UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CARRERA DE CIENCIA DE LA COMPUTACIÓN

MONITOREO EN TIEMPO REAL DE CASOS DE DENGUE EN LA REGIÓN LIMA USANDO ARQUITECTURA DE MICROSERVICIOS

TESIS

Para optar el título profesional de Licenciado en Ciencia de la Computación

AUTOR:

Piero Angelo Morales Alcalde

ASESOR

Jesus Edwin Bellido Angulo

Lima - Perú

2024

UTEC UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA | Dirección de Investigación

DECLARACIÓN JURADA

Yo, Mónica Cecilia Santa María Fuster identificada con DNI No 18226712 en mi condición de autoridad responsable de validar la autenticidad de los trabajos de investigación y tesis de la UNIVERSIDAD DE INGENIERIA Y TECNOLOGIA, DECLARO BAJO JURAMENTO:

Que la tesis denominada "MONITOREO EN TIEMPO REAL DE CASOS DE DENGUE EN LA REGIÓN LIMA USANDO ARQUITECTURA DE MICROSERVICIOS" ha sido elaborada por el señor Piero Angelo Morales Alcalde, con la asesoría de Jesús Edwin Bellido Angulo, identificado con el DNI N°41994747, y que se presenta para obtener el grado de Licenciado en Ciencia de la Computación, ha sido sometida a los mecanismos de control y sanciones anti plagio previstos en la normativa interna de la universidad, encontrándose un porcentaje de similitud de 0%.

En fe de lo cual firmo la presente.

Dra. Mónica Santa María Fuster
Directora de Investigación

En Barranco, el 17 de enero de 2024

Dedicatoria:

A mi familia,
a mis amigos,
a mis profesores,
a mis compañeros,
y a todos los que me apoyaron.

Agradecimientos:

Agradezco a mi familia por el apoyo incondicional que me brindaron durante toda mi carrera universitaria, a mis amigos por los buenos momentos que pasamos juntos y a mi asesor por su guía y apoyo durante la realización de este trabajo de investigación.

Índice general

Pág.

Índice de figuras

RESUMEN

El preocupante aumento de casos de dengue en Perú, especialmente en Lima, ha generado alarma, con un asombroso incremento del 323.5 % de 63,168 casos en 2022 a 268,660 casos hasta la semana 43 de 2023. Lima experimentó un drástico aumento del 3,190.6 %, pasando de 938 casos en 2022 a 30,946 hasta la semana 43 de 2023. Esta situación plantea riesgos significativos para la salud en Lima, con una población densamente poblada de aproximadamente 9,674,755 habitantes, y la posibilidad de tensionar su sistema de salud debido a la falta de experiencia con el dengue, cuyos primeros casos autóctonos se reportaron en febrero de 2022.

Para abordar este desafío, se propone un proyecto que visualiza informes de casos de dengue en tiempo real, permitiendo intervenciones inmediatas. La falta de control en áreas infectadas podría llevar a la saturación del sistema de salud, especialmente con el aumento de ingresos a UCI relacionados con COVID-19. La justificación radica en la necesidad de monitoreo en tiempo real para respuestas rápidas, como fumigaciones específicas y tratamiento oportuno.

El proyecto busca implementar una herramienta de monitoreo para controlar mejor las zonas infectadas con dengue, mitigando la propagación de la enfermedad y evitando la sobrecarga del sistema de salud. El objetivo es construir un sistema de software capaz de manejar 5,538 solicitudes diarias basadas en los casos de 2022, garantizando actualizaciones en tiempo real y mejorando la precisión de los informes con coordenadas georreferenciadas.

Finalmente, se desarrolló un backend basado en microservicios, demostrando mejoras en disponibilidad y tiempos de respuesta en comparación con una arquitectura monolítica. Una arquitectura híbrida superó a los microservicios, manteniendo un tiempo de respuesta de menos de 1.5 segundos a 4,000 solicitudes por segundo. Y por otro lado, el algoritmo de clusterización facilita un análisis temporal detallado de los registros de

dengue, mejorando la identificación de patrones para intervenciones más específicas.

