales: **user**, **waste** y **user\_waste**. Estas tablas establecieron las bases del sistema, permitiendo gestionar la relación entre los usuarios y los residuos que generan.

- **user:** Esta tabla almacena la información básica de los usuarios, incluyendo campos como **id**, nombre de usuario (**username**), correo electrónico, contraseña, nombre, apellido y fecha de nacimiento. Esta estructura es clave para identificar de manera única a cada usuario en el sistema.
- **waste:** Define los diferentes tipos de residuos, con un **id** único, el tipo de residuo (**waste\_type**), si es reciclable o no (**is\_recyclable**), y el tipo de medida asociada (**measure\_type**), lo que proporciona flexibilidad para manejar residuos de distintas categorías.
- **user\_waste:** Relaciona a los usuarios con los residuos que generan. Contiene los campos **id** del usuario, **id** del residuo, nombre del residuo, la cantidad generada, la fecha en que se generó el residuo, y el lugar donde ocurrió la generación de residuos. Esta tabla almacena todos los registros que los usuarios ingresan.

A continuación se presenta el diagrama entidad-relación de la primera iteración de la base de datos:

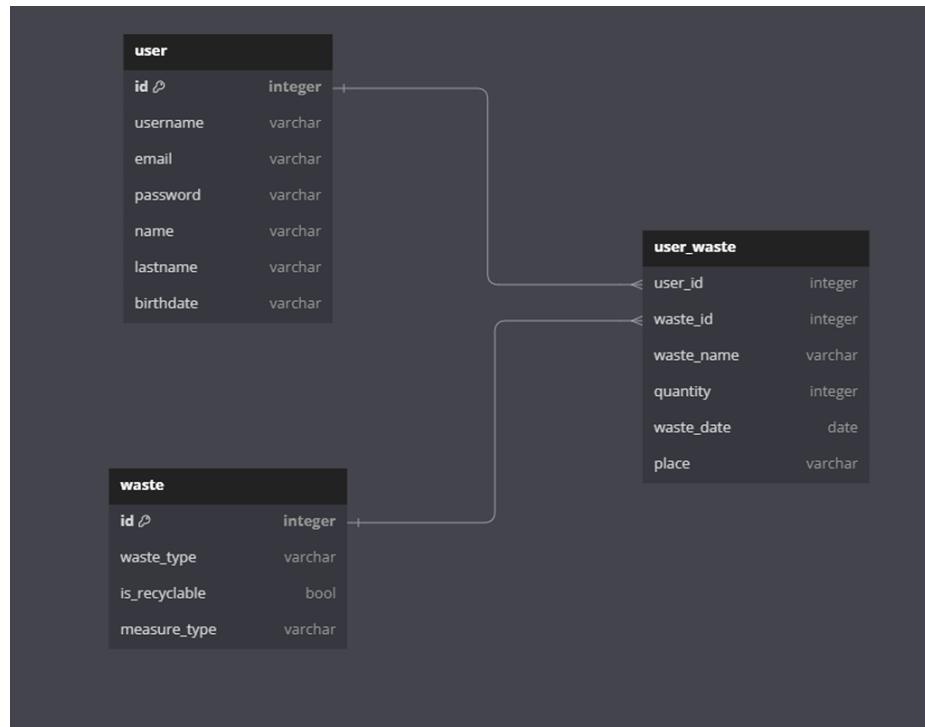


Figura 5.1: Diagrama entidad-relación de la primera iteración.

### 5.2.2. API: Definición de las funciones

Durante la **primera iteración**, se desarrollaron las funciones esenciales que permitieron a los usuarios registrarse en el sistema, acceder a sus cuentas y registrar los residuos generados. Estas funciones establecieron una base sencilla pero funcional para la interacción entre los usuarios y la base de datos, proporcionando un punto de partida confiable para futuras ampliaciones y mejoras del sistema.

En esta fase, se diseñó y configuró la estructura inicial de la base de datos. Las funciones de acceso a la base de datos se implementaron de forma manual, sin el uso de un ORM, permitiendo realizar las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) correspondientes para cada entidad en el sistema. Este enfoque inicial facilitó una implementación básica de las operaciones de gestión de datos y sentó las bases para manejar la estructura principal de las tablas y sus datos.

A continuación se presenta un query de ejemplo de creación de Usuarios:

```

1 INSERT INTO user (username, email, password, name, lastname, birthdate)
2 VALUES ('nombre_de_usuario', 'correo_electronico', 'contraseña', 'nombre',
3 'apellido', 'fecha_de_nacimiento');
  
```

Listing 5.1: Archivo createUser.sql

## 5.3. Segunda Iteración

### 5.3.1. Base de Datos: Diseño de tablas para ubicación y medición

En la **segunda iteración**, se realizó una revisión del diseño inicial para mejorar la precisión y el detalle de los datos almacenados. Se añadieron las tablas **location** y **measure**, lo que permitió incluir más información sobre la ubicación de los residuos generados y las medidas utilizadas para cuantificarlos.

- **location**: Almacena las ubicaciones relacionadas con los registros de residuos, permitiendo identificar el lugar donde se generan. Incluye los campos **id**, nombre del lugar, y su geoposición (coordenadas GPS).
- **measure**: Define las unidades de medida utilizadas para cuantificar los desechos, como kilogramos, litros, entre otros. Contiene los campos **id**, nombre de la medida, cantidad y unidad, lo que facilita la estandarización de las cantidades registradas.

El siguiente diagrama muestra la estructura mejorada de la base de datos en la segunda iteración:

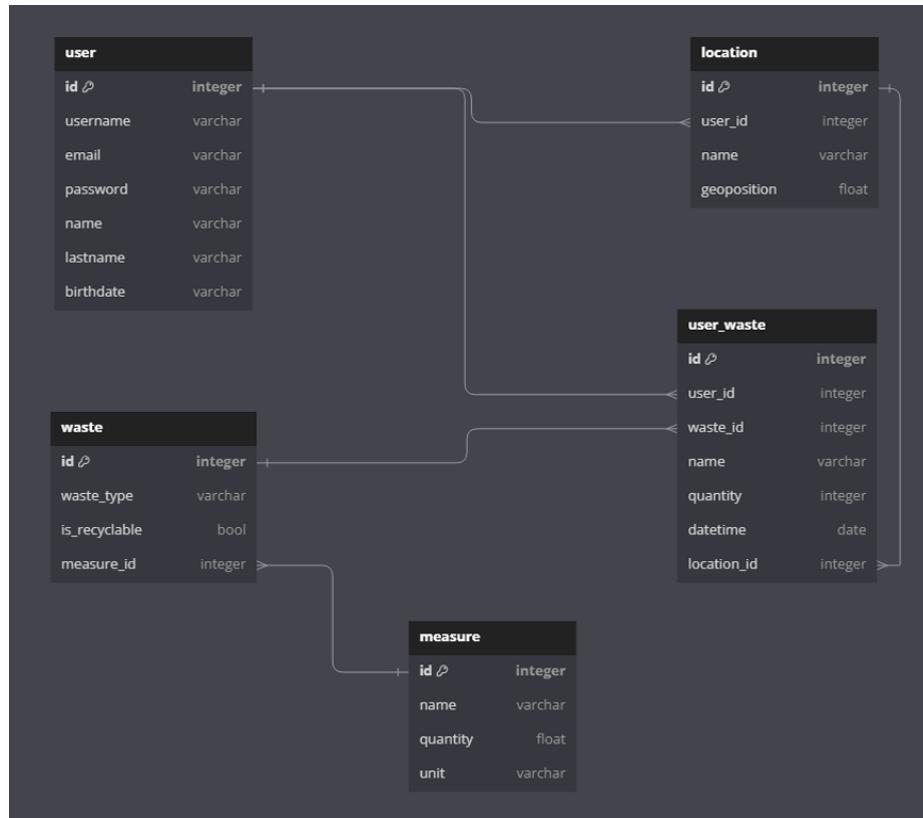


Figura 5.2: Diagrama entidad-relación de la segunda iteración.

### 5.3.2. API: Creación de funciones CRUD para ubicaciones y mediciones

En la **segunda iteración**, se ampliaron las funcionalidades del API para gestionar las nuevas tablas **location** y **measure**, permitiendo registrar de manera más detallada los residuos, junto con

la ubicación de donde provienen y las medidas asociadas.

Durante esta iteración, se implementaron los modelos y las relaciones entre todos los modelos existentes en el sistema principal. Para las tablas recién añadidas, se desarrollaron las funciones CRUD (Crear, Leer, Actualizar y Eliminar) de manera manual, facilitando un control directo sobre el acceso a la base de datos. Asimismo, se implementaron las primeras validaciones de datos para asegurar que los valores ingresados en estas tablas cumplieran con los requisitos establecidos, contribuyendo a la integridad y coherencia de la información en el sistema.

A continuación se presenta un ejemplo de función de validación:

```

1 // middleware/validateUser.js
2 const { body, validationResult } = require('express-validator');
3
4 // Validacion para el registro de usuarios
5 exports.validateRegister = [
6   body('email').isEmail().withMessage('Invalid email format'),
7   body('password').isLength({ min: 6 }).withMessage('Password must be at
8     least 6 characters long'),
9   (req, res, next) => {
10     const errors = validationResult(req);
11     if (!errors.isEmpty()) {
12       return res.status(400).json({ success: false, errors: errors.
13         array() });
14     }
15     next();
16   },
17 ];

```

Listing 5.2: Archivo validateUser.js

## 5.4. Tercera Iteración

### 5.4.1. Base de Datos: Diseño de tabla para la zona de la ciudad

En la **tercera iteración**, se introdujo una mejora significativa en la gestión de las ubicaciones de los registros de residuos. En lugar de utilizar coordenadas geográficas, se optó por un sistema basado en zonas dentro de la Ciudad de Guatemala, facilitando así la clasificación y administración de los desechos por zonas.

Para soportar esta funcionalidad, se añadió la tabla **area**, la cual está vinculada a **location** para que cada ubicación esté asociada a un área específica de la ciudad.

- **area**: Almacena información sobre las áreas geográficas dentro de la ciudad, incluyendo campos como **id**, el nombre de la ciudad, y el área específica, lo que facilita la identificación de zonas dentro del sistema.

A continuación, se presenta el diagrama actualizado de la estructura de la base de datos en la tercera iteración:

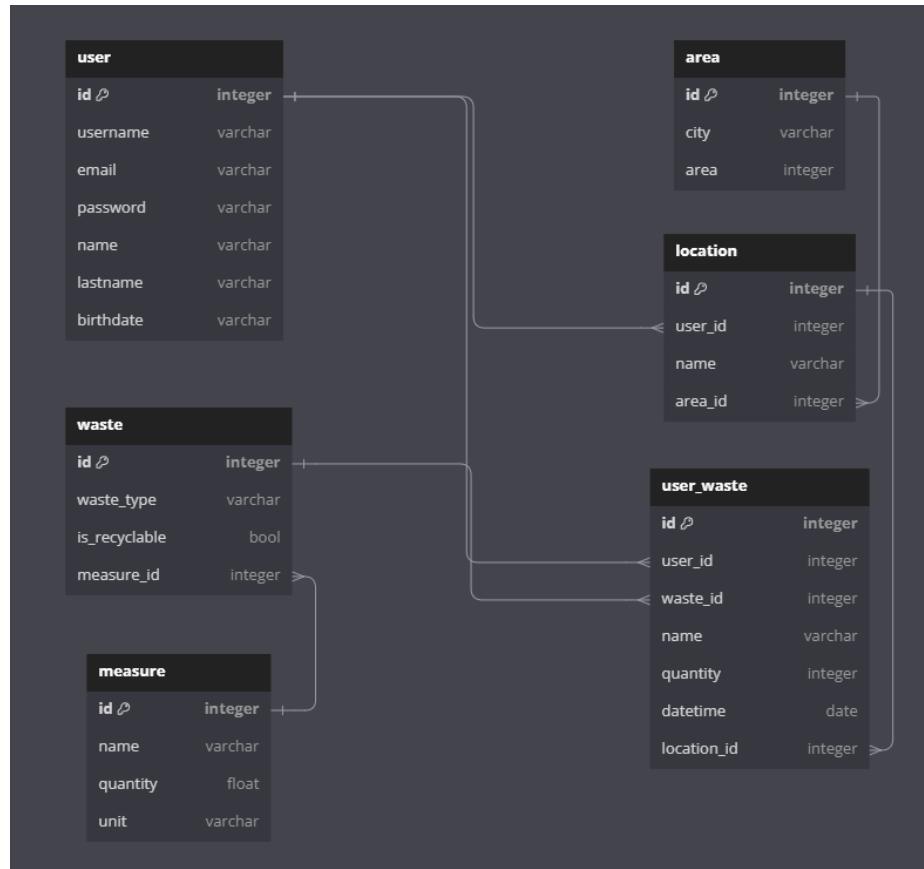


Figura 5.3: Diagrama entidad-relación de la tercera iteración.

#### 5.4.2. API: Creación de modelos y uso de ORM

Durante la **tercera iteración**, se realizó un cambio fundamental en la arquitectura del *API*, utilizando **Express.js** como el *framework* para el servidor y **Sequelize** como el *ORM* para la interacción con la base de datos PostgreSQL. Esto permitió mejorar la escalabilidad y eficiencia del sistema.

El código se organizó en carpetas específicas para garantizar una estructura modular:

- En la carpeta **config**, se creó el archivo **database.js**, que se encarga de la conexión a PostgreSQL y de la sincronización de los modelos con la base de datos.
- Los modelos de la base de datos se organizaron en la carpeta **models**, definiendo las propiedades de las tablas y las relaciones entre ellas, como en los archivos **User.js**, **Waste.js**, **Area.js**, **Location.js**, **Measure.js** y **UserWaste.js**.
- Los controladores para gestionar las operaciones CRUD sobre cada entidad se implementaron en la carpeta **controllers**
- Las rutas RESTful se definieron en la carpeta **routes**, permitiendo a los usuarios interactuar con la API.

A continuación se presenta un ejemplo de un modelo con Sequelize:

```

1 // models/Area.js
2 const { DataTypes } = require('sequelize');
3 const sequelize = require('../config/database');
4
5 const Area = sequelize.define('Area', {
6   id: {
7     type: DataTypes.INTEGER,
8     autoIncrement: true,
9     primaryKey: true,
10   },
11   city: {
12     type: DataTypes.STRING,
13     allowNull: false,
14   },
15   area: {
16     type: DataTypes.INTEGER,
17     allowNull: false,
18   },
19 }, {
20   tableName: 'area',
21   freezeTableName: true
22 });
23
24 module.exports = Area;

```

Listing 5.3: Archivo Area.js

## 5.5. Cuarta Iteración

### 5.5.1. Base de Datos: Cambios en mediciones de desechos

En la **cuarta iteración**, se realizaron cambios significativos en la estructura de la base de datos, reflejando un enfoque renovado para el ingreso y manejo de los datos de residuos por parte de los usuarios. Estos cambios se implementaron con el objetivo de proporcionar análisis y métricas más precisas y detalladas, permitiendo a los usuarios acceder a información más completa sobre sus hábitos de gestión de residuos.

Las modificaciones introducidas en esta iteración respondieron a la necesidad de simplificar el modelo de datos y ofrecer un análisis preciso de los registros de desechos proporcionados por el usuario.

La integración de los campos `measure_type` y `measure_unit` directamente en la tabla `waste` en la cuarta iteración simplifica el proceso de ingreso de datos y facilita el acceso a la información sin la necesidad de relaciones adicionales. Este cambio permite un acceso más rápido a las unidades de medida y tipos de residuos, lo que es crucial para el análisis en tiempo real y la presentación de métricas detalladas a los usuarios.

Además, la introducción del campo `average_weight` proporciona una solución más flexible para gestionar los residuos que se cuantifican en unidades, ofreciendo una mayor precisión en el cálculo de los residuos totales. Finalmente, la adición del campo `has_waste_collection` en la tabla `location` permite recopilar datos valiosos sobre la disponibilidad de servicios de recolección de basura, lo que enriquece los análisis sobre la accesibilidad y la gestión de residuos en diferentes ubicaciones.

A continuación se presenta el diagrama actualizado de la estructura de la base de datos en la

cuarta iteración:

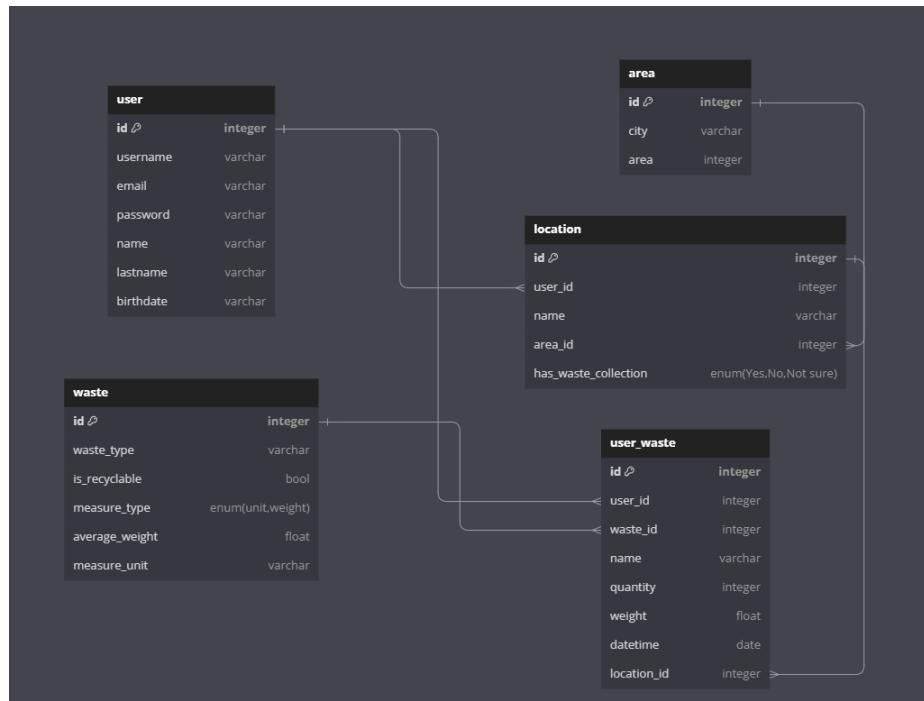


Figura 5.4: Diagrama entidad-relación de la cuarta iteración.

### 5.5.2. API: Ampliación de controladores

En esta iteración se implementaron diversas operaciones clave para asegurar un manejo adecuado de las operaciones del API y optimizar su funcionalidad.

Durante este proceso, se ampliaron las funciones de los controladores de cada uno de los modelos, atendiendo a los diferentes casos de uso previstos en la gestión de datos de los usuarios. Esto incluyó la incorporación de validaciones para cada entrada en las rutas del API, además de la implementación de un manejo de errores robusto y el uso de códigos de respuesta HTTP específicos para cada situación, asegurando así una comunicación clara y precisa con los clientes de la API.

Se implementaron también funciones de hash para almacenar las contraseñas de forma segura, preservando la integridad y privacidad de los datos de los usuarios. Mediante el uso de middleware, se estandarizaron las validaciones necesarias para las entradas de datos de los usuarios, logrando así una mayor consistencia y control en los datos procesados por el sistema.

Durante esta iteración se crearon dos módulos adicionales:

- En la carpeta `analysis`, se desarrolló el archivo `analysisController.js`, encargado de realizar las operaciones principales de análisis. Además, dentro de esta carpeta se incluyó un archivo adicional que proporciona las consultas necesarias al controlador.
- El middleware de validación para rutas de acceso se organizó en la carpeta `middleware` permitiendo una estructura más modular y facilitando la interacción de los usuarios con la API.

A continuación se presenta un ejemplo de un query de análisis de datos:

```

1 // Obtener estadísticas de desechos reciclables y no reciclables por usuario
2 // (ultimo mes)
3 exports.getRecyclableWasteStatsQuery = async (sequelize, userId) => {
4   const query = `
5     SELECT
6       COALESCE(SUM(CASE WHEN w.is_recyclable = TRUE THEN uw.weight
7           ELSE 0 END), 0) AS recyclable_weight,
8       COALESCE(SUM(CASE WHEN w.is_recyclable = FALSE THEN uw.weight
9           ELSE 0 END), 0) AS non_recyclable_weight
10    FROM user_waste uw
11    JOIN waste w ON uw.waste_id = w.id
12    WHERE uw.user_id = :userId
13    AND uw.datetime >= NOW() - INTERVAL '1 MONTH';
14  `;
15
16  const result = await sequelize.query(query, {
17    type: QueryTypes.SELECT,
18    replacements: { userId }
19  });
20
21  return {
22    recyclable_weight: result[0].recyclable_weight || 0,
23    non_recyclable_weight: result[0].non_recyclable_weight || 0
24  };
25};

```

Listing 5.4: Archivo userWasteAnalysis.js

### 5.5.3. Análisis de Datos: Creación de métricas iniciales

Para la **cuarta iteración** del análisis de datos, se consideró la estructura base de la aplicación con el objetivo de realizar un análisis sobre cómo la información almacenada podría proporcionar métricas significativas. Gracias a la colaboración de la Unidad de Reciclaje de la Municipalidad de Guatemala, se definieron las principales métricas que permitirían demostrar a los usuarios la importancia del reciclaje y la correcta clasificación de los desechos.

A continuación se presenta una foto de la reunión donde se mostró la estructura y se explicaron los datos que se guardaban de los usuarios:

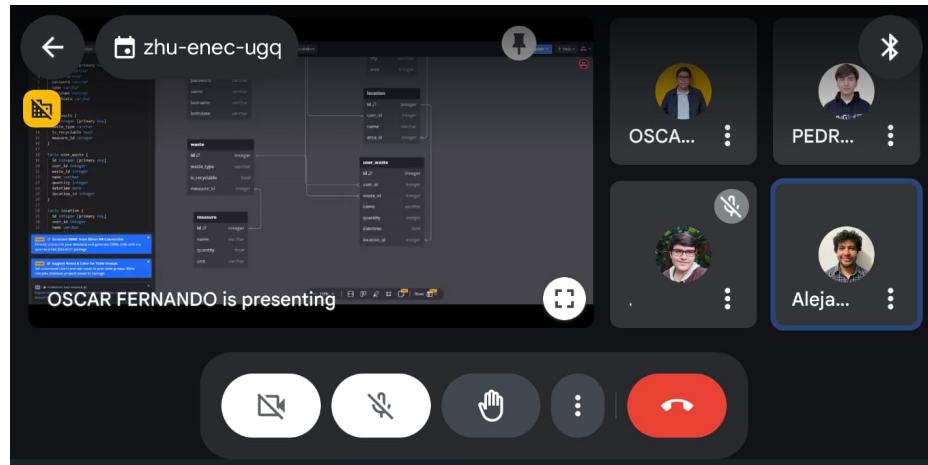


Figura 5.5: Reunión para Estructurar el Análisis de Datos.