Palabras clave:

Dengue; Casos en tiempo real; Microservicios; Monitoreo de enfermedades; Georreferenciación; Plataforma web; Prevención y control de enfermedades

ABSTRACT

REAL-TIME MONITORING OF DENGUE CASES IN THE LIMA REGION USING MICROSERVICES ARCHITECTURE

The worrying increase in dengue cases in Peru, especially in Lima, has generated alarm, with a staggering increase of 323.5 % from 63,168 cases in 2022 to 268,660 cases through week 43 of 2023. Lima experienced a drastic increase of 3,190.6 % , going from 938 cases in 2022 to 30,946 until week 43 of 2023. This situation poses significant health risks in Lima, with a densely populated population of approximately 9,674,755 inhabitants, and the possibility of straining its health system due to the lack of experience with dengue, whose first indigenous cases were reported in February 2022.

To address this challenge, a project is proposed that visualizes dengue case reports in real time, allowing immediate interventions. Lack of control in infected areas could lead to saturation of the health system, especially with the increase in ICU admissions related to COVID-19. The rationale lies in the need for real-time monitoring for rapid responses, such as targeted fumigations and timely treatment.

The project seeks to implement a monitoring tool to better control areas infected with dengue, mitigating the spread of the disease and avoiding overloading the health system. The goal is to build a software system capable of handling 5,538 daily requests based on 2022 cases, ensuring real-time updates and improving the accuracy of reports with georeferenced coordinates.

Finally, a microservices-based backend was developed, demonstrating improvements in availability and response times compared to a monolithic architecture. A hybrid architecture outperformed microservices, maintaining a response time of less than 1.5 seconds at 4,000 requests per second. And on the other hand, the clustering algorithm

facilitates a detailed temporal analysis of dengue records, improving the identification of patterns for more specific interventions.

Keywords:
Dengue; Real-time cases; Microservices; Disease monitoring; Georeferencing; Web platform; Prevention and control of diseases

Capítulo 1

INTRODUCCIÓN

El dengue es una de las enfermedades con más incremento de casos en Perú, pasando de 63,168 casos en 2022 a 268,660 casos hasta la semana 43 del 2023, lo que significa un incremento de 323.5 % en menos de un año [1]. Esto también ha afectado a Lima, en donde también se presentó un considerable incremento de casos, pasando de 938 casos en 2022 a 30,946 hasta la semana 43 del 2023. Esto significa un incremento de 3,190.6 % para la semana 43 del 2023 [2]. Este incremento, se puede ver en la Figura 1.1, donde se muestra el número de casos de dengue en Perú y en Lima desde el 2018 hasta la semana 43 del 2023.

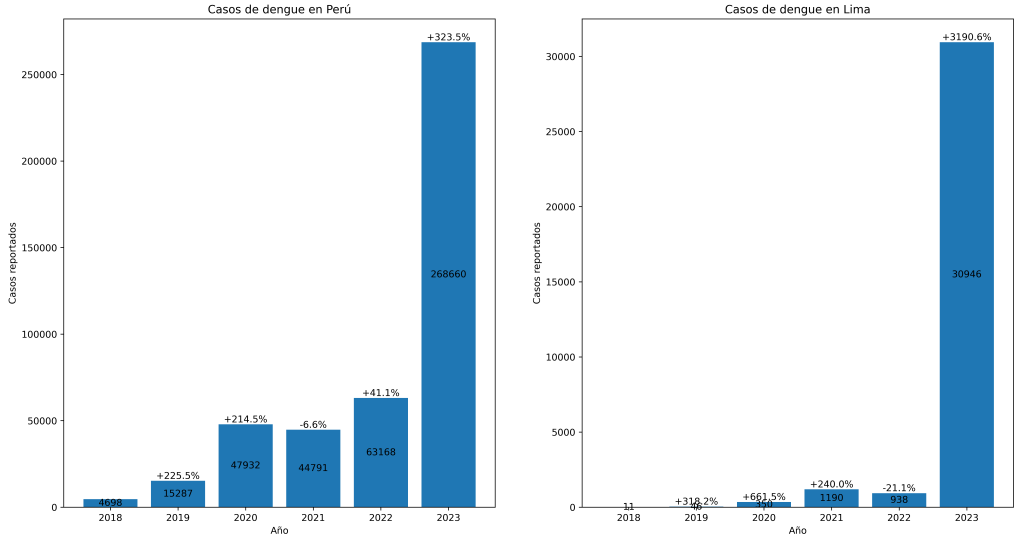


FIGURA 1.1: Casos de dengue reportados en Perú y Lima de 2018 a la semana 43 del 2023.