Durante este proceso, se identificaron las siguientes métricas clave:

- **Análisis de la cantidad de agua ahorrada:** Calcula el impacto positivo en términos de ahorro de agua derivado de la correcta clasificación y reciclaje.
- **Análisis de la cantidad de emisiones de CO<sub>2</sub>:** Evalúa la reducción de emisiones de CO<sub>2</sub> lograda mediante las prácticas de reciclaje.
- **Análisis por ubicaciones del usuario:** Proporciona información detallada del impacto de reciclaje basado en las distintas ubicaciones registradas por el usuario.

## 5.6. Quinta Iteración

### 5.6.1. Base de Datos: Creación de tabla para tipos de desecho

En la **quinta iteración**, se realizó un cambio importante en la estructura de la base de datos al agregar una nueva tabla denominada **waste\_type**, que tiene como objetivo principal mejorar la capacidad de análisis de los datos relacionados con los tipos de residuos que maneja la aplicación. Este cambio fue motivado por la necesidad de tener una estructura más clara y eficiente para realizar análisis de impacto ambiental y para generar métricas más precisas que ayuden a los usuarios a tomar mejores decisiones sobre la gestión de sus residuos.

Esta nueva tabla almacena información detallada sobre los tipos de residuos con los que los usuarios pueden interactuar en la aplicación. Los campos principales incluyen:

- **id:** Identificador único para cada tipo de residuo.
- **type\_name:** Nombre del tipo de residuo (por ejemplo, plástico, vidrio, orgánico, etc.).
- **water\_savings\_index:** Índice de ahorro de agua asociado a este tipo de residuo, que permite calcular el impacto positivo en términos de ahorro de agua cuando el residuo es gestionado correctamente.
- **co2\_emission\_index:** Índice de emisiones de CO<sub>2</sub> asociado a la gestión de este tipo de residuo, utilizado para calcular el impacto ambiental en términos de emisiones de carbono.

A continuación se presenta el diagrama actualizado de la estructura de la base de datos en la quinta iteración:

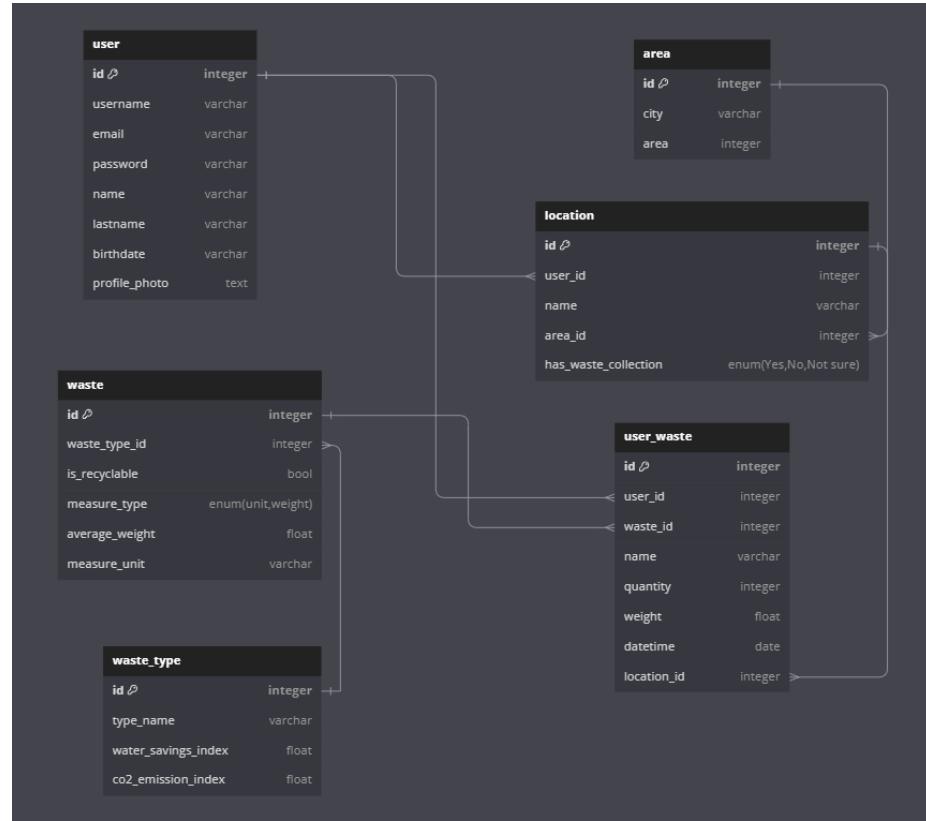


Figura 5.6: Diagrama entidad-relación de la quinta iteración.

### 5.6.2. API: Implementación de autenticación con JWT

Durante la **quinta iteración**, los esfuerzos se centraron en implementar autenticación para cada una de las rutas del API, utilizando JSON Web Tokens (JWT) para mejorar los procesos de seguridad en las peticiones hacia el sistema. Este mecanismo asegura que solo los usuarios autenticados puedan acceder a los recursos y realizar operaciones, fortaleciendo así la protección de los datos.

El enfoque principal consistió en configurar un middleware de autenticación que verificaría el token generado durante el proceso de inicio de sesión. Este token se incluye en la cabecera de cada solicitud y es validado por el middleware antes de permitir el acceso a las rutas. La configuración permite un control de acceso específico para cada usuario, ya que el token contiene información única que identifica al usuario y sus permisos.

Si una solicitud no contiene el token en la cabecera, o si el token es inválido o ha expirado, el sistema responde con un error de autenticación, evitando el acceso no autorizado a la información y resguardando los datos sensibles. Esta implementación asegura que cada recurso del API esté disponible únicamente para los usuarios que han pasado por el proceso de autenticación, manteniendo la privacidad y la integridad de los datos.

Este enfoque también establece una estructura flexible y escalable para futuras ampliaciones de seguridad, permitiendo la incorporación de diferentes niveles de acceso y políticas de autorización adicionales en próximas iteraciones.

A continuación se presenta la función de autenticación:

```

1 // middleware/jwtAuth.js
2 const jwt = require('jsonwebtoken');
3
4 const jwtAuth = (req, res, next) => {
5     // Definir las rutas que deseas excluir del middleware de autenticacion
6     const excludedPaths = ['/auth/token', '/user/login'];
7     // Excluir por rutas o por el tipo de entorno
8     if (excludedPaths.includes(req.path) || process.env.NODE_ENV !== 'production') {
9         return next(); // Omitir autenticacion para estas rutas
10    }
11
12    // Obtener el encabezado de autorizacion
13    const authHeader = req.header('Authorization');
14    // Verificar si el encabezado existe y tiene el formato adecuado
15    if (!authHeader?.startsWith('Bearer ')) {
16        return res.status(401).json({ error: 'Authorization header missing or malformed' });
17    }
18
19    // Extraer el token del encabezado
20    const token = authHeader.replace('Bearer ', '');
21    // Verificar el token JWT
22    try {
23        const decoded = jwt.verify(token, process.env.JWT_SECRET);
24        req.user = decoded; // Agregar los datos decodificados del usuario
25        next(); // Pasar al siguiente middleware o ruta
26    } catch (error) {
27        // Manejar errores especificos de JWT
28        if (error.name === 'TokenExpiredError') {
29            return res.status(401).json({ error: 'Token expired' });
30        }
31        if (error.name === 'JsonWebTokenError') {
32            return res.status(401).json({ error: 'Invalid token' });
33        }
34        // Otro tipo de error
35        return res.status(500).json({ error: 'Failed to authenticate token' });
36    }
37}
38 module.exports = jwtAuth;

```

Listing 5.5: Archivo jwtAuth.js

### 5.6.3. Análisis de Datos: Diseño de funciones principales

Durante la **quinta iteración** del análisis de datos se implementaron cambios significativos en la estructura de la base de datos, permitiendo así la realización de cálculos precisos sobre la reducción de emisiones de CO<sub>2</sub> y el ahorro de agua mediante el reciclaje y la clasificación de residuos. Este enfoque se inspiró en metodologías empleadas en países de Europa y América Latina, donde se aplica un factor específico al peso y al tipo de material de los residuos clasificados para evaluar el impacto ambiental.

Este enfoque resultó fundamental, ya que permitió agregar a la estructura de la base de datos una nueva tabla diseñada para el análisis específico de los materiales reciclados y clasificados. Ade-

más, proporciona flexibilidad para realizar estudios de impacto ambiental sobre las zonas donde se encuentran los usuarios, evaluando así cómo las buenas prácticas de gestión de residuos contribuyen a la sostenibilidad en distintas ubicaciones.

A continuación se demuestra el cálculo que se toma en cuenta para cada métrica:

**Ahorro de Agua:**

$$\text{Ahorro de Agua} = \text{Peso de la Basura} \times \text{Factor de Conversión de Agua} \quad (5.1)$$

**Emisiones de CO2:**

$$\text{Emisiones de CO2} = \text{Peso de la Basura} \times \text{Factor de Conversión de CO2} \quad (5.2)$$

## 5.7. Sexta Iteración

### 5.7.1. Base de Datos: Creación de tabla de bitácora de auditoría

En la **sexta iteración**, se introdujo una tabla adicional denominada **audit\_log**, cuyo propósito es servir como una bitácora de auditoría. Este cambio responde a la necesidad de llevar un registro detallado de las operaciones críticas realizadas en la base de datos, es fundamental para garantizar la integridad de los datos y para identificar posibles problemas relacionados con el acceso no autorizado o modificaciones incorrectas.

La bitácora de auditoría no solo facilita el seguimiento de los cambios, sino que también se convierte en una herramienta crucial para el cumplimiento de normativas y estándares de seguridad, especialmente en sistemas que manejan información sensible. Al mantener un historial de operaciones, se puede revisar en cualquier momento quién realizó una modificación, en qué momento y qué datos se vieron afectados, lo cual es esencial para el mantenimiento de la confiabilidad del sistema.

- **id:** Identificador único de cada registro de auditoría.
- **operation\_type:** Indica el tipo de operación realizada (CREATE, UPDATE, DELETE).
- **table\_name:** Nombre de la tabla en la cual se realizó la operación.
- **record\_id:** Identificador del registro afectado por la operación.
- **old\_values:** Almacena los valores anteriores a la operación (en caso de actualizaciones o eliminaciones).
- **new\_values:** Almacena los nuevos valores después de la operación (en caso de creaciones o actualizaciones).
- **performed\_by:** Identificador del usuario que realizó la operación.
- **created\_at:** Marca temporal que indica cuándo se realizó la operación.

A continuación se presenta el diagrama actualizado de la estructura de la base de datos en la sexta iteración:



Figura 5.7: Diagrama entidad-relación de la sexta iteración.

### 5.7.2. API: Implementación de registros en bitácora de auditoría

En la **sexta iteración**, se integraron funciones adicionales para gestionar el registro de auditoría mediante el modelo **audit\_log**. El objetivo principal de esta iteración fue mejorar la trazabilidad de las operaciones realizadas en el sistema, permitiendo registrar todas las acciones CRUD en las entidades clave de la base de datos para un seguimiento detallado de cada cambio.

Para implementar esta funcionalidad, se creó un nuevo módulo denominado **utils**, en el cual se definió una función principal llamada **logAudit**. Esta función se encarga de realizar el registro de cada operación de agregación, actualización y eliminación realizada en el API, generando un historial de cambios y facilitando la supervisión de las actividades del sistema.

Se realizaron actualizaciones en cada uno de los controladores para integrar esta funcionalidad, habilitando así el almacenamiento automático de los registros en la tabla **audit\_log** y garantizando que los datos se registren de forma consistente en cada operación relevante.

A continuación se presenta un ejemplo de la función:

```

1 // utils/auditLogger.js
2 const AuditLog = require('../models/AuditLog');
3
4 // Función para registrar las operaciones en la tabla audit_log

```

```

5 exports.logAudit = async (operation, table, recordId, oldValues, newValues,
6   performedBy) => {
7   try {
8     await AuditLog.create({
9       operation_type: operation,
10      table_name: table,
11      record_id: recordId,
12      old_values: oldValues ? JSON.stringify(oldValues) : null,
13      new_values: newValues ? JSON.stringify(newValues) : null,
14      performed_by: performedBy
15    });
16   } catch (error) {
17     console.error('Error logging audit:', error);
18   }
19 };

```

Listing 5.6: Archivo auditLogger.js

### 5.7.3. Análisis de Datos: Diseño de funciones de análisis

Durante la **sexta iteración**, se diseñaron nuevas funciones que permiten proporcionar a los usuarios datos detallados sobre sus comportamientos y patrones de consumo en relación con el reciclaje y la gestión de residuos. Estas funciones buscan informar a los usuarios sobre sus prácticas actuales y fomentar una mayor conciencia ambiental a través de un seguimiento preciso de sus hábitos.

En esta iteración se propuso un conjunto de métricas y funciones clave que se aplicarían al análisis de datos, incluyendo:

- **Análisis de la cantidad de desechos reciclables y no reciclables:** Proporciona un desglose de los residuos en función de la forma en la que fueron clasificados en el registro.
- **Análisis de los tipos de desechos más consumidos por el usuario:** Identifica los materiales que el usuario consume en mayor medida, facilitando el reconocimiento de hábitos específicos.
- **Análisis del promedio diario de desechos en la última semana:** Calcula el promedio diario de residuos gestionados por el usuario en los últimos siete días.
- **Análisis del promedio diario de desechos en el último mes:** Proporciona una visión más amplia del consumo promedio diario de desechos en el último mes.
- **Porcentaje de consumo con respecto a la última semana:** Evalúa el porcentaje de consumo actual del usuario en relación con el promedio de la semana anterior, permitiendo una comparación rápida de cambios en el comportamiento de gestión de residuos.

#### Porcentaje del consumo del día con respecto a la semana anterior:

$$\text{Porcentaje de Consumo} = \left( \frac{\text{Consumo del Día Actual}}{\text{Promedio de Consumo de la Semana Anterior}} \right) \times 100 \quad (5.3)$$

## 5.8. Séptima Iteración

### 5.8.1. Base de Datos: Optimización y mejora de la estructura

En la **séptima iteración**, se realizaron ajustes adicionales en las tablas **waste** y **user\_waste**, enfocados en simplificar la estructura de los datos, almacenando únicamente la información necesaria para llevar a cabo las acciones requeridas y realizar un análisis más eficiente de la gestión de residuos por parte de los usuarios. Estos cambios fueron impulsados por la necesidad de optimizar el rendimiento del sistema, eliminando cualquier redundancia y asegurando que solo se almacenaran los datos esenciales. Al reducir la cantidad de datos almacenados en esta tabla, se optimizó la capacidad de realizar consultas más rápidas sobre los registros de desechos del usuario.

Se eliminaron los campos **measure\_type** y **measure\_unit**, que anteriormente se usaban para gestionar las unidades de medida de los residuos. Estos campos ya no eran necesarios, dado que la estructura del sistema fue simplificada para centrarse en los tipos de residuos y sus relaciones con las métricas ambientales almacenadas en la tabla **waste\_type**.

Se eliminó el campo **quantity**, que anteriormente almacenaba la cantidad de residuos generados por los usuarios. Ahora, únicamente se mantiene el campo **weight**, que almacena el peso exacto del residuo, ya que este es el dato más relevante para el análisis de la huella de carbono y las métricas de ahorro de recursos.

Este ajuste permite simplificar el cálculo de métricas, ya que los análisis se enfocan en el peso total de los residuos gestionados, eliminando la necesidad de manejar cantidades adicionales que podrían generar complejidad en el procesamiento de los datos.

A continuación se presenta el diagrama actualizado de la estructura de la base de datos en la séptima iteración:



Figura 5.8: Diagrama entidad-relación de la séptima iteración.

### 5.8.2. API: Optimización y mejora del API

La **séptima iteración** se enfocó en ajustar los controladores y las rutas en función del diseño final de la base de datos, implementando las operaciones necesarias para desarrollar los controladores de análisis de datos y adaptando la estructura del sistema para una mayor coherencia y funcionalidad.

En esta fase, se realizaron cambios para alinear los modelos y relaciones de acuerdo con el esquema final. Estas modificaciones incluyeron ajustes en las asociaciones entre tablas, garantizando que los modelos reflejaran con precisión la nueva lógica y soportaran las consultas complejas necesarias para el análisis de datos.

Además, se implementó una función específica para obtener el token de autenticación de un usuario con privilegios administrativos. Esta funcionalidad permite el acceso seguro a rutas especiales y prepara el sistema para futuras integraciones con un tablero de visualización de datos. Con esta visión, se espera que el API pueda conectarse fácilmente a plataformas de análisis de datos en el futuro, proporcionando un flujo de datos seguro y directo hacia herramientas de visualización para el monitoreo y análisis en tiempo real.

Con esta iteración, el desarrollo del *API* se considera completo, cumpliendo con los requisitos establecidos y proporcionando una estructura sólida y escalable que soporta tanto la seguridad como

la integridad de los datos.

### 5.8.3. Análisis de Datos: Creación de las rutas de análisis en API

Durante esta iteración, los esfuerzos se centraron en adaptar y replicar las funciones y métricas en la API, permitiendo que los usuarios pudieran consultar y visualizar estos datos de manera accesible y dinámica.

Para lograr este objetivo, fue necesario desarrollar funciones en SQL que extrajeran datos específicos de cada usuario, asegurando que la información fuera precisa y relevante. Estas funciones de consulta se integraron en el API para que los datos se procesaran de manera eficiente y segura.

Además, se implementaron funciones auxiliares que ejecutan las consultas y procesan los resultados, proporcionando un flujo de datos optimizado para cada métrica definida. Finalmente, se crearon controladores específicos para cada una de las funciones de análisis propuestas, facilitando el acceso a través de rutas en la API y permitiendo una interacción fluida con la interfaz de usuario.

Este enfoque garantiza que todas las métricas y funciones analíticas puedan ser consultadas en tiempo real por los usuarios, proporcionando información clara y detallada sobre sus hábitos de consumo y reciclaje.

A continuación se muestra un ejemplo de la implementación de una de las funciones de análisis de datos:

```

1 // Obtener el ahorro de emisiones de CO2 para el usuario (ultimo mes)
2 exports.getCO2SavingsQuery = async (sequelize, userId) => {
3   const query = `
4     SELECT
5       COALESCE(SUM(uw.weight * wt.co2_emission_index), 0) AS
6         total_co2_savings
7     FROM user_waste uw
8     JOIN waste w ON uw.waste_id = w.id
9     JOIN waste_type wt ON w.waste_type_id = wt.id
10    WHERE uw.user_id = :userId
11    AND uw.datetime >= NOW() - INTERVAL '1 MONTH';
12  `;
13
14  const result = await sequelize.query(query, {
15    type: QueryTypes.SELECT,
16    replacements: { userId }
17  });
18
19  return result[0]?.total_co2_savings || 0;
}

```

Listing 5.7: Archivo userWasteAnalysis.js

## 5.9. Octava Iteración

### 5.9.1. Análisis de Datos: Creación de análisis de datos global de los usuarios

Durante esta última iteración del análisis de datos, los esfuerzos se centraron en proporcionar un análisis general sobre los usuarios y sus datos relacionados con la gestión de desechos. Para ello, se

adaptaron y extendieron las funciones previamente utilizadas para análisis individual, modificándolas para calcular métricas y estadísticas a nivel global.

Para facilitar una visualización clara y comprensible de estos resultados agregados, y con el objetivo de analizar los comportamientos y registros de una población específica, se decidió implementar un tablero en **Power BI**. Este tablero se conecta directamente con el API, permitiendo presentar los datos en gráficos interactivos y visualizaciones fáciles de interpretar. De esta forma, se simplifica el análisis y seguimiento de los datos, brindando una herramienta eficaz para identificar tendencias y patrones en la gestión de desechos.

A continuación se presenta un ejemplo de las funciones globales:

```

1 // Obtener estadísticas de desechos reciclables y no reciclables de todos
2 // los usuarios (ultimo mes)
3 exports.getRecyclableWasteStatsQueryAll = async (sequelize) => {
4     const query = `
5         SELECT
6             COALESCE(SUM(CASE WHEN w.is_recyclable = TRUE THEN uw.weight
7                             ELSE 0 END), 0) AS recyclable_weight,
8             COALESCE(SUM(CASE WHEN w.is_recyclable = FALSE THEN uw.weight
9                             ELSE 0 END), 0) AS non_recyclable_weight
10        FROM user_waste uw
11        JOIN waste w ON uw.waste_id = w.id
12        WHERE uw.datetime >= NOW() - INTERVAL '1 MONTH';
13    `;
14
15    const result = await sequelize.query(query, {
16        type: QueryTypes.SELECT
17    });
18
19    return {
20        recyclable_weight: result[0].recyclable_weight || 0,
21        non_recyclable_weight: result[0].non_recyclable_weight || 0
22    };
23};

```

Listing 5.8: Archivo userWasteAnalysis.js

Y se presenta el ejemplo de conexión por medio de Power BI:

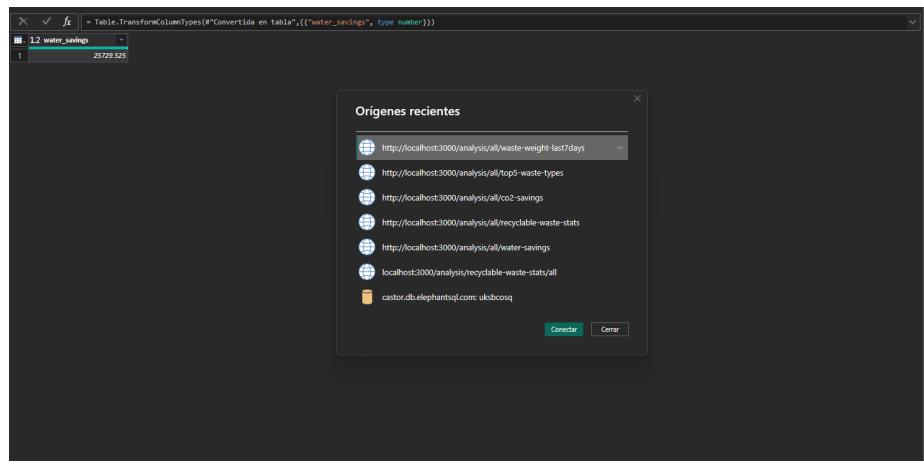


Figura 5.9: Ejemplo de solicitudes en orígenes de conexión.

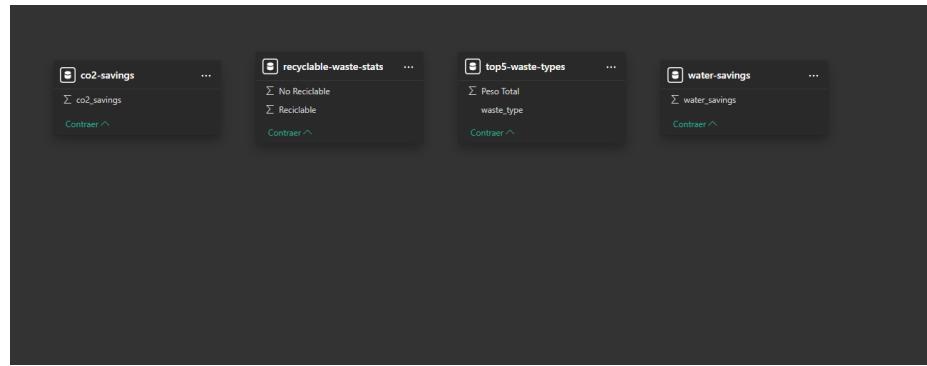


Figura 5.10: Tablas resultantes de la conexión del API.

## 5.10. Implementación de CI/CD

Al completar las tres principales secciones necesarias para almacenar, gestionar y proporcionar datos a los usuarios, los esfuerzos se orientaron hacia la creación de una base sólida para el desarrollo y expansión futuros del *backend*. Mediante un análisis exhaustivo, se decidió adoptar buenas prácticas de desarrollo de software, las cuales contribuyen a mantener un código limpio, modular y sostenible a largo plazo.

Por lo tanto, se implementó un pipeline de Integración y Despliegue Continuo (CI/CD), permitiendo la automatización de pruebas, la detección temprana de errores y un proceso de despliegue optimizado. Esta estructura facilita una gestión eficiente del proyecto, al tiempo que prepara el sistema para futuras iteraciones y ampliaciones de la aplicación. Con estas prácticas, se estableció una infraestructura robusta que no solo agiliza el desarrollo actual, sino que también sienta las bases para un crecimiento escalable y una evolución continua de la aplicación.

Ejemplo del Pipeline de CI/CD:

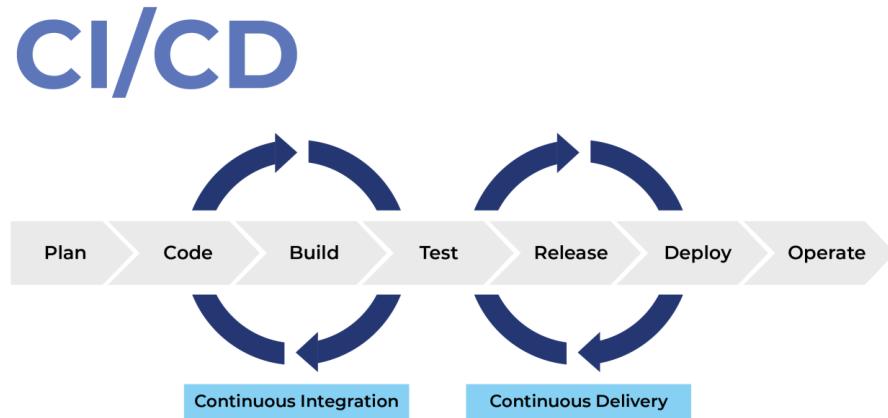


Figura 5.11: Pipeline de CI/CD [13].

### 5.10.1. Pruebas Automatizadas

La primera etapa para la implementación del *pipeline CI/CD*, una vez finalizado el desarrollo del código, fue la de construcción y prueba. Esta etapa es fundamental para verificar el correcto funcionamiento del sistema después de cada integración de código, permitiendo identificar y corregir cualquier error que pudiera surgir. La construcción asegura que el entorno y las dependencias se configuren adecuadamente, mientras que la fase de pruebas valida la estabilidad y fiabilidad del código.

Durante esta etapa, se diseñaron, implementaron y configuraron tres tipos de pruebas fundamentales para obtener información detallada sobre el comportamiento del API y asegurar su fiabilidad en la gestión y procesamiento de datos. Estas pruebas permiten evaluar el rendimiento del sistema bajo diversas condiciones de uso, verificando su capacidad para operar de manera consistente y precisa.

#### Pruebas Unitarias

Durante esta etapa de pruebas, se desarrollaron y estructuraron pruebas unitarias específicas para validar el funcionamiento de las funciones clave del API. Estas pruebas se centraron principalmente en verificar el comportamiento de los controladores encargados de gestionar la información, asegurando que cada controlador procesara los datos de forma precisa y consistente.

Se definieron diversos escenarios para simular posibles situaciones de uso, incluyendo casos en los que el usuario proporciona valores incorrectos o fuera de los parámetros esperados. En estos casos, las pruebas unitarias evaluaron cómo los controladores respondían a entradas erróneas, tomando en cuenta el tipo de error, la validación de los datos, y los mensajes de respuesta generados. Esto permitió identificar y manejar de forma efectiva los errores, asegurando que el sistema ofrezca una respuesta confiable.

Para construir estas pruebas, se utilizaron varias dependencias de Node.js que facilitaron las simulaciones necesarias para verificar el correcto funcionamiento de los controladores. La lista de dependencias empleadas es la siguiente:

- **Sinon.js:** Esta herramienta fue fundamental para la simulación de tablas y funciones durante las pruebas. Sinon permite realizar *mocks* y *stubs* de funciones y tablas de la base de datos, lo que facilita la creación de datos de prueba sin necesidad de interactuar con la base de datos real.
- **Chai.js:** Esta biblioteca de aserciones permite comparar los datos de prueba o simulados con los resultados obtenidos de las funciones. Su sintaxis facilita una validación clara y legible de los resultados esperados y reales.
- **Mocha.js:** Este marco de pruebas facilita la organización y ejecución de pruebas unitarias e integración, generando reportes detallados sobre los resultados. En caso de fallos, Mocha proporciona mensajes de error específicos que ayudan a identificar los problemas en el código.

A continuación se muestra un ejemplo de una prueba para el controlador de usuario:

```

1 it('Deberia crear un nuevo usuario y registrar auditoria', async () => {
2   const req = { body: { email: 'test@example.com', password: 'password123' },
3     user: { id: 999 } };
4   const res = { status: sinon.stub().returnsThis(), json: sinon.stub() };
5   const userMock = { id: 1, email: 'test@example.com' };

```

```

6   sinon.stub(User, 'findOne').resolves(null); // Simula que el email no
    esta en uso.
7   sinon.stub(User, 'create').resolves(userMock);
8   sinon.stub(AuditLog, 'create').resolves(); // Simula la creacion del
    registro de auditoria
9
10  await userController.createUser(req, res);
11
12  expect(res.status.calledWith(201)).to.be.true;
13  expect(res.json.calledWith(sinon.match.has('data'))).to.be.true;
14  expect(AuditLog.create.calledOnce).to.be.true; // Verifica que se
    registro la auditoria
15 });

```

Listing 5.9: Archivo userControllerTest.js

## Pruebas de Integración

Como siguiente paso en el proceso de pruebas, se procedió al diseño y ejecución de las pruebas de integración. Estas pruebas se enfocaron en evaluar la interacción entre las rutas de la API, los valores requeridos, y las funciones de los controladores, garantizando una integración coherente y fluida de los distintos componentes del sistema. A través de estas pruebas, se verificaron los flujos completos de datos y las respuestas que recibirán los usuarios al utilizar el API, permitiendo validar el funcionamiento del sistema de extremo a extremo.

A diferencia de las pruebas unitarias, que evalúan componentes aislados, las pruebas de integración comprueban la interacción de múltiples elementos, simulando el flujo real que el usuario experimentaría al interactuar con el sistema. Estas pruebas también se diseñaron para confirmar que los mensajes de respuesta, tanto de éxito como de error, se generen de manera clara y consistente. Esto incluye la verificación de que los mensajes de error contengan información útil y adecuada para guiar al usuario en caso de entradas incorrectas, sin exponer detalles técnicos innecesarios.

Para construir estas pruebas, se utilizaron varias dependencias de Node.js que facilitaron las simulaciones necesarias para verificar el correcto funcionamiento de las integración de los controladores con las rutas que el API brinda. La lista de dependencias empleadas es la siguiente:

- **Sinon.js:** Esta herramienta fue fundamental para la simulación de tablas y funciones durante las pruebas. Sinon permite realizar *mocks* y *stubs* de funciones y tablas de la base de datos, lo que facilita la creación de datos de prueba sin necesidad de interactuar con la base de datos real.
- **Chai.js:** Esta biblioteca de aserciones permite comparar los datos de prueba o simulados con los resultados obtenidos de las funciones. Su sintaxis facilita una validación clara y legible de los resultados esperados y reales.
- **Mocha.js:** Este marco de pruebas facilita la organización y ejecución de pruebas unitarias e integración, generando reportes detallados sobre los resultados. En caso de fallos, Mocha proporciona mensajes de error específicos que ayudan a identificar los problemas en el código.
- **Supertest.js:** Esta herramienta permite simular las solicitudes HTTP realizadas al API, proporcionando información detallada sobre el comportamiento de las respuestas y los resultados obtenidos. Es esencial para verificar el funcionamiento y la correcta respuesta del API ante diferentes tipos de solicitudes.

A continuación se presenta un ejemplo de prueba de integración:

```

1 it('Deberia iniciar sesion correctamente', async () => {
2   const userMock = { id: 1, username: 'testuser', email: 'test@example.com',
3     password: 'hashedPassword' };
4
5   // Simulamos la busqueda del usuario y la comparacion de passwords
6   sinon.stub(User, 'findOne').resolves(userMock);
7   sinon.stub(bcrypt, 'compare').resolves(true); // Simulamos que la
8   // password es valida
9
10  const res = await request(app)
11    .post('/api/users/login')
12    .send({
13      email: 'test@example.com',
14      password: 'password123'
15    });
16
17  expect(res.status).to.equal(200);
18  expect(res.body.success).to.be.true;
19  expect(res.body.data.username).to.equal('testuser');
20});

```

Listing 5.10: Archivo userRoutesTest.js

## Pruebas de Carga y Rendimiento

Para evaluar el comportamiento de la API bajo condiciones de alta demanda y monitorear su rendimiento en diversos escenarios, se implementaron pruebas de carga y rendimiento. Estas pruebas permiten analizar la capacidad de respuesta de la API, midiendo el tiempo que tarda en procesar solicitudes y la eficacia de sus respuestas bajo distintos niveles de carga. Además, proporcionan información valiosa sobre la estabilidad y escalabilidad del sistema, aspectos fundamentales para asegurar un rendimiento confiable y adecuado.

En estas pruebas, se configuró un entorno de prueba específico, cargando datos representativos en la base de datos y simulando un tráfico de solicitudes similar al uso real que podría experimentar la API en un ambiente de producción. Al iniciar la API en este entorno, se envían múltiples solicitudes simultáneas, simulando condiciones de carga que varían desde niveles bajos hasta picos de alta demanda. Esto permite observar cómo el sistema maneja la concurrencia y si existen cuellos de botella que afectan el rendimiento.

Durante las pruebas de carga, se registran métricas clave, como el tiempo promedio de respuesta, el porcentaje de solicitudes exitosas, y los tiempos de espera en diferentes endpoints, brindando datos que ayudan a identificar y optimizar los puntos críticos en la arquitectura de la API.

### Cálculo de Solicitudes por Segundo para una Aplicación Móvil

Para determinar la cantidad de solicitudes que la API debe soportar por segundo en un entorno de alta demanda, se realizó el siguiente cálculo basado en la población objetivo y un supuesto de uso promedio. Este cálculo se basa en una estimación de la población de la Ciudad de Guatemala en 2022, obtenida de [3].

**Paso 1: Cálculo de Usuarios Activos** Dado que la población total es:

$$P = 1,213,651$$

Supongamos una tasa de adopción del 20 % de la población. Esto nos da:

$$U = P \times 0.20$$

Sustituyendo el valor de  $P$ :

$$U = 1,213,651 \times 0.20 = 242,730 \text{ usuarios activos}$$

**Paso 2: Cálculo de Solicitudes Diarias** Si los usuarios realizan en promedio 5 solicitudes por día, la cantidad total de solicitudes al día es:

$$SD = U \times 5$$

Sustituyendo el valor de  $U$ :

$$SD = 242,730 \times 5 = 1,213,650 \text{ solicitudes diarias}$$

**Paso 3: Distribución de Solicitudes por Segundo** Para calcular la cantidad de solicitudes por segundo, dividimos las solicitudes diarias por el número de segundos en un día (24 horas multiplicado por 60 minutos y 60 segundos):

$$SPS = \frac{SD}{24 \times 60 \times 60}$$

Sustituyendo el valor de  $SD$ :

$$SPS = \frac{1,213,650}{24 \times 60 \times 60} \approx 14 \text{ solicitudes por segundo}$$

Este cálculo permite establecer un objetivo claro para las pruebas de carga, asegurando que la API esté preparada para manejar un promedio de 14 solicitudes por segundo en condiciones de alta demanda. Esta preparación garantiza que el sistema pueda escalar de manera eficiente, ofreciendo un rendimiento estable y óptimo para los usuarios en producción.

A continuación se muestra el ejemplo de una prueba para las rutas de análisis de datos:

```

1 config:
2   target: 'http://localhost:3000'
3   phases:
4     - duration: 60 # Duracion de la prueba en segundos
5       arrivalRate: 20 # Numero de usuarios simulados por segundo
6   defaults:
7     headers:
8       content-type: "application/json"
9
10 scenarios:
11   - name: Test de analisis de desechos reciclables
12     flow:
13       - get:
14         url: "/analysis/recyclable-waste/1"
15   - name: Test de las 5 principales ubicaciones por desechos
16     flow:
17       - get:
18         url: "/analysis/top5-locations/1"
19   - name: Test de ahorro de agua
20     flow:
21       - get:
22         url: "/analysis/water-savings/1"
23   - name: Test de ahorro de CO2
24     flow:
25       - get:
26         url: "/analysis/co2-savings/1"
```

```

27      - name: Test de los 5 tipos de desechos mas comunes
28          flow:
29              - get:
30                  url: "/analysis/top5-waste-types/1"
31      - name: Test de peso de desechos de los ultimos 7 dias
32          flow:
33              - get:
34                  url: "/analysis/waste-last7days/1"
35      - name: Test de comparacion de desechos de hoy con el promedio mensual
36          flow:
37              - get:
38                  url: "/analysis/compare-waste-today/1"

```

Listing 5.11: Archivo performance.yml

### 5.10.2. Manejo y Monitoreo de Errores con Sentry

Para asegurar un monitoreo continuo del rendimiento y detectar errores en el funcionamiento de la API, se decidió integrar una herramienta especializada: Sentry. Sentry permite realizar un seguimiento exhaustivo de los errores en tiempo real, capturando información detallada sobre los fallos y excepciones que puedan surgir en el entorno de producción, los cuales podrían no haberse detectado en las pruebas iniciales.

Durante el proceso de integración de Sentry, se configuraron diversas funcionalidades de monitoreo, como el *profiling*, que permite analizar el rendimiento y los tiempos de respuesta en tiempo real, y el monitoreo automático de errores, que facilita la detección de fallos en el sistema. Sin embargo, debido a que la estructura de los controladores ya capturaba los errores de forma interna, estos no generaban mensajes que Sentry pudiera interceptar automáticamente, lo cual limitaba la visibilidad de los problemas en la herramienta de monitoreo.

Para solucionar este inconveniente, se implementó una función personalizada, `captureError`, en cada controlador donde se manejaban errores. Esta función permite enviar manualmente a Sentry la información relevante sobre cada error, incluso cuando este es capturado y manejado en el código. Al utilizar `captureError`, se asegura que los errores que ocurren en la API sean registrados en Sentry con el contexto completo del fallo, incluyendo detalles como el controlador específico, el endpoint afectado, y los datos de la solicitud.

Para realizar la configuración de Sentry se hicieron las configuraciones correspondientes en la plataforma.

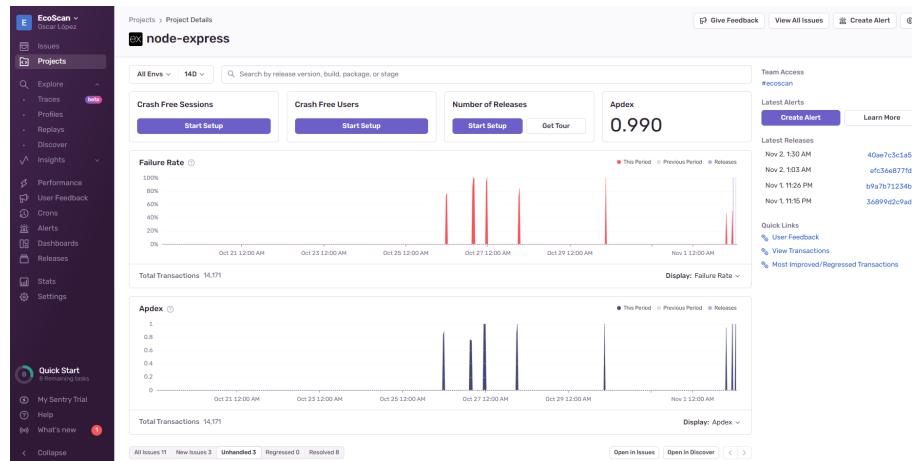


Figura 5.12: Tablero del Proyecto en Sentry.

Y se realizaron las configuraciones correspondientes en el API, para poder conectar la herramienta y capturar los errores.

Configuración principal de Sentry en el proyecto:

```

1 // middleware/instrument.js
2 require('dotenv').config(); // Cargar las variables de entorno
3
4 const Sentry = require("@sentry/node");
5 const { nodeProfilingIntegration } = require("@sentry/profiling-node");
6
7 Sentry.init({
8   dsn: process.env.SENTRY_DSN, // Usar la variable de entorno
9   integrations: [
10     nodeProfilingIntegration(),
11   ],
12   tracesSampleRate: 1.0,
13   profilesSampleRate: 1.0,
14 });

```

Listing 5.12: Archivo instrument.js

Ejemplo de captura de errores:

```

1 // middleware/captureError.js
2 const Sentry = require("@sentry/node");
3
4 // Funcion para capturar y manejar errores con Sentry
5 const captureError = (error) => {
6   // Verificar si el entorno no es de pruebas
7   if (process.env.NODE_ENV !== 'test') {
8     // Capturar el error con Sentry solo si no es un entorno de pruebas
9     Sentry.captureException(error);
10   }
11 };
12
13 module.exports = captureError;

```

Listing 5.13: Archivo captureError.js

### 5.10.3. Implementación y Automatización con GitHub Actions

Por último, en el proceso de implementación de un *Pipeline* de CI/CD se consideró la automatización de tareas clave para asegurar la calidad y el correcto funcionamiento del API. En este caso, se utilizó **GitHub Actions** debido a su integración con el repositorio en GitHub, donde se encuentra almacenado el código principal del API. Esta integración permite que, al realizar cambios en las funcionalidades del API, dichos cambios se sometan automáticamente a un conjunto de pruebas, asegurando que las nuevas implementaciones o modificaciones mantengan la estabilidad y calidad del sistema.

La configuración del *pipeline* se llevó a cabo mediante la creación de *workflows* en GitHub, los cuales definen los pasos necesarios para instalar dependencias, ejecutar pruebas, y generar reportes.

Para la ejecución, se configuraron dos *workflows*. En el primero, se llevan a cabo las pruebas unitarias y de integración, las cuales validan la funcionalidad de cada componente del API y verifican la interacción entre los distintos módulos del sistema. En el segundo *workflow*, se ejecutan las pruebas de carga y rendimiento, evaluando la capacidad del API para manejar múltiples solicitudes simultáneas y midiendo su desempeño bajo diferentes niveles de carga.

A continuación se muestra una parte de las configuraciones para los dos flujos:

```

1 - name: Navigate to eco-scan-backend folder
2   run: cd eco-scan-backend
3
4 - name: Install dependencies
5   run: npm install
6   working-directory: ./eco-scan-backend
7
8 - name: Set up environment variables
9   run: |
10    // Realizar la configuracion de variables de entorno
11   working-directory: ./eco-scan-backend
12
13 - name: Wait for Postgres
14   run: |
15     until pg_isready -h localhost -p 5432 -U postgres; do echo waiting for
16       postgres; sleep 5; done
17
18 - name: Run tests
19   run: npm run test
20   working-directory: ./eco-scan-backend
21
22 - name: Show logs if test fails
23   if: failure()
24   run: cat ./eco-scan-backend/npm-debug.log

```

Listing 5.14: Archivo test-deploy.yml

```

1 deploy:
2   needs: test
3   runs-on: ubuntu-latest
4   if: github.ref == 'refs/heads/master'
5
6 steps:
7 - name: Checkout code
8   uses: actions/checkout@v3
9
10 - name: Set up Node.js

```

```
11 uses: actions/setup-node@v3
12   with:
13     node-version: '20'
14
15 - name: Deploy to Elastic Beanstalk
16   env:
17     AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
18     AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
19     AWS_REGION: 'us-east-1'
20   run:
21     zip -r deploy.zip /*
22     aws elasticbeanstalk create-application-version --application-name "appName" --version-label version-1 --source-bundle S3Bucket="s3Name",S3Key="deploy.zip"
23     aws elasticbeanstalk update-environment --application-name "appName" --environment-name "appEnv" --version-label version-1
24   working-directory: ./eco-scan-backend
```

Listing 5.15: Archivo test-deploy.yml

# CAPÍTULO 6

---

## Resultados

---

El desarrollo del módulo se completó en las fases establecidas en la metodología, y cada iteración contribuyó a obtener resultados concretos que demuestran la funcionalidad del API para gestionar, proporcionar y transformar la información de los usuarios.

A continuación, se presentan los resultados obtenidos en cada apartado:

### 6.1. Diseño y Estructura de la Base de Datos

A través de las iteraciones, se logró definir una estructura final optimizada para gestionar los datos de los usuarios, desechos y ubicaciones relacionadas. Como resultado final, se establecieron las siguientes siete tablas principales:

- **User:** Guarda la información personal de los usuarios.
- **Area:** Almacena la información de las zonas disponibles para las ubicaciones.
- **Location:** Registra las ubicaciones con respecto a la zona y el usuario asociado.
- **Waste Type:** Almacena los tipos de desechos y sus propiedades para el análisis del impacto ambiental.
- **Waste:** Relaciona el tipo de desecho y guarda información detallada sobre un desecho específico.
- **User Waste:** Registra todos los datos de los residuos gestionados por el usuario.
- **Audit Log:** Registra todas las operaciones realizadas de creación, actualización y eliminación en la base de datos.

A continuación, se muestra el diagrama final de la estructura de la base de datos:

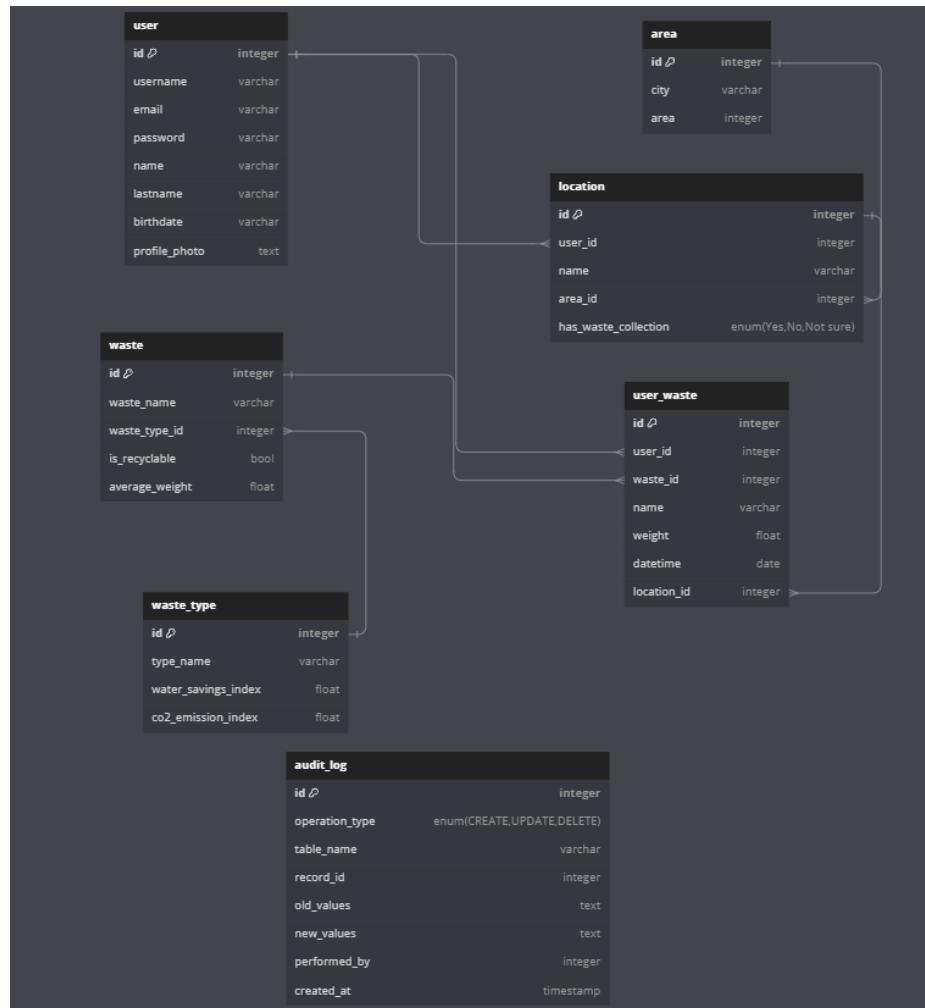


Figura 6.1: Diagrama entidad-relación final.

## 6.2. API

A lo largo de las iteraciones, se desarrollaron los módulos del API, cada uno contribuyendo a una gestión eficiente de los datos y asegurando que el API responda adecuadamente a las solicitudes de información.

Los módulos finales son los siguientes:

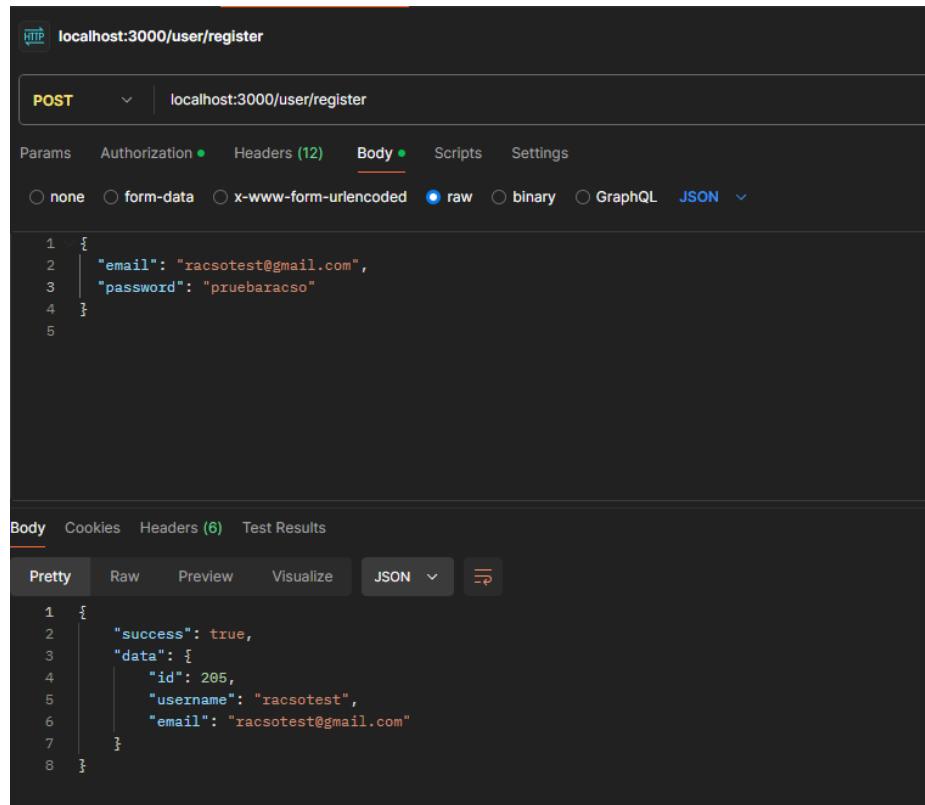
- **analysis**: Proporciona las funciones y consultas necesarias para el análisis de datos del sistema.
- **config**: Gestiona la conexión a la base de datos, garantizando un acceso estable y seguro.
- **controllers**: Contiene las funciones principales que ejecutan las operaciones en cada una de las tablas de la base de datos.
- **middleware**: Incluye funciones auxiliares que facilitan el funcionamiento del API, tales como el monitoreo y la autenticación de usuarios.
- **models**: Define los modelos que representan la estructura de las tablas en la base de datos.

- **routes:** Gestiona las rutas que exponen la información del API, permitiendo acceder a los recursos de manera estructurada.
- **test:** Almacena las pruebas unitarias, de integración y de rendimiento, que verifican la funcionalidad y estabilidad del sistema.
- **utils:** Contiene funciones auxiliares, como la encargada de registrar las entradas en la bitácora de auditoría.

### 6.2.1. Resultados de Pruebas de Funcionalidad

Se realizaron pruebas de funcionalidad para verificar el correcto desempeño del API en distintas operaciones. Los resultados de algunas de las rutas principales para el funcionamiento del API se muestran a continuación:

- **Registro de usuarios:**



```

POST      localhost:3000/user/register
Params   Authorization  Headers (12)  Body  Scripts  Settings
none    form-data  x-www-form-urlencoded  raw  binary  GraphQL  JSON
1 {
2   "email": "racsotest@gmail.com",
3   "password": "pruebaracso"
4 }
5

Body  Cookies  Headers (6)  Test Results
Pretty  Raw  Preview  Visualize  JSON
1 {
2   "success": true,
3   "data": {
4     "id": 206,
5     "username": "racsotest",
6     "email": "racsotest@gmail.com"
7   }
8 }

```

Figura 6.2: Registro de usuarios.

- **Login de usuarios:**

The screenshot shows a POST request to `localhost:3000/user/login`. The JSON body contains:

```

1 {
2   "email": "racsotest@gmail.com",
3   "password": "pruebaracso"
4 }
5

```

The response status is `200 OK` with the following JSON data:

```

1 {
2   "success": true,
3   "data": [
4     {
5       "id": 285,
6       "username": "racsotest",
7       "email": "racsotest@gmail.com"
8     },
9     "token": "eyJhbGciOiJIUzI1NiIsInR5cGVCIkxVQ39-eyJpZC16MjA1LCJlWFrpbC16InJhY3NvdGVoE8nbWFpbG5jb2g1LClpYXQ1OjE3MzA2MTIxNjQsIiV4cC16MTczNDYxNTc2NjB. C1M6uNVu6zehqz@1x1o3uxdKt-j71NZHFOz#J3Y10X8"
10   ]
11 }
12

```

Figura 6.3: Login de usuarios.

#### ■ Registro en la bitácora:

A screenshot of a database audit log table showing a single row of data:

id	operation_type	table_name	record_id	old_values	new_values	performed_by	timestamp
1	CREATE	user	211	[NULL]	{"\\"id"\":211,"\\username\"\":\\\"joji\\\"}	211	03:00:07:09.765 -0600

Figura 6.4: Registro en la bitácora de auditoría.

#### ■ Acceso no autorizado:

The screenshot shows a GET request to `localhost:3000/analysis/co2-savings/1`. The Authorization header is set to `Bearer Token`. The response status is `401 Unauthorized` with the following JSON error message:

```

1 {
2   "error": "Authorization header missing or malformed"
3 }

```

Figura 6.5: Intento de acceso no autorizado.

#### ■ Consulta de análisis:

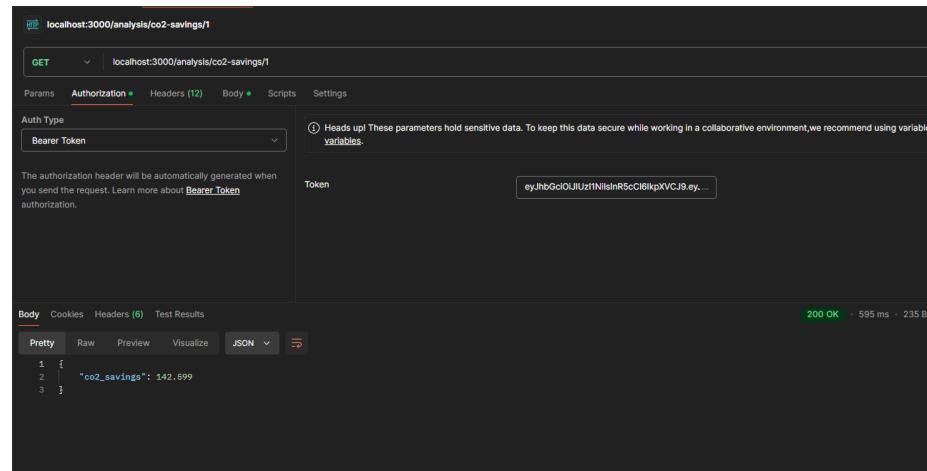


Figura 6.6: Consulta de análisis para usuarios autorizados.

### 6.2.2. Pruebas Unitarias y de Integración

Con el objetivo de asegurar el correcto funcionamiento de todas las rutas y funcionalidades del API, se llevaron a cabo pruebas unitarias y de integración. Estas pruebas permitieron validar cada componente tanto de forma individual como en conjunto, garantizando que el sistema funcione conforme a lo esperado. Las pruebas se aplicaron directamente a los controladores de todos los modelos y a todas las rutas que proporciona el API.

En total, se realizaron 220 pruebas, todas las cuales se ejecutaron exitosamente, confirmando la efectividad y robustez del sistema. A continuación, se presentan los resultados detallados:



The screenshot shows a terminal window with the title "Run tests". The output of the command `npm test` is displayed, showing test results for various routes in the API. The routes are categorized into "Analysis Routes" and "Area Routes". Each route has several test cases with their expected behavior.

```
4
5 > eco-scan-backend@1.0.0 test
6 > mocha --recursive --exclude "test/reports/**"
7
8
9
10 Analysis Routes
11   GET /api/analysis/recyclable-waste/:userId
12     ✓ Debería devolver las estadísticas de desechos reciclables
13     ✓ Debería devolver un error 500 si ocurre un problema en el proceso
14   GET /api/analysis/top5-locations/:userId
15     ✓ Debería devolver las 5 ubicaciones principales con más cantidad de basura registrada
16     ✓ Debería devolver una lista vacía si no se encuentran registros
17     ✓ Debería devolver un error 500 si ocurre un problema en el proceso
18   GET /api/analysis/water-savings/:userId
19     ✓ Debería devolver el ahorro de agua del usuario
20     ✓ Debería devolver 0 si no hay ahorro de agua
21     ✓ Debería devolver un error 500 si ocurre un problema en el proceso
22   GET /api/analysis/co2-savings/:userId
23     ✓ Debería devolver el ahorro de CO2 del usuario
24     ✓ Debería devolver 0 si no hay ahorro de CO2
25     ✓ Debería devolver un error 500 si ocurre un problema en el proceso
26   GET /api/analysis/waste-last7days/:userId
27     ✓ Debería devolver la cantidad de basura ingresada en los últimos 7 días
28     ✓ Debería devolver una lista vacía si no se encuentran registros
29     ✓ Debería devolver un error 500 si ocurre un problema en el proceso
30   GET /api/analysis/compare-waste-today/:userId
31     ✓ Debería devolver un mensaje cuando no hay suficientes datos para calcular el promedio
32     ✓ Debería devolver un error 500 si ocurre un problema en el proceso
33   GET /api/analysis/waste-today/:userId
34     ✓ Debería devolver la cantidad de basura desechara hoy
35     ✓ Debería devolver 0 si no hay registros de basura desechara hoy
36     ✓ Debería devolver un error 500 si ocurre un problema en el proceso
37
38 Area Routes
39   POST /api/areas
40     ✓ Debería crear un área correctamente
41     ✓ Debería devolver un error 400 si faltan campos requeridos
42     ✓ Debería devolver un error 500 si ocurre un problema en el servidor
43   GET /api/areas/:id
44     ✓ Debería devolver un área por su ID
```

Figura 6.7: Tests realizados al API.

```

    ✓ Debería devolver un error si el Waste no existe
309   ✓ Debería devolver un error si ocurre un problema con la base de datos
310 deleteWaste
311   ✓ Debería eliminar un Waste y registrar auditoría
312   ✓ Debería devolver un error si el Waste no existe
313   ✓ Debería devolver un error si ocurre un problema con la base de datos
314
315 WasteType Controller
316 createWasteType
317   ✓ Debería crear un nuevo WasteType y registrar auditoría
318   ✓ Debería devolver un error si los datos de entrada son inválidos
319   ✓ Debería devolver un error si ocurre un problema con la base de datos
320 getWasteType
321   ✓ Debería devolver un WasteType por su ID
322   ✓ Debería devolver un error si el WasteType no existe
323   ✓ Debería devolver un error si ocurre un problema con la base de datos
324 updateWasteType
325   ✓ Debería actualizar un WasteType existente y registrar auditoría
326   ✓ Debería devolver un error si el WasteType no existe
327   ✓ Debería devolver un error si ocurre un problema con la base de datos
328 deleteWasteType
329   ✓ Debería eliminar un WasteType por su ID y registrar auditoría
330   ✓ Debería devolver un error si el WasteType no existe
331   ✓ Debería devolver un error si ocurre un problema con la base de datos
332
333
334 220 passing (551ms)
335

```

Figura 6.8: Resultado de los tests acertados.

### 6.2.3. Pruebas de Carga y Rendimiento

Durante la fase de pruebas, se llevaron a cabo evaluaciones de carga y rendimiento para medir la capacidad de respuesta del API ante diferentes volúmenes de solicitudes. Estos escenarios permitieron observar el comportamiento y los tiempos de respuesta del sistema bajo distintas condiciones de carga.

Es importante destacar que las pruebas de rendimiento se enfocaron especialmente en las consultas de análisis de datos, ya que estas suelen presentar tiempos de respuesta más elevados debido a la necesidad de realizar búsquedas complejas en múltiples tablas. Estas consultas fueron seleccionadas por ser las más representativas del uso intensivo del sistema y por su relevancia en la evaluación de la capacidad del API para gestionar grandes volúmenes de datos.

En base a los cálculos de la metodología, se determinó que un 20 % de la población de la Ciudad de Guatemala generaría aproximadamente 14 solicitudes por segundo. Se realizó una prueba con una carga mayor, alcanzando 20 solicitudes por segundo, a continuación se presentan los resultados de la prueba de rendimiento durante 1 minuto:

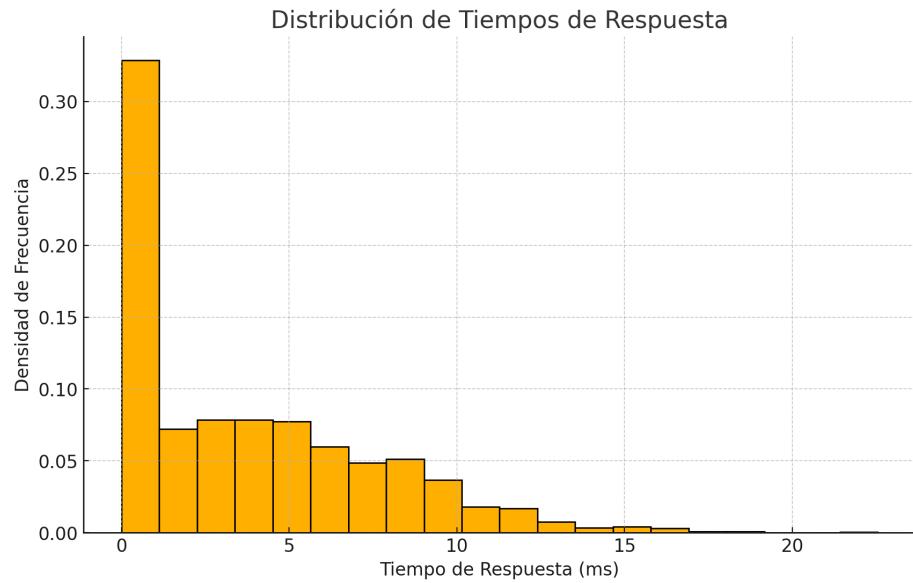


Figura 6.9: Resultados prueba de carga del 20 %.

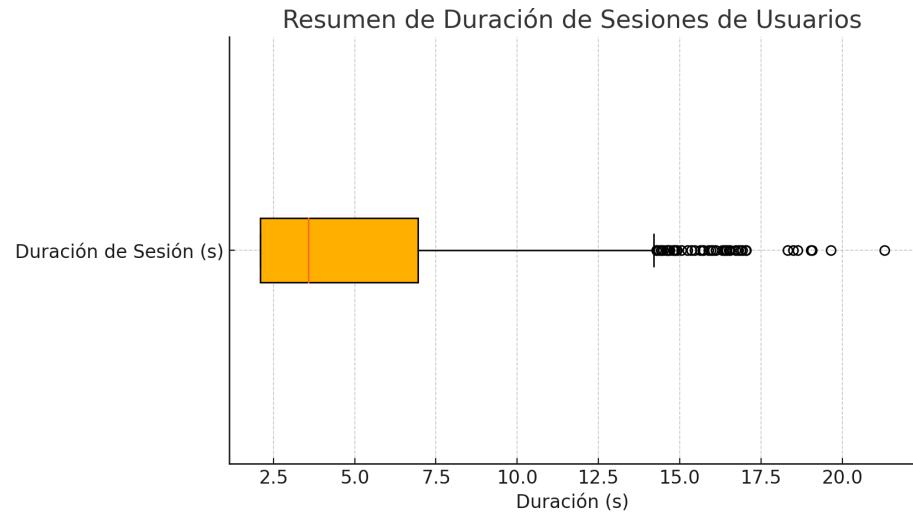


Figura 6.10: Resultados prueba de carga del 20 %.

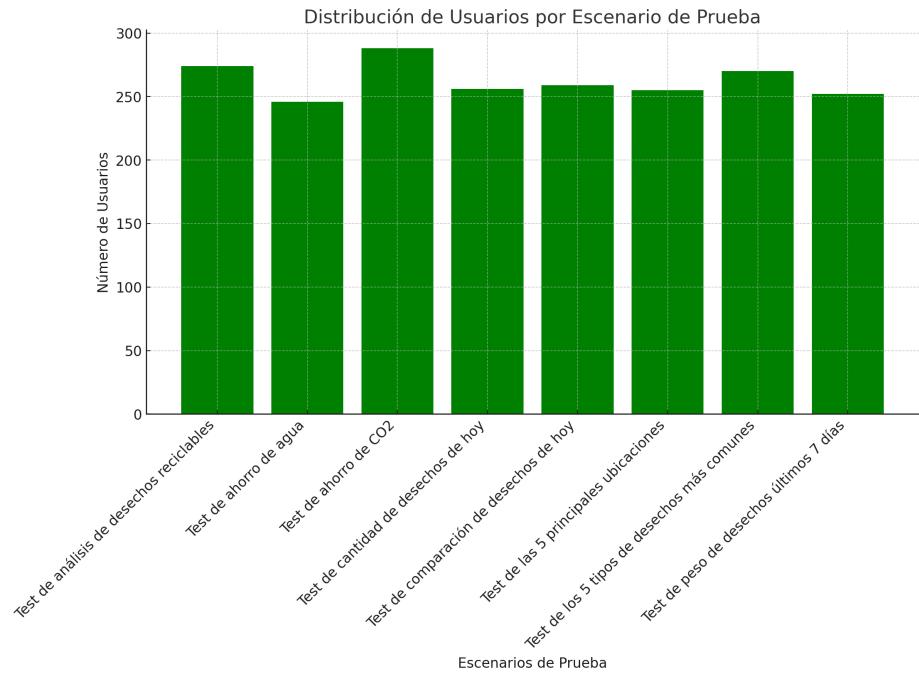


Figura 6.11: Resultados prueba de carga del 20 %.

```
▼ ⓘ Run Performance Tests
317 Summary report @ 06:19:44(+0000)
318 -----
319
320 http.codes.200: ..... 1200
321 http.downloaded_bytes: ..... 52559
322 http.request_rate: ..... 20/sec
323 http.requests: ..... 1200
324 http.response_time:
325   min: ..... 0
326   max: ..... 48
327   mean: ..... 3.3
328   median: ..... 3
329   p95: ..... 5
330   p99: ..... 12.1
331 http.response_time.2xx:
332   min: ..... 0
333   max: ..... 48
334   mean: ..... 3.3
335   median: ..... 3
336   p95: ..... 5
337   p99: ..... 12.1
338 http.responses: ..... 1200
339 vusers.completed: ..... 1200
340 vusers.created: ..... 1200
341 vusers.created_by_name.Test de ahorro de CO2: ..... 156
342 vusers.created_by_name.Test de ahorro de agua: ..... 170
343 vusers.created_by_name.Test de análisis de desechos reciclables: ..... 155
344 vusers.created_by_name.Test de cantidad de desechos de hoy: ..... 160
345 vusers.created_by_name.Test de comparación de desechos de hoy con el promedi... 136
346 vusers.created_by_name.Test de las 5 principales ubicaciones por desechos: ..... 136
347 vusers.created_by_name.Test de los 5 tipos de desechos más comunes: ..... 138
348 vusers.created_by_name.Test de peso de desechos de los últimos 7 días: ..... 149
349 vusers.failed: ..... 0
350 vusers.session_length:
351   min: ..... 2.9
352   max: ..... 52.1
353   mean: ..... 4.7
354   median: ..... 4.2
355   p95: ..... 7
356   p99: ..... 15
```

Figura 6.12: Resultados prueba de carga del 20 %.

Además, se calculó que para un 50 % de la población de la Ciudad de Guatemala, el API debería soportar alrededor de 35 solicitudes por segundo. Los resultados de esta prueba también se analizaron para comprender el rendimiento del API bajo una carga considerable. Se presentan a continuación los resultados de la prueba de rendimiento durante 1 minuto:

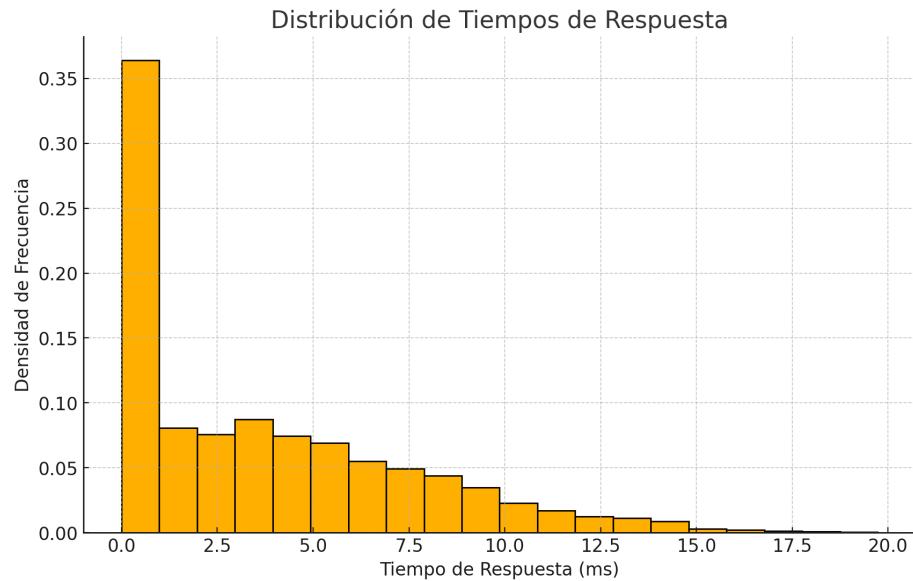


Figura 6.13: Resultados prueba de carga del 50 %.

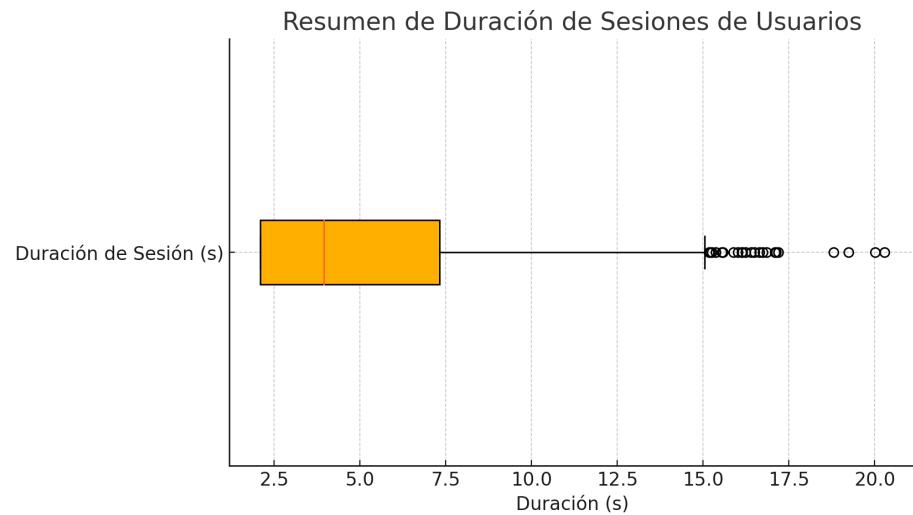


Figura 6.14: Resultados prueba de carga del 50 %.

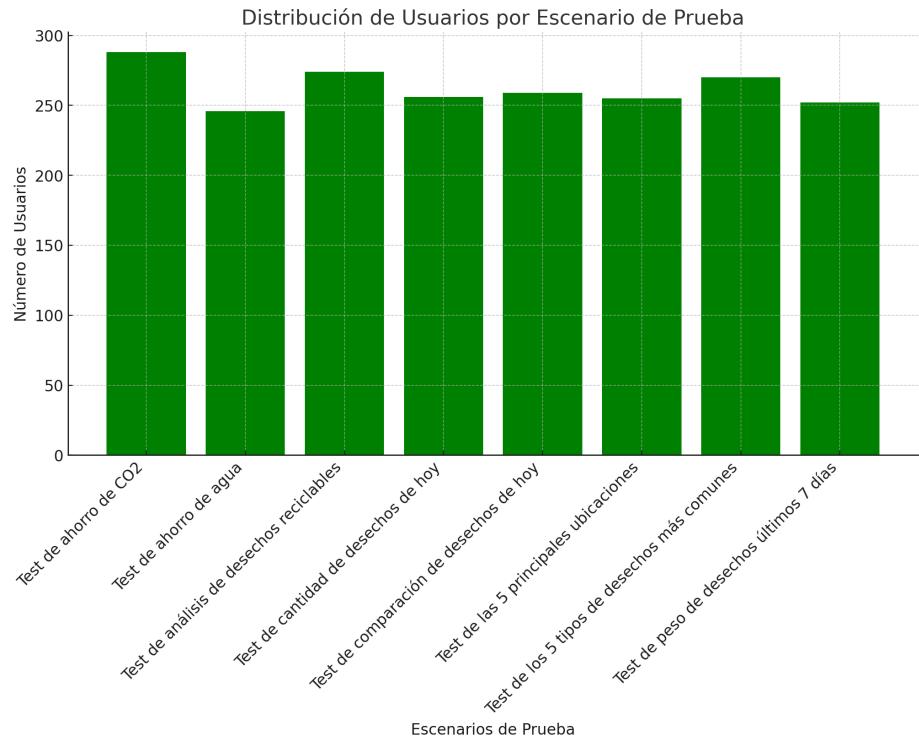


Figura 6.15: Resultados prueba de carga del 50 %.

```
✓ Run Performance Tests
315 Summary report @ 06:13:21(+0000)
316 -----
317
318 http.codes.200: ..... 2100
319 http.downloaded_bytes: ..... 94722
320 http.request_rate: ..... 31/sec
321 http.requests: ..... 2100
322 http.response_time:
323     min: ..... 0
324     max: ..... 93
325     mean: ..... 2.8
326     median: ..... 2
327     p95: ..... 6
328     p99: ..... 25.8
329 http.response_time.2xx:
330     min: ..... 0
331     max: ..... 93
332     mean: ..... 2.8
333     median: ..... 2
334     p95: ..... 6
335     p99: ..... 25.8
336 http.responses: ..... 2100
337 vusers.completed: ..... 2100
338 vusers.created: ..... 2100
339 vusers.created_by_name.Test de ahorro de CO2: ..... 288
340 vusers.created_by_name.Test de ahorro de agua: ..... 246
341 vusers.created_by_name.Test de análisis de desechos reciclables: ..... 274
342 vusers.created_by_name.Test de cantidad de desechos de hoy: ..... 256
343 vusers.created_by_name.Test de comparación de desechos de hoy con el promedi... 259
344 vusers.created_by_name.Test de las 5 principales ubicaciones por desechos: ..... 255
345 vusers.created_by_name.Test de los 5 tipos de desechos más comunes: ..... 270
346 vusers.created_by_name.Test de peso de desechos de los últimos 7 días: ..... 252
347 vusers.failed: ..... 0
348 vusers.session_length:
349     min: ..... 2.1
350     max: ..... 96.4
351     mean: ..... 3.9
352     median: ..... 2.9
353     p95: ..... 7.3
354     p99: ..... 27.4
```

Figura 6.16: Resultados prueba de carga del 50 %.

Por último, se llevó a cabo una prueba de carga para evaluar los tiempos de respuesta del API, simulando un escenario en el que el 100 % de la población de la Ciudad de Guatemala interactúa con el sistema. De acuerdo con los cálculos, esto representa aproximadamente 70 solicitudes por segundo. Los resultados de la prueba de rendimiento durante 1 minuto se presentan a continuación:

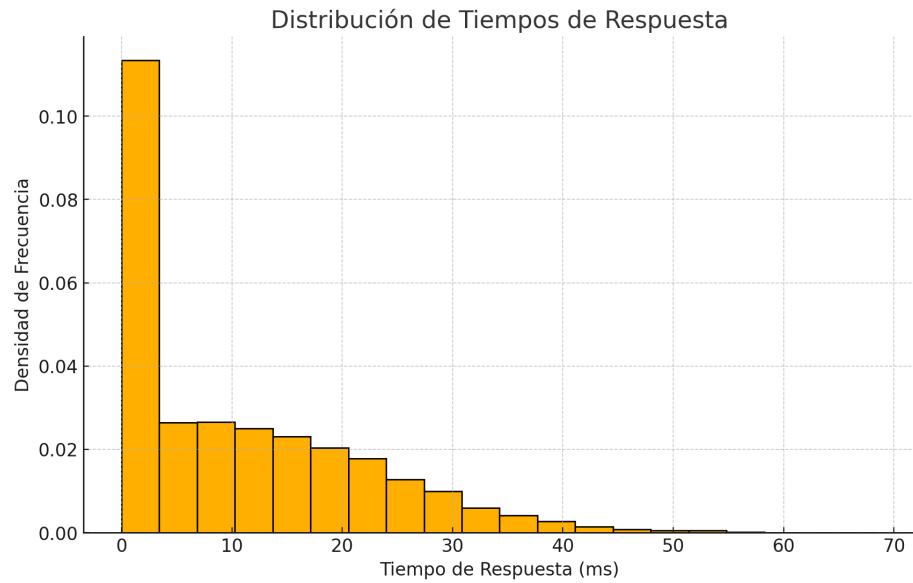


Figura 6.17: Resultados de la prueba de carga para el 100 % de la población.

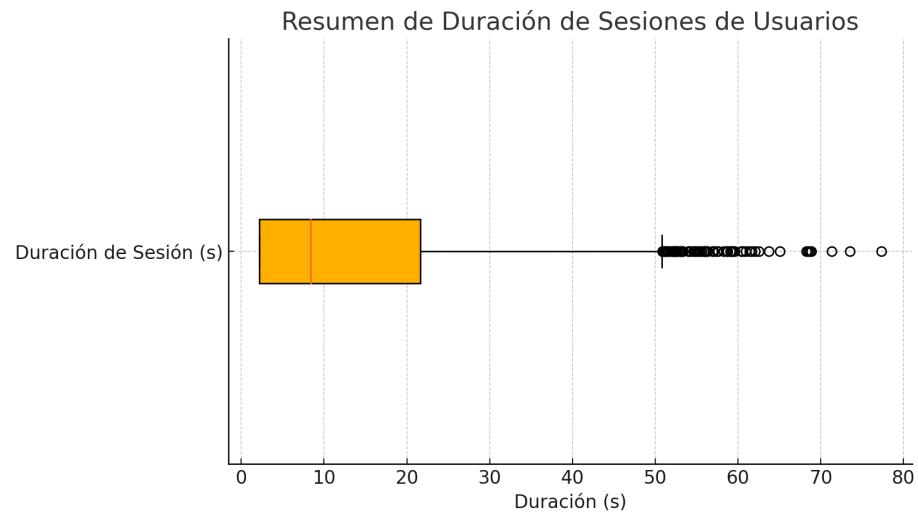


Figura 6.18: Resultados de la prueba de carga para el 100 % de la población.

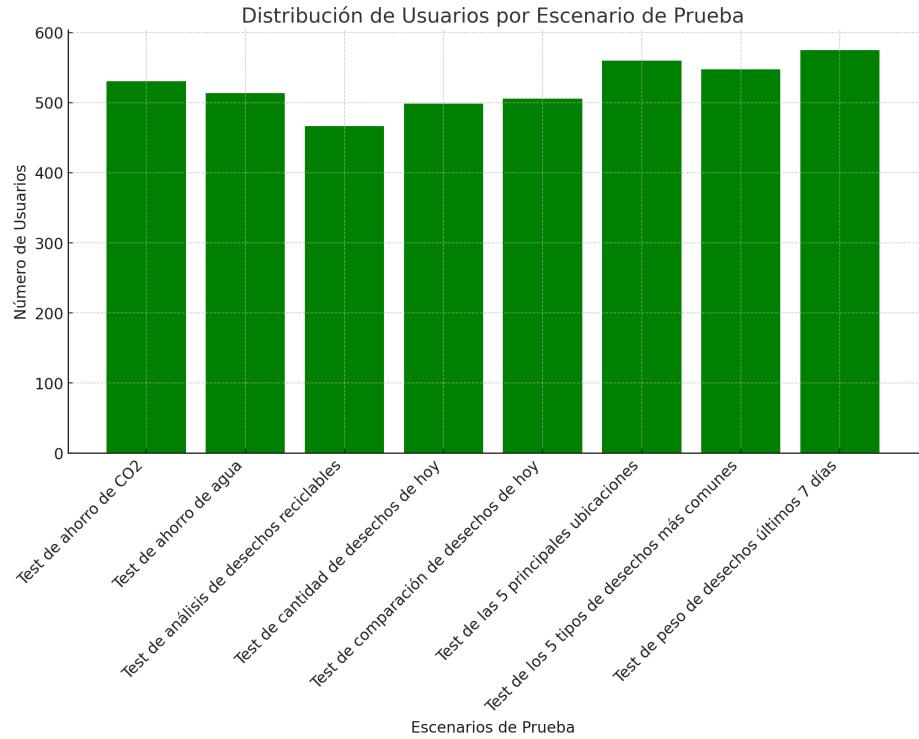


Figura 6.19: Resultados de la prueba de carga para el 100 % de la población.

```

310
317 Summary report @ 06:53:10(+0000)
318 -----
319
320 http.codes.200: ..... 4200
321 http.downloaded_bytes: ..... 190922
322 http.request_rate: ..... 70/sec
323 http.requests: ..... 4200
324 http.response_time:
325   min: ..... 0
326   max: ..... 189
327   mean: ..... 7.5
328   median: ..... 3
329   p95: ..... 30.9
330   p99: ..... 117.9
331 http.response_time.2xx:
332   min: ..... 0
333   max: ..... 189
334   mean: ..... 7.5
335   median: ..... 3
336   p95: ..... 30.9
337   p99: ..... 117.9
338 http.responses: ..... 4200
339 vusers.completed: ..... 4200
340 vusers.created: ..... 4200
341 vusers.created_by_name.Test de ahorro de CO2: ..... 531
342 vusers.created_by_name.Test de ahorro de agua: ..... 514
343 vusers.created_by_name.Test de análisis de desechos reciclables: ..... 467
344 vusers.created_by_name.Test de cantidad de desechos de hoy: ..... 499
345 vusers.created_by_name.Test de comparación de desechos de hoy con el promedi... 506
346 vusers.created_by_name.Test de las 5 principales ubicaciones por desechos: ..... 560
347 vusers.created_by_name.Test de los 5 tipos de desechos más comunes: ..... 548
348 vusers.created_by_name.Test de peso de desechos de los últimos 7 días: ..... 575
349 vusers.failed: ..... 0
350 vusers.session_length:
351   min: ..... 2.2
352   max: ..... 192.1
353   mean: ..... 8.7
354   median: ..... 3.4
355   p95: ..... 32.8
356   p99: ..... 120.3

```

Figura 6.20: Resumen de resultados de la prueba de carga para el 100 %.

Haciendo un resumen de los resultados se tiene lo siguiente:

Métrica	20 solicitudes/seg	35 solicitudes/seg	70 solicitudes/seg
Total de solicitudes	1200	2100	4200
Tiempo de respuesta (promedio)	3.5 ms	2.8 ms	7.5 ms
Tiempo de respuesta (mediana)	3 ms	2 ms	3 ms
Tiempo de respuesta (p95)	5 ms	6 ms	30.9 ms
Tiempo de respuesta (p99)	8.9 ms	25.8 ms	117.9 ms
Duración de sesión (promedio)	4.8 s	3.9 s	8.7 s
Duración de sesión (p95)	7.5 s	7.3 s	32.8 s
Duración de sesión (p99)	11.1 s	27.4 s	120.3 s

Tabla 6.1: Resultados de pruebas de carga a diferentes tasas de solicitudes

Además, en el Anexo A, titulado *Pruebas de carga y rendimiento en API de producción*, se presentan los resultados obtenidos al realizar múltiples solicitudes a la API. En este anexo, se detallan los resultados de las pruebas realizadas para una carga de 35 solicitudes por segundo A.1 y 70 solicitudes por segundo A.8.

Métrica	20 solicitudes/seg	35 solicitudes/seg	70 solicitudes/seg
Total de solicitudes	1200	2100	4200
Solicitudes exitosas (código 200)	1200	2096	4194
Errores (ETIMEDOUT)	0	4	6
Tiempo de respuesta (promedio)	3.5 ms	78.4 ms	4211.6 ms
Tiempo de respuesta (mediana)	3 ms	71.5 ms	4583.6 ms
Tiempo de respuesta (p95)	5 ms	120.3 ms	5168 ms
Tiempo de respuesta (p99)	8.9 ms	159.2 ms	6976.1 ms
Duración de sesión (promedio)	4.8 s	145 ms	4298 ms
Duración de sesión (mediana)	4.3 s	135.7 ms	4676.2 ms
Duración de sesión (p95)	7.5 s	202.4 ms	5378.9 ms
Duración de sesión (p99)	11.1 s	257.3 ms	6976.1 ms

Tabla 6.2: Comparación de rendimiento en tres escenarios de carga en API de producción

### 6.3. Análisis de Datos

La implementación de funciones para el análisis de datos de usuarios específicos y el análisis general permite aprovechar la información generada en los registros de desechos para obtener resultados significativos. Este análisis facilita la evaluación de los hábitos de los usuarios, el impacto ambiental de sus acciones, y su contribución en términos de reducción de residuos y reciclaje.

Estas funciones analíticas están diseñadas para ofrecer a cada usuario un resumen de su comportamiento en la gestión de desechos, así como métricas generales que abarcan todos los usuarios del sistema.

A continuación, se muestran los resultados de algunas solicitudes de análisis realizadas por usuarios individuales:

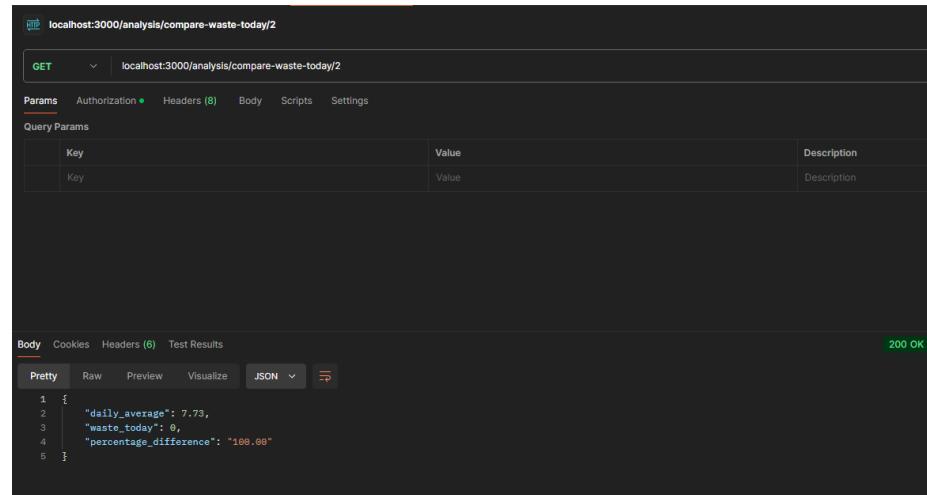
The screenshot shows a Postman interface with the following details:

- URL:** localhost:3000/analysis/top5-waste-types/2
- Method:** GET
- Headers:** Authorization (with a green checkmark), Headers (8), Body, Scripts, Settings
- Query Params:** Key, Value, Description
- Body:** Pretty, Raw, Preview, Visualize, JSON, 200 OK
- JSON Response:**

```

1 [ 
2   { 
3     "waste_type": "ewaste",
4     "total_weight": 7.73
5   }
6 ]
    
```

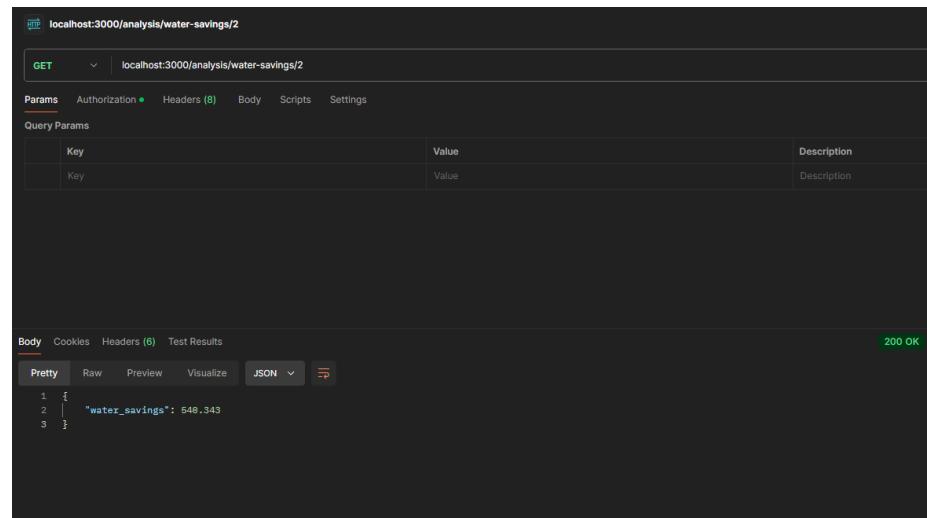
Figura 6.21: Resultado top 5 desechos usuario.



A screenshot of a Postman API client interface. The URL is `localhost:3000/analysis/compare-waste-today/2`. The method is `GET`. The Headers tab shows `Authorization` and `Content-Type`. The Body tab is empty. The JSON response is:

```
1 {  
2   "daily_average": 7.73,  
3   "waste_today": 0,  
4   "percentage_difference": "100.00"  
5 }
```

Figura 6.22: Resultado porcentaje comparación usuario.



A screenshot of a Postman API client interface. The URL is `localhost:3000/analysis/water-savings/2`. The method is `GET`. The Headers tab shows `Authorization` and `Content-Type`. The Body tab is empty. The JSON response is:

```
1 {  
2   "water_savings": 540.343  
3 }
```

Figura 6.23: Resultado porcentaje comparación usuario.

A continuación, se muestran algunos ejemplos de visualizaciones generadas en Power BI a partir de solicitudes al API:

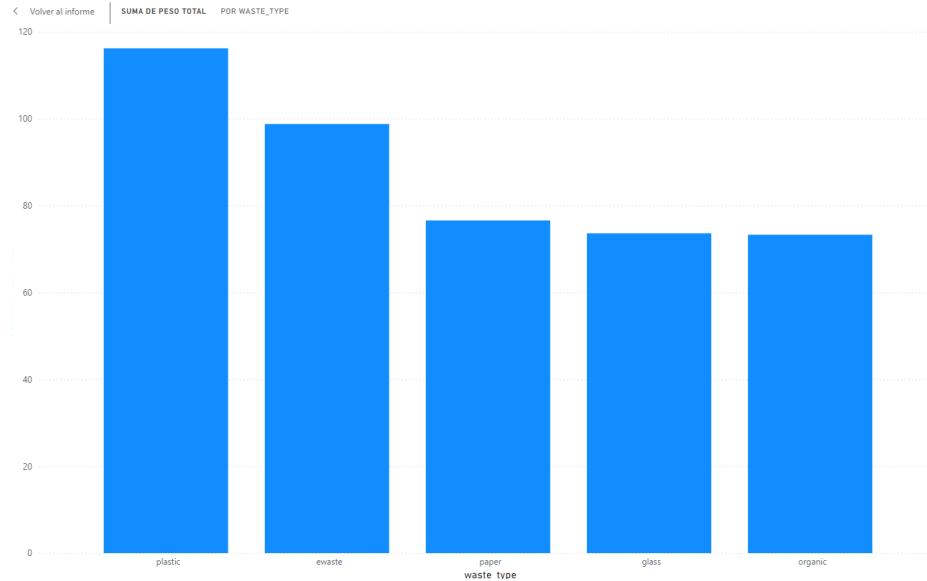


Figura 6.24: Resultado top 5 desechos generales.

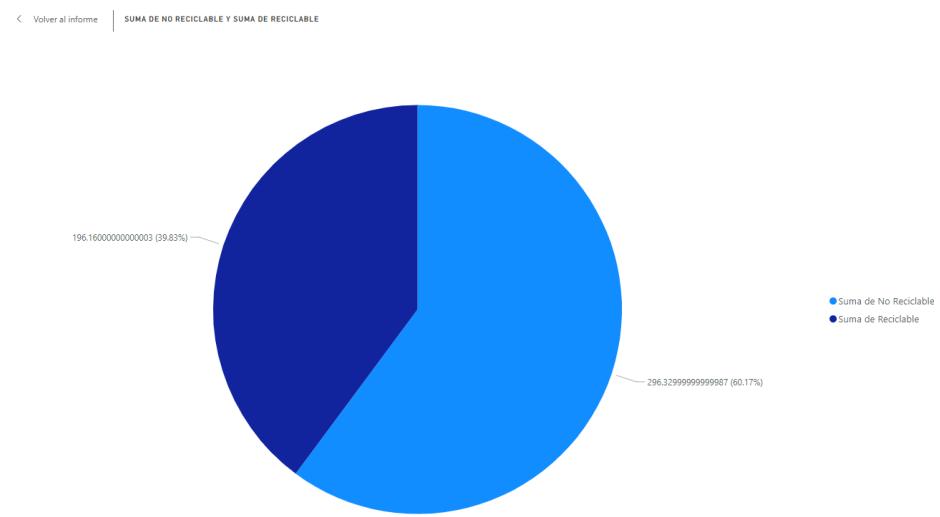


Figura 6.25: Resultado desechos reciclables generales.

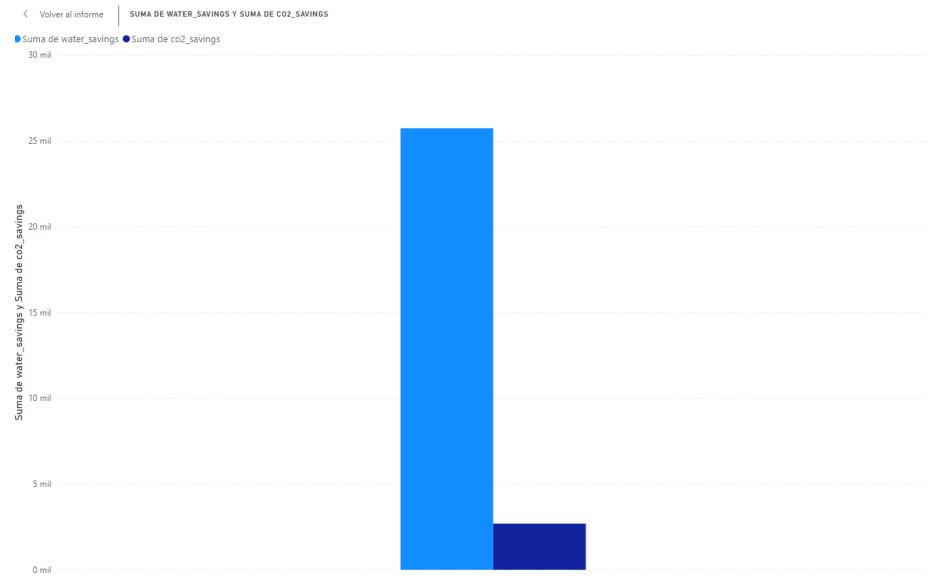


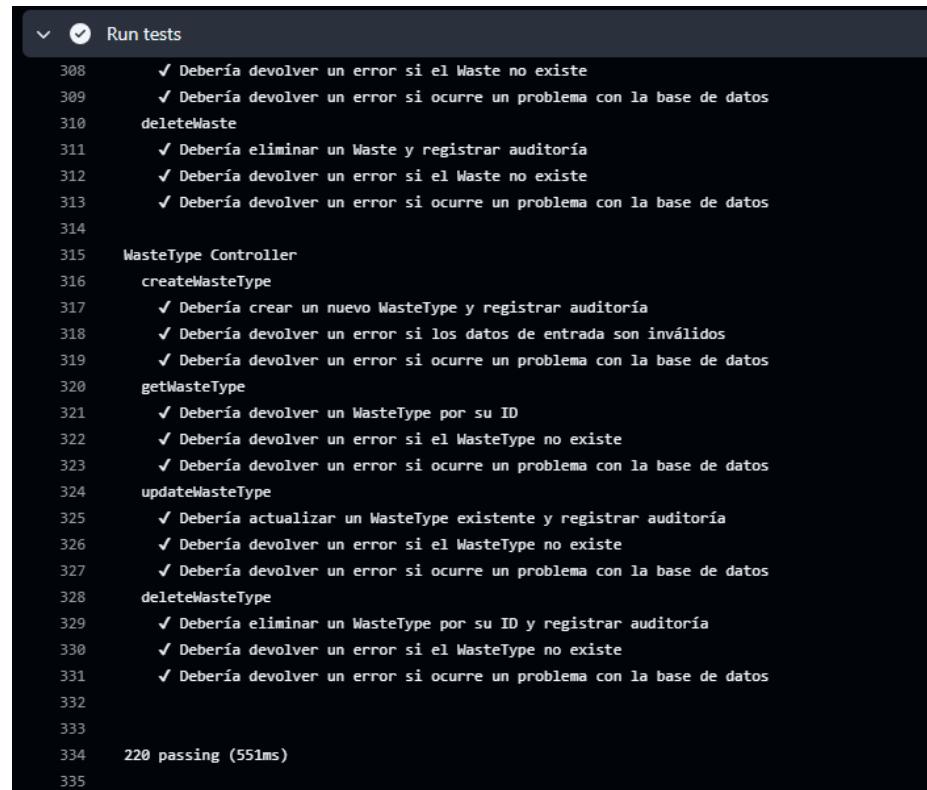
Figura 6.26: Resultado CO2 y Agua generales.

## 6.4. Implementación de CI/CD

Como parte final del módulo y un componente crucial en el proceso de desarrollo, se presentan los resultados obtenidos a partir de la implementación del *pipeline* de CI/CD. Este *pipeline* fue diseñado para automatizar las tareas de integración y despliegue, asegurando que cada cambio en el código sea verificado y probado antes de ser incorporado en el sistema de producción. Su implementación no solo mejora la eficiencia del proceso de desarrollo, sino que también reduce el riesgo de errores y asegura la estabilidad del API en cada actualización.

### 6.4.1. Pruebas

Durante el proceso de pruebas, se evaluaron distintos casos que permitieron comprobar cómo el sistema responde ante situaciones comunes que pueden surgir al realizar modificaciones en el código. Estas pruebas ayudan a validar la robustez del API, identificando y corrigiendo problemas antes de que impacten a los usuarios finales. A continuación, se presentan los resultados de los dos escenarios principales, demostrando cómo el *pipeline* gestiona los cambios y asegura un despliegue confiable:



```
Run tests

308     ✓ Debería devolver un error si el Waste no existe
309     ✓ Debería devolver un error si ocurre un problema con la base de datos
310 deleteWaste
311     ✓ Debería eliminar un Waste y registrar auditoría
312     ✓ Debería devolver un error si el Waste no existe
313     ✓ Debería devolver un error si ocurre un problema con la base de datos
314
315 WasteType Controller
316 createWasteType
317     ✓ Debería crear un nuevo WasteType y registrar auditoría
318     ✓ Debería devolver un error si los datos de entrada son inválidos
319     ✓ Debería devolver un error si ocurre un problema con la base de datos
320 getWasteType
321     ✓ Debería devolver un WasteType por su ID
322     ✓ Debería devolver un error si el WasteType no existe
323     ✓ Debería devolver un error si ocurre un problema con la base de datos
324 updateWasteType
325     ✓ Debería actualizar un WasteType existente y registrar auditoría
326     ✓ Debería devolver un error si el WasteType no existe
327     ✓ Debería devolver un error si ocurre un problema con la base de datos
328 deleteWasteType
329     ✓ Debería eliminar un WasteType por su ID y registrar auditoría
330     ✓ Debería devolver un error si el WasteType no existe
331     ✓ Debería devolver un error si ocurre un problema con la base de datos
332
333
334 220 passing (551ms)
335
```

Figura 6.27: Resultado de los tests acertados.

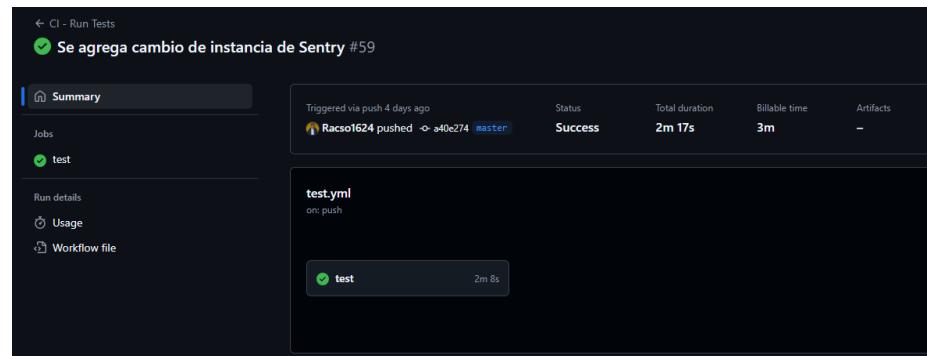
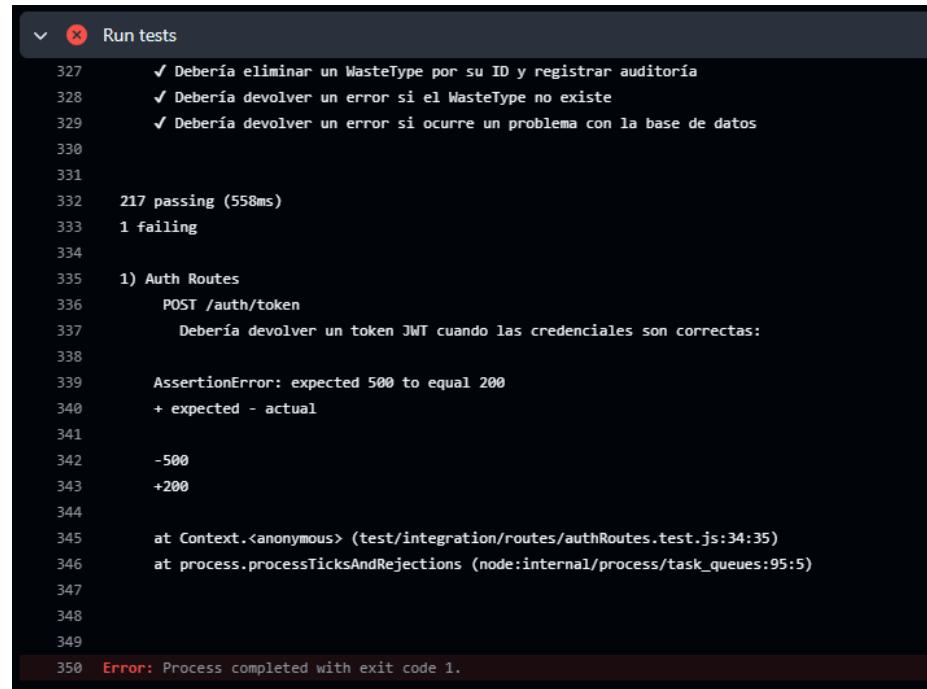


Figura 6.28: Resultado del action de los tests acertados.



The screenshot shows a terminal window titled "Run tests". The output of the tests is displayed, showing 217 passing tests (558ms) and 1 failing test. The failing test is related to an "Auth Routes" POST /auth/token endpoint, where an Assertion Error occurred: expected 500 to equal 200. The error message includes the expected value (-500) and the actual value (+200). The stack trace indicates the error happened at Context.<anonymous> (test/integration/routes/authRoutes.test.js:34:35) and at process.processTicksAndRejections (node:internal/process/task\_queues:95:5). The final message at the bottom of the terminal window is "Error: Process completed with exit code 1."

Figura 6.29: Resultado de los tests fallados.

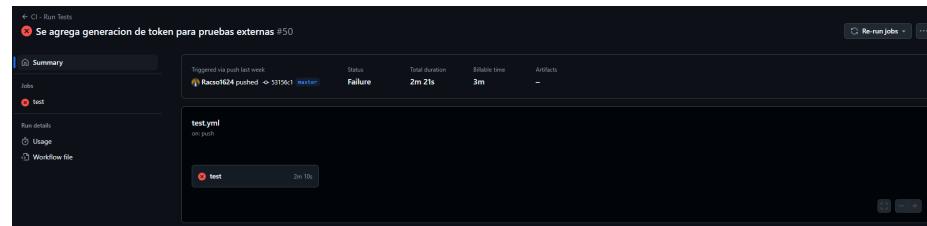


Figura 6.30: Resultado del action de los tests fallados.

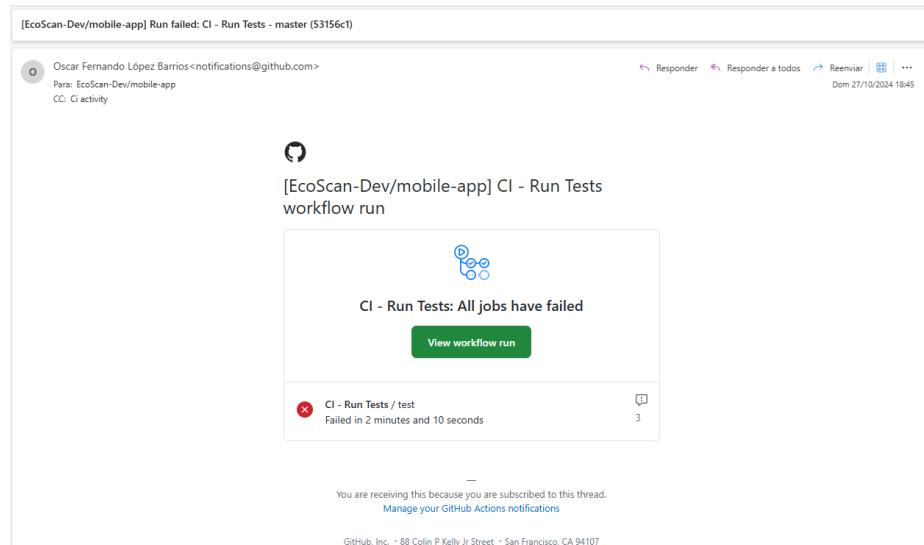


Figura 6.31: Resultado de los tests fallados en el mail.

#### 6.4.2. Monitoreo de Errores con Sentry

La elección de una herramienta de monitoreo de errores es fundamental, ya que permite capturar datos precisos para identificar y resolver problemas que pueden haber pasado desapercibidos una vez que el API se encuentra en producción.

Además, **Sentry** no solo proporciona monitoreo de errores, sino que también ofrece una amplia gama de datos adicionales. En este caso, Sentry permite obtener trazas detalladas de ejecución y monitorear el rendimiento de las consultas (*queries*), proporcionando una visión integral del funcionamiento del sistema. Esto facilita la detección de cuellos de botella y problemas de rendimiento, lo cual es esencial para mantener la eficiencia y confiabilidad del API en entornos de producción.

A continuación, se presentan los resultados de los escenarios en los que **Sentry** ha sido utilizado para el monitoreo y diagnóstico de errores en el sistema:

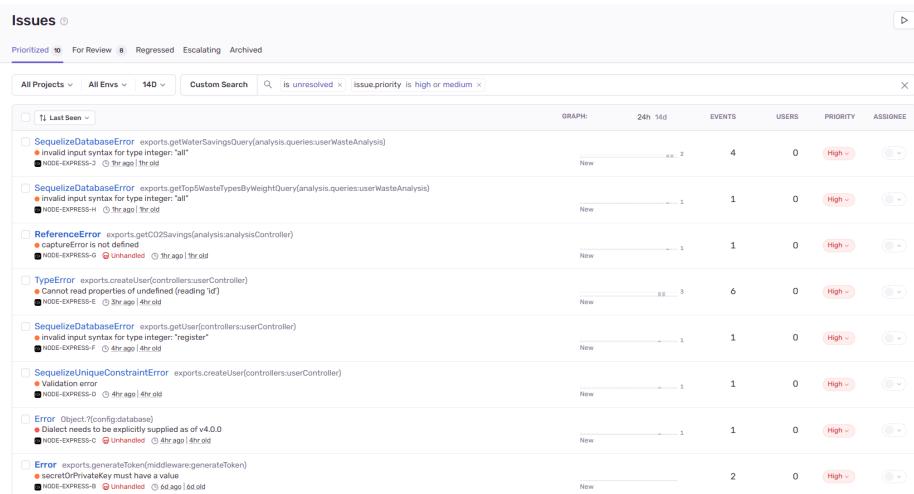


Figura 6.32: Resultado de los errores de Sentry.

The screenshot shows the Sentry interface for an issue titled "SequelizeDatabaseError exports.getTop5WasteTypesByWeightQuery(analysis.queries:userWasteAnalysis...)" with the error message "invalid input syntax for type integer: "all"".

Event ID: 6ef94303 Nov 3, 1:56 AM

Event Highlights:

handled	yes	transaction	--
level	error	status_code	--
release	--	Trace: Trace ID	ae4fd3daca34eebb13a21f4d9b51cd2
environment	production	Runtime: Name	node
url	--	Runtime: Version	v20.14.0

Figura 6.33: Resultado de la descripcion de un error en Sentry.

The screenshot shows a Sentry notification email for a new issue titled "NODE-EXPRESS-H - SequelizeDatabaseError: invalid input syntax for type integer: "all"".

**New issue**  
We notified recently active members in the node-express project of this issue

**ISSUE**  
**SequelizeDatabaseError**  
invalid input syntax for type integer: "all"

Nov 3, 2024, 1:56:55 a.m. CST ID: 6ef94303016d4a65a6377be844ff83c0

**Project**: node-express  
**environment**: production  
**Level**: error

**Exception**

```
SequelizeDatabaseError: invalid input syntax for type integer: "all"
  File "D:\UNIVERSIDAD\PROYECTO-GRADUACION\mobile-app\eco-scan-backend\analysis\queries\userWasteAnalysis.js", line 106, in
  exports.getTop5WasteTypesByWeightQuery
    const result = await sequelize.query(query, {
      File "D:\UNIVERSIDAD\PROYECTO-GRADUACION\mobile-app\eco-scan-backend\analysis\analysisController.js", line 112, in
  exports.getTop5WasteTypes
    const result = await getTop5WasteTypesByWeightQuery(sequelize,
      userId);
```

Figura 6.34: Resultado una notificacion de error.

TRANSACTION	PROJECT	TPM ↓	P50	P95	FAILURE RATE	APDEX	USERS	USER MISERY
GET /analysis/co2-savings/:userId	node-express	0.199/min	2.46ms	55.46ms	2.9%	0.995	0 ▲	(no value)
GET /analysis/waste-last7days/:userId	node-express	0.197/min	2.41ms	54.98ms	3.5%	0.996	0 ▲	(no value)
GET /analysis/top5-waste-types/:userId	node-express	0.197/min	2.50ms	27.74ms	3.35%	0.995	0 ▲	(no value)
GET /analysis/compare-waste-today/:userId	node-express	0.196/min	3.28ms	47.49ms	2.89%	0.995	0 ▲	(no value)
GET /analysis/waste-today/:userId	node-express	0.195/min	2.39ms	36.88ms	3.11%	0.995	0 ▲	(no value)
GET /analysis/top5-locations/:userId	node-express	0.194/min	2.57ms	54.89ms	3.42%	0.994	0 ▲	(no value)
GET /analysis/water-savings/:userId	node-express	0.193/min	2.54ms	54.39ms	3.54%	0.996	0 ▲	(no value)
GET /analysis/recyclable-waste/:userId	node-express	0.193/min	2.48ms	44.87ms	3.16%	0.996	0 ▲	(no value)
POST /	node-express	0.0298/min	3.30ms	6.15ms	100%	0.998	0 ▲	(no value)

Figura 6.35: Resultado del apartado de performance de Sentry.

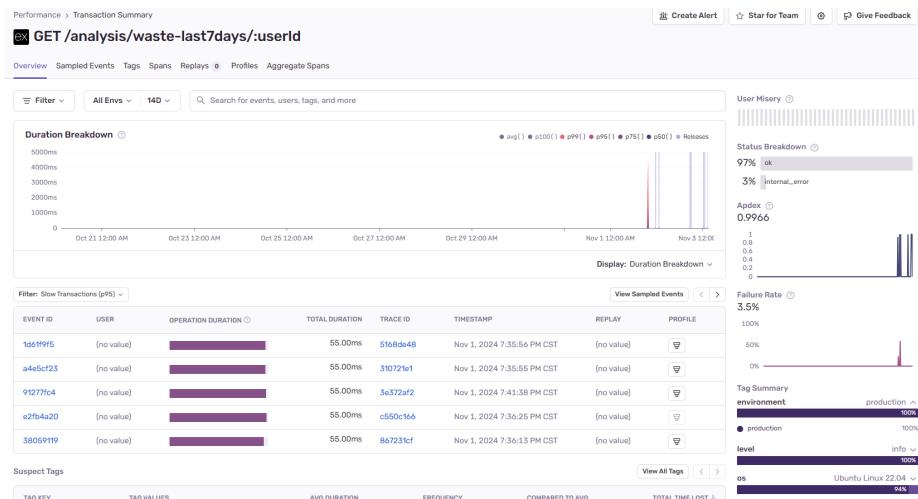


Figura 6.36: Resultado del detalle de descripción de performance de Sentry.

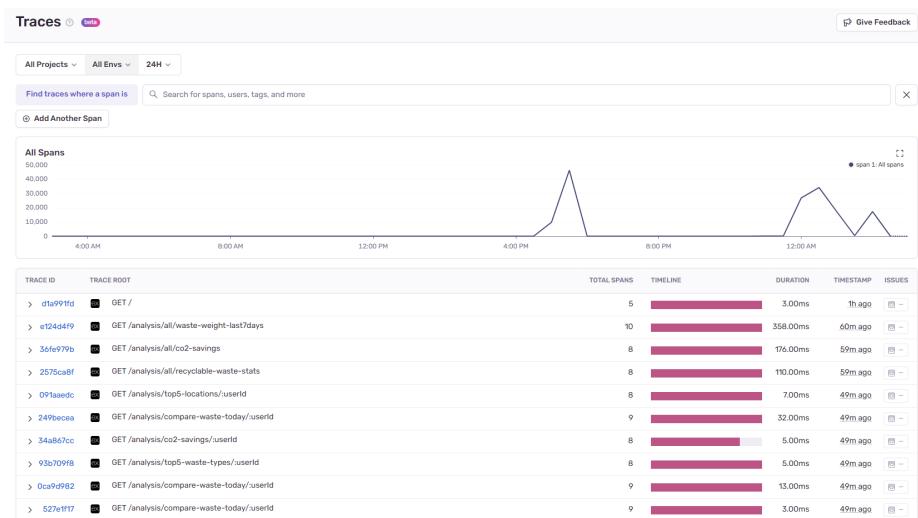


Figura 6.37: Resultado de los traces de Sentry.

#### 6.4.3. Automatización con GitHub Actions

La última funcionalidad necesaria para establecer una base sólida de CI/CD es la automatización de tareas. En este caso, se logró implementar exitosamente mediante **GitHub Actions**. Esta herramienta permite definir flujos de trabajo automatizados que verifican la calidad del código y facilitan el despliegue continuo. Con GitHub Actions, es posible configurar una serie de acciones para evaluar el código en cada cambio realizado, garantizando así un control de calidad constante y permitiendo una entrega continua y confiable del software.

A continuación se muestra el funcionamiento:

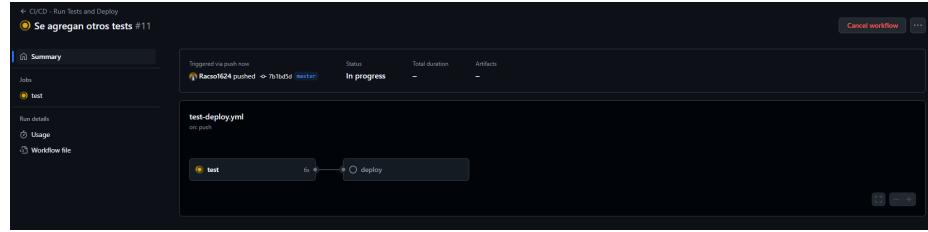


Figura 6.38: Resultado de GitHub Actions.

Resultado del deploy:



Figura 6.39: Resultado del deploy.

# CAPÍTULO 7

---

## Análisis de Resultados

---

Una vez obtenidos los resultados, es fundamental analizarlos para evaluar si cumplen con el enfoque y los objetivos planteados inicialmente. En los siguientes apartados se examinará cada una de las implementaciones y su contribución a la eficiencia, robustez y calidad del módulo final.

### 7.1. Elección de las Tecnologías

Las tecnologías seleccionadas para el proyecto han demostrado ser adecuadas, proporcionando una integración fluida y efectiva entre sí. La elección de estas tecnologías fue acertada, ya que cada una facilita la operabilidad y permite la incorporación de herramientas externas que fortalecen y consolidan una base sólida para el módulo. Esta combinación de tecnologías no solo garantiza la estabilidad y eficiencia del sistema, sino que también facilita futuras expansiones y mejoras.

### 7.2. Diseño y Estructura de la Base de Datos

El diseño final de la base de datos permite gestionar de manera eficiente la información de los usuarios. A lo largo de las iteraciones, se implementaron mejoras y ajustes específicos que dieron lugar a un modelo optimizado para almacenar los datos necesarios para el análisis. Este proceso permitió evaluar y ajustar cada aspecto de la estructura de la base de datos, asegurando que solo se almacene la información relevante y optimizando el rendimiento general del sistema.

La incorporación de una tabla de auditoría o bitácora fortalece la integridad de la información al permitir el monitoreo de todos los cambios realizados en los datos. Esto crea una base sólida para futuras expansiones de la aplicación, facilitando el manejo de roles y permisos para usuarios y administradores, y asegurando un control detallado de las operaciones dentro del sistema.

### 7.3. API

La versión final del API incorpora funcionalidades clave para el manejo seguro y eficiente de la información. Entre estas, destaca la integración de autenticación en las solicitudes mediante **JWT** (JSON Web Token), lo cual asegura que solo usuarios autorizados puedan acceder y manipular los datos, protegiendo la integridad de la información desde su uso en una aplicación hasta su almacenamiento en la base de datos.

Las pruebas de funcionalidad han verificado que estas restricciones de autenticación operan correctamente, permitiendo a los usuarios acceder mediante la aplicación sin exponer la información a personas no autorizadas.

Las funciones implementadas en cada modelo permiten el ingreso adecuado de los datos, asegurando el almacenamiento preciso de la información relacionada con los usuarios, ubicaciones y registros de desechos. Además, el registro de actividades en la bitácora de auditoría proporciona un historial detallado de las operaciones realizadas, incluyendo las acciones de agregación, actualización y eliminación, junto con la información de los usuarios y las fechas correspondientes. Esto garantiza un seguimiento completo y confiable de las actividades en el sistema.

Las funciones de análisis de datos permiten calcular con precisión la información relacionada con los usuarios en cuanto a sus registros de gestión de desechos. Estas funciones proporcionan una visión detallada del comportamiento y patrones de uso, facilitando el análisis de datos clave para evaluar el impacto de la gestión de residuos y mejorar la toma de decisiones.

Mediante una fase de pruebas exhaustiva del API final, se verificó que todas las funcionalidades fueron implementadas correctamente y funcionan conforme a lo esperado. Para alcanzar esta conclusión, se realizaron tres tipos de pruebas que confirman el correcto funcionamiento del sistema.

Las pruebas realizadas incluyen pruebas unitarias, las cuales validaron cada uno de los controladores, verificando que proporcionen la información de manera precisa y eficiente. Además, se llevaron a cabo pruebas de integración, que demostraron la coordinación adecuada entre las rutas y los controladores al procesar las solicitudes del API, garantizando una integración fluida y coherente.

Finalmente, se realizaron pruebas de rendimiento y carga, donde, a través de múltiples iteraciones, se evaluó la capacidad de respuesta del sistema ante diferentes niveles de demanda. Estas pruebas abarcaron tres escenarios específicos, simulando el 20 %, 50 % y 100 % de la carga esperada, y en todos los casos el sistema mostró un rendimiento robusto. De acuerdo con las recomendaciones de [42], se verificó que el tiempo de respuesta de las solicitudes se mantuviera por debajo de los 10 segundos, asegurando así una experiencia fluida para el usuario y minimizando la probabilidad de pérdida de atención.

### 7.4. Análisis de Datos

El apartado de análisis de datos se implementó mediante una estructuración adecuada de la información de los usuarios. Cada una de las funciones diseñadas para el análisis fue construida de manera precisa, con el apoyo directo de la Unidad de Reciclaje de la Municipalidad de Guatemala.

Gracias a la orientación de la Unidad de Reciclaje, se lograron identificar las métricas necesarias para proporcionar información relevante al usuario en el contexto de la aplicación. Se definieron funciones que permiten a los usuarios comprender cómo sus prácticas de gestión de residuos pueden contribuir al cuidado del medio ambiente.

Entre las principales funcionalidades, destacan las métricas de reducción de emisiones de CO<sub>2</sub> y de ahorro de agua, que permiten a los usuarios evaluar el impacto positivo de sus acciones en la

reducción de su huella de carbono. Esto es consistente con lo mencionado en [11], donde se indica que una de las maneras más efectivas de reducir la huella de carbono personal es a través de un reciclaje y una gestión de desechos adecuados. Además, la reducción de la huella de carbono se relaciona estrechamente con la disminución de emisiones de CO<sub>2</sub> y el ahorro de agua, promoviendo una gestión ambientalmente responsable.

Por otro lado, además de proporcionar resultados personalizados para cada usuario, se establecieron las bases para realizar análisis de datos a nivel global, considerando a todos los usuarios de la aplicación. Esto permite obtener una visión más amplia y comprender los patrones de comportamiento que pueden incentivar y guiar a los usuarios hacia un impacto ambiental positivo.

Como resultado de este proceso, se crearon tableros de análisis de datos en **Power BI**, lo cual permite a los administradores y responsables de áreas ambientales acceder a información detallada sobre el comportamiento de la población que utiliza la aplicación. Estos tableros ofrecen una visión completa y precisa, facilitando la toma de decisiones informadas y la implementación de estrategias para promover prácticas sostenibles.

## 7.5. Implementación de CI/CD

En el apartado de **CI/CD**, se establecieron bases sólidas para que el proyecto esté preparado para futuras iteraciones y expansiones a largo plazo. Esto se logró mediante la aplicación de buenas prácticas de Ingeniería de Software, que permitieron diseñar un *pipeline* que abarca las fases de desarrollo, pruebas, lanzamiento y monitoreo.

Cada una de las etapas en este *pipeline* automatizado desempeña un rol clave en asegurar la robustez y adecuada gestión del API. Las herramientas implementadas en cada fase permiten detectar y corregir errores de manera temprana, garantizando la calidad y estabilidad del sistema en cada actualización y facilitando un crecimiento continuo y ordenado del proyecto.

La fase de pruebas permite automatizar la verificación del correcto funcionamiento de los controladores y rutas del API, lo cual facilita la detección temprana de errores y alerta al equipo en caso de problemas ocasionados por iteraciones de código incorrectas. Esto asegura una identificación de problemas generales y un manejo adecuado de errores antes de realizar el despliegue en producción.

Además, se implementaron pruebas de carga y rendimiento automatizadas, con el objetivo de evaluar el desempeño del API en futuras iteraciones donde se anticepe un mayor volumen de datos o la incorporación de nuevas funcionalidades. Estas pruebas permiten monitorear el rendimiento del API de manera continua, garantizando que mantenga su eficiencia y robustez a medida que evoluciona.

Gracias a la automatización y a la verificación adecuada, cada actualización permite que el API se despliegue de manera automática. Esto garantiza que cada cambio aprobado esté inmediatamente disponible para los usuarios, facilitando un flujo continuo de mejoras y optimizaciones en tiempo real.

Por último, una vez que el API se encuentra en producción, es monitoreada constantemente mediante la herramienta **Sentry**. Esta herramienta permite gestionar cualquier error inesperado de manera eficaz, proporcionando información detallada sobre cada incidente para facilitar una resolución rápida y eficiente.

Además, Sentry permite analizar el comportamiento de las solicitudes realizadas al API, ofreciendo un monitoreo continuo de métricas clave. Esto facilita una evaluación constante del rendimiento del API, asegurando que se mantenga en óptimas condiciones y responda adecuadamente a las demandas de los usuarios.

# CAPÍTULO 8

---

## Conclusiones

---

- **Eficiencia y robustez del sistema:** La elección de tecnologías adecuadas y la correcta estructuración de la base de datos han permitido construir un módulo de *Backend* eficiente y robusto, capaz de gestionar de manera óptima las solicitudes de los usuarios sobre sus registros y el análisis correspondiente.
- **Gestión y Estructura de Datos de Residuos:** A lo largo del proyecto, se estableció una estructura sólida y eficiente para almacenar información detallada sobre los tipos de residuos y la frecuencia de desecho de los usuarios. Asimismo, se desarrolló un sistema de base de datos integral que permite clasificar y gestionar de manera óptima los datos relacionados con la gestión de residuos. Esta infraestructura no solo facilita el análisis tanto individual como global de la información, sino que también permite generar resultados analíticos detallados que ofrecen una visión clara del comportamiento y patrones de los usuarios en relación con la gestión de residuos.
- **Autorización e Integridad de los Datos:** La integración de una bitácora de auditoría asegura la integridad de los datos de los usuarios, permitiendo rastrear cualquier cambio en caso necesario. Además, la implementación de JWT proporciona un nivel de seguridad y autenticación avanzado para el API, protegiendo así los datos de accesos no autorizados.
- **Análisis de Datos:** Las funcionalidades del API permiten ofrecer a los usuarios información personalizada sobre sus registros de desechos, mostrando el impacto positivo de sus acciones en el medio ambiente. Además, el sistema facilita la generación de análisis globales, permitiendo examinar los comportamientos colectivos de los usuarios y apoyar la toma de decisiones estratégicas.
- **Mejora Continua mediante CI/CD:** La implementación de un *pipeline* de CI/CD prepara al sistema para una mejora continua en cada actualización y facilita su escalabilidad a largo plazo. Las pruebas automatizadas aumentan la eficiencia y permiten identificar y corregir errores antes de que afecten el funcionamiento del API. Además, este enfoque posibilita un despliegue rápido de cambios, asegurando que las mejoras estén disponibles para los usuarios en tiempo real.
- **Monitoreo y Rendimiento del API:** La integración de **Sentry** para monitorear el sistema en producción permite una respuesta rápida ante errores que puedan afectar el funcionamiento del API. Además, el monitoreo del rendimiento garantiza que el sistema pueda adaptarse al crecimiento y a la incorporación de nuevas funcionalidades sin comprometer la velocidad y la eficiencia.

- **Escalabilidad y Futuras Expansiones:** La implementación de CI/CD, el diseño optimizado de la base de datos y las herramientas de monitoreo posicionan al proyecto para futuras expansiones. Este diseño permite que el sistema se escale y se introduzcan nuevas funcionalidades sin afectar su estabilidad ni rendimiento, asegurando una evolución fluida y robusta.

# CAPÍTULO 9

---

## Recomendaciones

---

- **Ampliación de las métricas ambientales:** Considerar la inclusión de nuevas métricas de análisis ambiental que permitan ofrecer a los usuarios información más detallada sobre cómo sus registros de desechos impactan positivamente en otras áreas ambientales. Esto podría incluir métricas adicionales relacionadas con la reducción de residuos, ahorro energético, o preservación de recursos naturales.
- **Análisis y fortalecimiento de la seguridad:** Realizar auditorías de seguridad periódicas en el API, incluyendo pruebas específicas para evaluar su comportamiento frente a posibles ataques o vulnerabilidades. Este enfoque permitirá identificar y mitigar riesgos proactivamente, asegurando la protección de los datos de los usuarios y la resiliencia del sistema frente a amenazas.
- **Funcionalidades Administrativas:** Implementar funcionalidades específicas para usuarios administrativos, como municipalidades o centros de reciclaje. Esto permitirá integrar el análisis de datos y facilitar la carga de información sobre desechos cuando sea necesario, mejorando la colaboración y gestión de datos ambientales a nivel institucional.
- **Aporte de los Usuarios:** Habilitar la opción para que los usuarios puedan sugerir la inclusión de nuevos tipos de desechos o registrar aquellos que no se encuentren disponibles en la aplicación. Esto fomentará la cooperación de la comunidad y permitirá el análisis de nuevos tipos de residuos, ampliando así la base de datos para futuras métricas y estudios.
- **Información Adicional sobre los Desechos:** Evaluar la incorporación de información adicional sobre los desechos, como características específicas o impacto ambiental detallado, lo cual permitirá enriquecer las métricas de análisis de datos y proporcionar a los usuarios una comprensión más completa de sus hábitos de gestión de residuos.

---

## Bibliografía

---

- [1] Bass, Len, Ingo Weber y Liming Zhu: *DevOps: A Software Architect's Perspective*. IEEE Software, 33(3):94–97, 2016.
- [2] Brown, Cynthia y Robert Allen: *Efficiency in Automated CI/CD Pipelines for RESTful APIs*. Journal of Software Engineering and Automation, 17(2):78–89, 2020.
- [3] CityAdapt: *Habitantes Ciudad de Guatemala*. <https://cityadapt.com/ciudad/ciudad-de-guatemala/#:~:text=Habitantes%20Ciudad%20de%20Guatemala%3A%20La,2022%20es%20de%201%2C213%2C651%20habitantes>, 2022. <https://cityadapt.com/ciudad/ciudad-de-guatemala/#:~:text=Habitantes%20Ciudad%20de%20Guatemala%3A%20La,2022%20es%20de%201%2C213%2C651%20habitantes>, Estimación de población en 2022 para la Ciudad de Guatemala.
- [4] Clarke, R.: *JouleBug: Sustainable Living Through Gamification*. Green Tech Media, 2016.
- [5] Codd, E. F.: *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, 13(6):377–387, 1970.
- [6] Coyle, K.: *Environmental Literacy in America: What Ten Years of NEETF/Roper Research and Related Studies Say About Environmental Literacy in the U.S.*, 2010.
- [7] Date, C. J.: *An Introduction to Database Systems*. Addison-Wesley, 2003.
- [8] Dean, J. y S. Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters*. Communications of the ACM, 51(1):107–113, 2008.
- [9] Demers, Hannah y Jordan Kim: *Automated Testing Techniques for RESTful APIs in CI/CD*. International Journal of Software Engineering and Applications, 11(2):43–60, 2020.
- [10] Elmasri, R. y S. B. Navathe: *Fundamentals of Database Systems*. Pearson, 7th edición, 2015.
- [11] European Youth Portal: *How to Reduce My Carbon Footprint*. [https://youth.europa.eu/get-involved/sustainable-development/how-reduce-my-carbon-footprint\\_en](https://youth.europa.eu/get-involved/sustainable-development/how-reduce-my-carbon-footprint_en), 2023. [https://youth.europa.eu/get-involved/sustainable-development/how-reduce-my-carbon-footprint\\_en](https://youth.europa.eu/get-involved/sustainable-development/how-reduce-my-carbon-footprint_en), Accedido: 2023-11-03.
- [12] Feathers, Michael: *Test Driven Development for Embedded C*. Addison-Wesley Professional, Boston, MA, 2019.
- [13] Ferris, Maggie: *How Does Automated Testing For CI/CD Help Improve Software Quality and Delivery*, Mar 2023. <https://qameta.io/blog/automated-testing-for-ci-cd/>, Accessed: 2023-10-31.

- [14] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Tesis de Doctorado, University of California, Irvine, 2000.
- [15] Fowler, M.: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2018.
- [16] Fowler, Martin y James Lewis: *Continuous Integration and Continuous Delivery: Principles and Practices*. Tech Books, New York, USA, 2018.
- [17] Gray, J. y A. Reuter: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1992.
- [18] Hack a Boss: *Frontend y Backend Explicados*, 2023. <https://www.hackaboss.com/blog/frontend-backend-explicados>.
- [19] Hailperin, M., B. Kaiser y K. Pfister: *Operating Systems and Middleware: Supporting Controlled Interaction*. Princeton University Press, 2007.
- [20] Han, J., J. Pei y M. Kamber: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.
- [21] Han, J., J. Pei y M. Kamber: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3rd edición, 2011.
- [22] Hardt, D.: *The OAuth 2.0 Authorization Framework*, 2012. <https://tools.ietf.org/html/rfc6749>, RFC 6749, Internet Engineering Task Force (IETF).
- [23] Harrison, G.: *Next Generation Databases: NoSQL and Big Data*. Apress, 2015.
- [24] Hartl, Michael: *Ruby on Rails Tutorial: Learn Web Development with Rails*. Addison-Wesley, Boston, MA, 3rd edición, 2016.
- [25] Holovaty, A. y J. Kaplan-Moss: *The Definitive Guide to Django: Web Development Done Right*. Apress, 2009.
- [26] Humble, Jez y David Farley: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, Boston, MA, 2010.
- [27] Jacobson, M.: *RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity*. O'Reilly Media, 2009.
- [28] Johnson, Emily y Victor Sanders: *Automating CI/CD with GitHub Actions: Streamlined Workflows for Modern Development*. Journal of DevOps and Automation, 13(1):33–49, 2022.
- [29] Jones, Laura: *Effective Monitoring and Alerting: For Web Operations*. O'Reilly Media, Sebastopol, CA, 2021.
- [30] Kaplan, Anna y Tyler Roberson: *Automated Testing in Continuous Integration: A Practical Guide*. Journal of Software Testing, 15(2):90–105, 2020.
- [31] Karan, Matthew: *Backend Architecture and Design*. Coding Press, San Francisco, CA, 2021.
- [32] Kim, Rachel y Philip Moore: *Continuous Deployment Strategies for Microservices and APIs*. International Journal of Software Engineering, 19(3):150–168, 2021.
- [33] Kim, Sarah y Peter Allen: *Real-Time Monitoring in CI/CD Pipelines: Techniques and Tools*. Software Development Journal, 23(4):210–225, 2022.
- [34] Kumar, Amit: *Backend Development Fundamentals*. Tech Books Publishing, New York, USA, 2020.
- [35] Lewis, Craig y Barbara Thompson: *Observability and Monitoring for High-Performance APIs*. Journal of DevOps and Observability, 5(1):19–33, 2021.

- [36] Martínez-Alier, J.: *The Environmentalism of the Poor: A Study of Ecological Conflicts and Valuation*. Edward Elgar Publishing, 2014.
- [37] McKinney, W.: *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, 2nd edición, 2017.
- [38] Meyer, David: *Automating Deployment Processes: Challenges and Solutions in CI/CD*. Journal of DevOps and Agile Development, 10(3):45–58, 2020.
- [39] Meyer, Laura y Frank O'Donnell: *Testing RESTful APIs in CI/CD Pipelines: Best Practices and Tools*. Journal of Software Testing, 18(3):120–135, 2021.
- [40] Ministerio de Ambiente y Recursos Naturales: *Reglamento 164-2021*, n.d. <https://www.marn.gob.gt/reglamento-164-2021/>.
- [41] Nick: *What Is REST API?*, 2020. <https://blog.iron.io/what-is-rest-api/>.
- [42] Nielsen, Jakob: *Usability Engineering*. Morgan Kaufmann, Amsterdam, 1994, ISBN 0125184069.
- [43] OpenJS Foundation: *Express - Node.js web application framework*, 2020. <https://expressjs.com/>.
- [44] Pacheco, Laura y John Smith: *Understanding the Role of Backend in Modern Web Applications*. Journal of Web Development, 12(3):45–58, 2019.
- [45] Patel, Ayesha y Michael Lee: *Real-Time Monitoring Techniques in CI/CD: Tools and Methodologies*. Journal of Software Development and Operations, 12(4):56–72, 2020.
- [46] Pedamkar, Priya: *Second Normal Form*, 2020. <https://www.educba.com/second-normal-form/>.
- [47] Pérez, C.: *El 95 del agua se desperdicia en Guatemala debido a la contaminación ambiental*. Prensa Libre, August 7 2017. <https://www.prenslibre.com/ciudades/el-95-del-agua-se-desperdicia-en-guatemala-debido-a-contaminacion-ambiental/#:~:text=Ciudades->.
- [48] Richardson, Leonard y Sam Ruby: *RESTful Web Services*. O'Reilly Media, Sebastopol, CA, 2007.
- [49] Smith, Karen y Robert Williams: *CI/CD Tools and Pipelines: Best Practices for Automation*. IT Books Press, San Francisco, CA, 2021.
- [50] Stevenson, Eric y Mark Johnson: *DevOps and Automation for Continuous Delivery: Tools and Techniques*. IEEE Software, 36(4):23–31, 2019.
- [51] Stonebraker, M.: *SQL databases v. NoSQL databases*. Communications of the ACM, 53(4):10–11, 2010.
- [52] Stonebraker, M. y J. M. Hellerstein: *What goes around comes around*. En *Readings in Database Systems, 4th Edition*. MIT Press, 2005.
- [53] The PostgreSQL Global Development Group: *PostgreSQL Documentation*, 2020. <https://www.postgresql.org/docs/>.
- [54] Tilkov, S. y S. Vinoski: *Node.js: Using JavaScript to Build High-Performance Network Programs*. IEEE Internet Computing, 14(6):80–83, 2010.
- [55] United Nations Environment Programme: *MyWaste App: Empowering Citizens for Better Waste Management*, 2020. <https://www.unep.org>.
- [56] Wallace, John y Emily Gordon: *Scalable API Testing in CI/CD Environments*. Software Quality Journal, 27(4):231–245, 2019.

- [57] Williams, Sarah y Mark Johnson: *Essential Components of Backend Development*. International Journal of Software Engineering, 18(2):23–36, 2020.

## ANEXO A

### Pruebas de carga y rendimiento en API de producción

Se llevaron a cabo pruebas de carga y rendimiento directamente sobre la API desplegada en una infraestructura de AWS Beanstalk. Estas pruebas, realizadas con **Artillery**, consistieron en ejecutar solicitudes de análisis de datos generales sobre los datos almacenados en la base de datos de producción. Este enfoque permitió evaluar la capacidad de respuesta y el rendimiento de la API bajo condiciones de uso intensivo, asegurando su estabilidad y eficiencia en un entorno real.

El script de ejecución de pruebas fue el siguiente:

```
1 config:
2   target: rutaAPI
3   phases:
4     - duration: 60 # Duracion de la prueba en segundos
5       arrivalRate: 35 # Numero de usuarios simulados por segundo
6   defaults:
7     headers:
8       content-type: "application/json"
9       Authorization: "Bearer tokenAutorizacion" # Token JWT
10
11 scenarios:
12   - name: Test de cantidad de desechos reciclables generales
13     flow:
14       - get:
15         url: "/analysis/all/recyclable-waste-stats"
16
17   - name: Test de cantidad de ahorro de agua general
18     flow:
19       - get:
20         url: "/analysis/all/water-savings"
21
22   - name: Test de cantidad de ahorro de CO2 general
23     flow:
24       - get:
25         url: "/analysis/all/co2-savings"
```

```

27   - name: Test de top 5 desechos generales
28     flow:
29       - get:
30         url: "/analysis/all/top5-waste-types"

```

Listing A.1: Archivo performance.yml

Los resultados para 35 solicitudes por segundo por 1 minuto seguido son:

```

All VUs finished. Total time: 1 minute, 2 seconds

-----
Summary report @ 01:02:45(-0600)
-----

errors.ETIMEDOUT: ..... 4
http.codes.200: ..... 2096
http.downloaded_bytes: ..... 93441
http.request_rate: ..... 35/sec
http.requests: ..... 2100
http.response_time:
  min: ..... 59
  max: ..... 453
  mean: ..... 86.4
  median: ..... 77.5
  p95: ..... 144
  p99: ..... 223.7
http.response_time.2xx:
  min: ..... 59
  max: ..... 453
  mean: ..... 86.4
  median: ..... 77.5
  p95: ..... 144
  p99: ..... 223.7
http.responses: ..... 2096
vusers.completed: ..... 2096
vusers.created: ..... 2100
vusers.created_by_name.Test de cantidad de ahorro de CO2 general: ..... 515
vusers.created_by_name.Test de cantidad de ahorro de agua general: ..... 537
vusers.created_by_name.Test de cantidad de desechos reciclables generales: ..... 516
vusers.created_by_name.Test de top 5 desechos generales: ..... 532
vusers.failed: ..... 4
vusers.session_length:
  min: ..... 114
  max: ..... 1193.6
  mean: ..... 156.4
  median: ..... 144
  p95: ..... 228.2
  p99: ..... 308
Log file: test/performance/artillery-report.json
PS D:\UNIVERSIDAD\PROYECTO-GRADUACION\mobile-app\eco-scan-backend> []

```

Figura A.1: Resultados prueba de carga del 50 %.

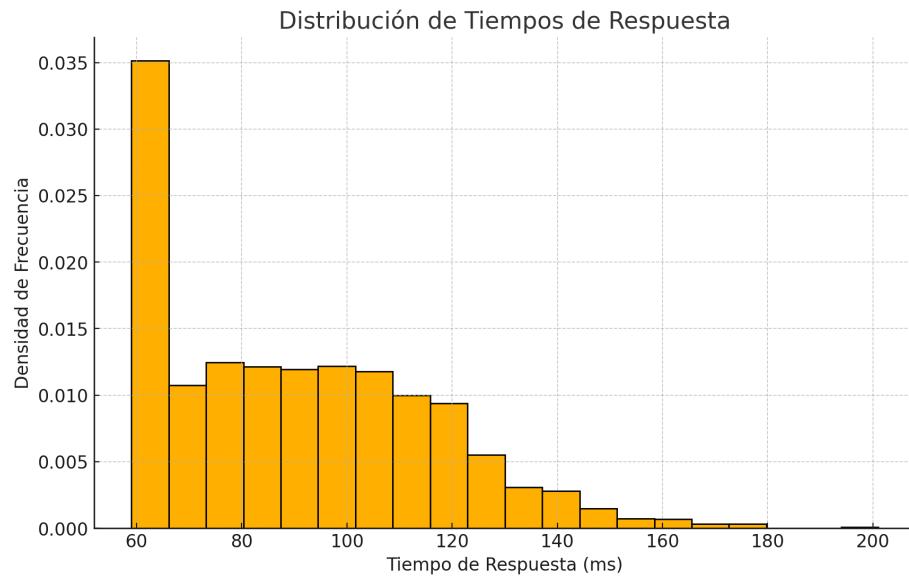


Figura A.2: Resultados prueba de carga del 50 %.

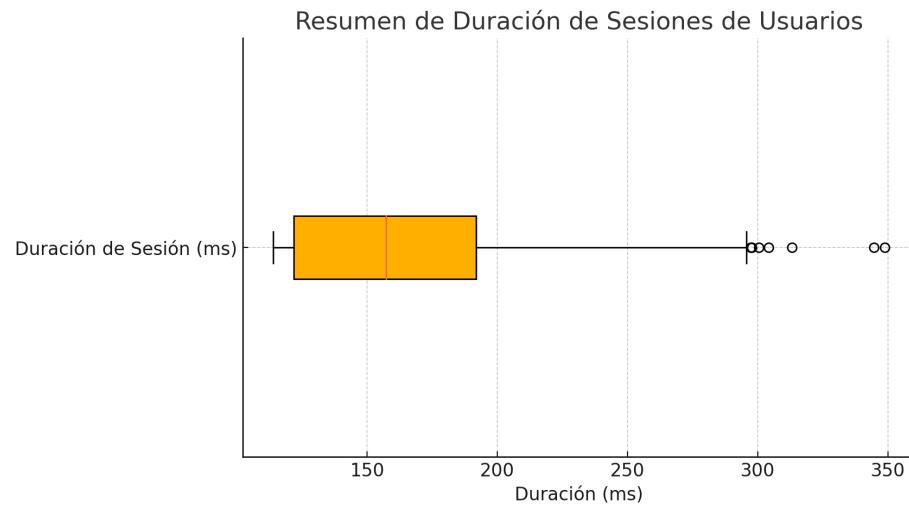


Figura A.3: Resultados prueba de carga del 50 %.

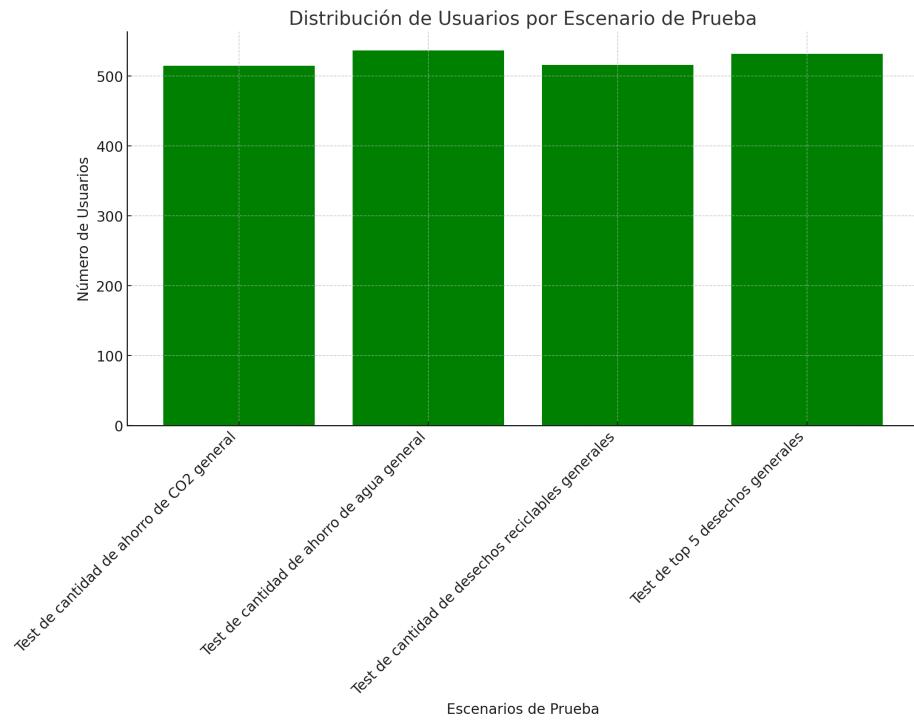


Figura A.4: Resultados prueba de carga del 50 %.

Los resultados para 35 solicitudes por segundo por 2 minuto seguidos son:

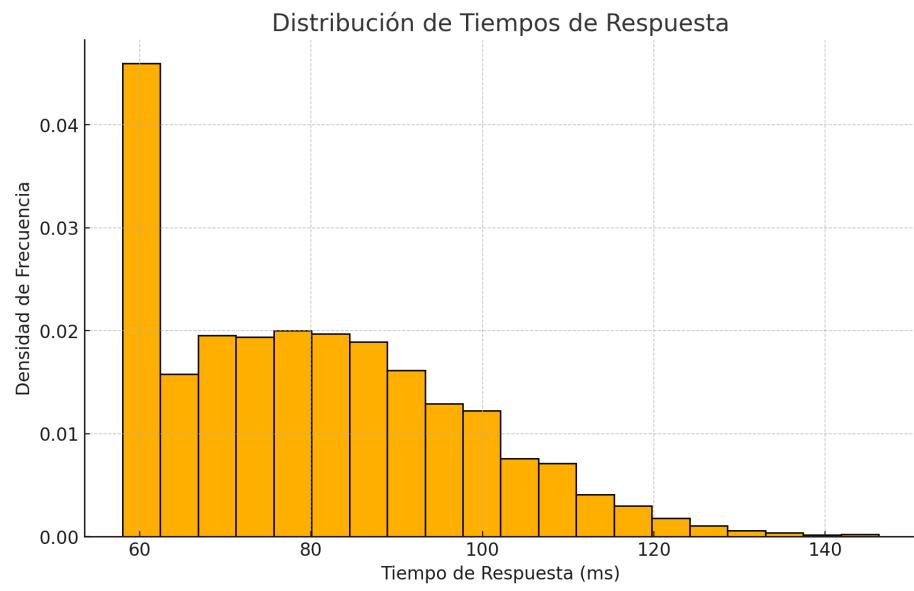


Figura A.5: Resultados prueba de carga del 50 %.

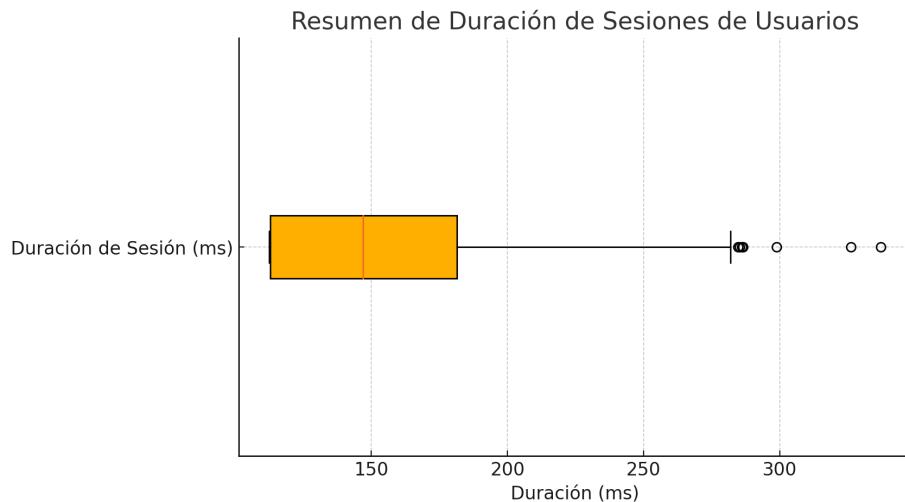


Figura A.6: Resultados prueba de carga del 50 %.

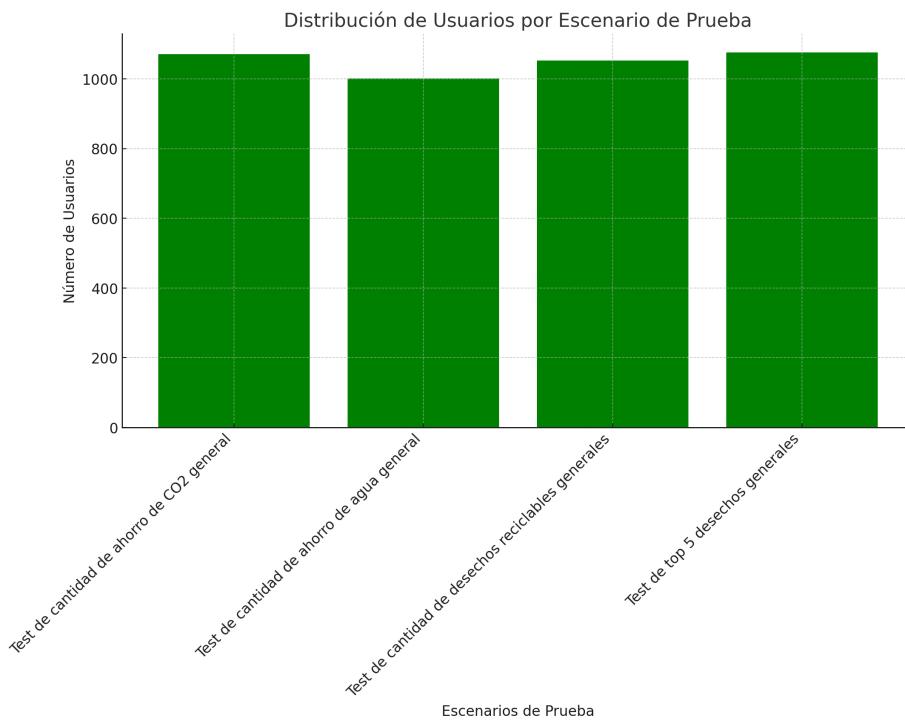


Figura A.7: Resultados prueba de carga del 50 %.

Los resultados para 70 solicitudes por segundo por 1 minuto seguido son:

```
All VUs finished. Total time: 1 minute, 8 seconds

-----
Summary report @ 01:06:04(-0600)
-----

errors.ETIMEDOUT: ..... 3
http.codes.200: ..... 4197
http.downloaded_bytes: ..... 186991
http.request_rate: ..... 59/sec
http.requests: ..... 4200
http.response_time:
    min: ..... 135
    max: ..... 6998
    mean: ..... 4211.6
    median: ..... 4583.6
    p95: ..... 5168
    p99: ..... 6976.1
http.response_time.2xx:
    min: ..... 135
    max: ..... 6998
    mean: ..... 4211.6
    median: ..... 4583.6
    p95: ..... 5168
    p99: ..... 6976.1
http.responses: ..... 4197
vusers.completed: ..... 4197
vusers.created: ..... 4200
vusers.created_by_name.Test de cantidad de ahorro de CO2 general: ..... 1091
vusers.created_by_name.Test de cantidad de ahorro de agua general: ..... 1042
vusers.created_by_name.Test de cantidad de desechos reciclables generales: ..... 1033
vusers.created_by_name.Test de top 5 desechos generales: ..... 1034
vusers.failed: ..... 3
vusers.session_length:
    min: ..... 205.6
    max: ..... 7312.5
    mean: ..... 4298
    median: ..... 4676.2
    p95: ..... 5378.9
    p99: ..... 6976.1
Log file: test/performance/artillery-report.json
PS D:\UNIVERSIDAD\PROYECTO-GRADUACION\mobile-app\eco-scan-backend> []
```

Figura A.8: Resultados prueba de carga del 100 %.

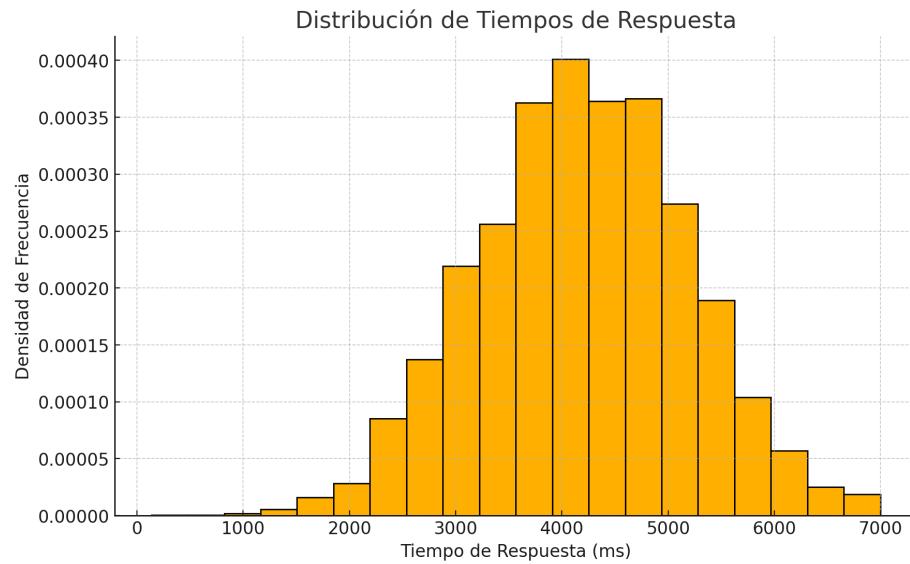


Figura A.9: Resultados prueba de carga del 100 %.

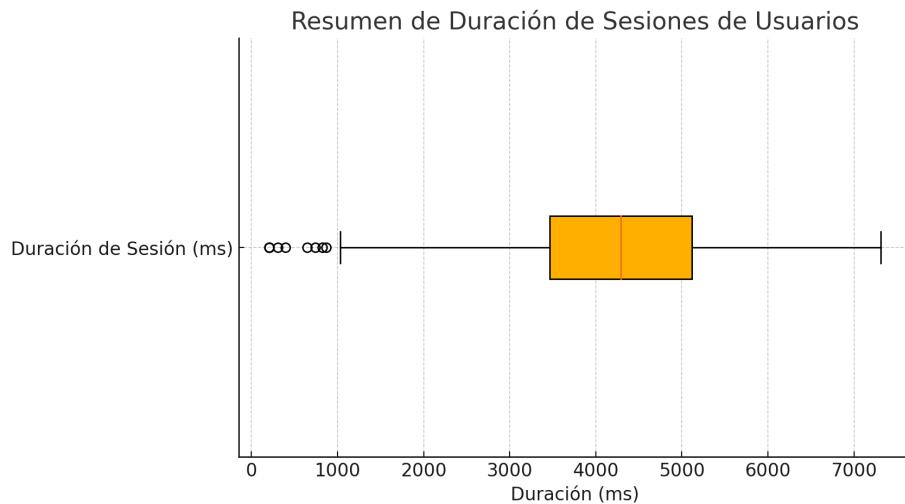


Figura A.10: Resultados prueba de carga del 100 %.

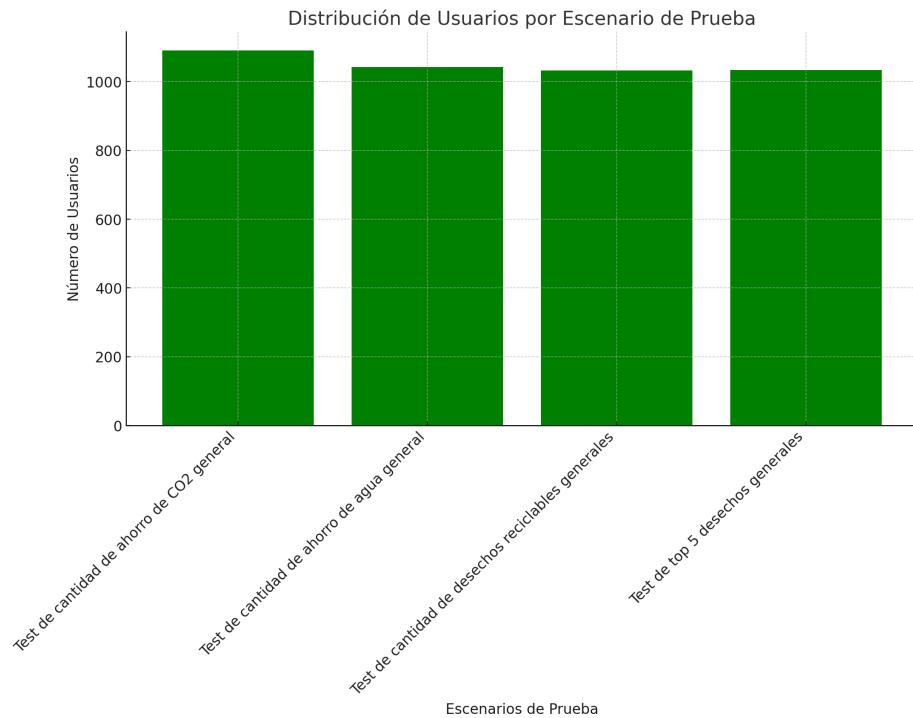


Figura A.11: Resultados prueba de carga del 100 %.