El incremento de casos de dengue en una ciudad tan poblada como Lima, con aproximadamente 9,674,755 habitantes y una densidad poblacional promedio de 3,697 personas por kilómetro cuadrado [3] podrían llegar a saturar el sistema de salud, ya que se enfrentaría a una enfermedad con la que no tiene experiencia previa, y en donde ya se han comenzado a reportar los primeros casos autóctonos desde febrero de 2022 [4].

Por este motivo, la presente investigación tiene el fin de dar una respuesta rápida frente a nuevos casos y de esta forma ayudar al Centro Nacional de Epidemiología, Prevención y Control de Enfermedades (CDC MINSA) y a los centros de salud locales a tener un mejor control de las zonas afectadas.

La plataforma usada actualmente para visualizar los casos de varias enfermedades (incluido el Dengue) es la Sala virtual de situación de salud [2], en donde se encuentran registrados todos los casos que han sido reportados a nivel nacional. Sin embargo, la precisión con la que se realizan estos reportes es a nivel de distritos. Además, la frecuencia con la que se actualizan los datos es semanalmente, de acuerdo con el número de semana epidemiológica.

Por otra parte, en la investigación realizada por Eric M. Delmelle, Hongmei Zhu, Wenwu Tang y Irene Casas [5] se propone una plataforma web con un conjunto de herramientas geoespaciales, para el monitoreo de la enfermedad de dengue. Aquí se puede mostrar la información de diferentes maneras, como la ubicación de los reportes de casos, la densidad de casos en una determinada área y conexiones espacio temporales entre los casos durante un determinado periodo de tiempo. Sin embargo, en esta investigación la información no se muestra en tiempo real, debido a que se basa en *datos estáticos* extraídos tiempo antes de mostrar la información.

La importancia de la actualización de los datos en tiempo real se basa en la necesidad de poder actuar con la mayor inmediatez posible frente a los nuevos casos que sean reportados, esto debido principalmente a dos razones. Primero, dar la posibilidad que las entidades encargadas puedan realizar fumigaciones lo más pronto posible, en las

zonas con una mayor densidad de casos reportados. Esto es importante debido a que en el 2022, la efectividad de la fumigación ha logrado alcanzar el 98 % de efectividad para eliminar el zancudo que transmite el dengue [6]. Segundo, es permitir al Ministerio de Salud (MINSA) poder aplicar tratamiento y/o medicamentos a las personas que ya han sido infectadas, con el objetivo de evitar complicaciones y pérdida de vidas por el dengue [7].

Esto crea una brecha de investigación sobre la que se basará el desarrollo de este proyecto, donde se realizará la visualización de los reportes de casos de dengue registrados en tiempo real.

1.1 Descripción del problema

El incremento de los casos de dengue en la región de Lima a partir del 2020 hizo que se incremente el interés en monitorizar esta enfermedad, especialmente porque no es muy común en la región Lima. Este problema trae consigo consecuencias severas, como un incremento de muertes, así como también la saturación del sistema de salud, en donde ya se ha incrementado la tasa de hospitalización en las camas UCI debido al COVID-19, pasando de 1,553 durante el 2020 a 1,865 durante el 2021 para un total de 1,900 camas UCI disponibles [8]. Esto deja una poca capacidad de respuesta frente a otras enfermedades, en donde una de las principales es el dengue.

Por consiguiente, este proyecto se enfocará en tener un mayor control de zonas infectadas con dengue, con el propósito de prevenir la propagación de la enfermedad y evitar la saturación del sistema de salud. Para ello, es importante implementar una herramienta de monitoreo que ayude a tener una supervisión mayor de las zonas infectadas.

La estructura del problema de investigación se puede observar en el árbol de problema mostrado en la Figura 1.2. Esta investigación se centrará en la falta de control de

zonas infectadas que produce la saturación del sistema de salud y trae consigo el incremento de casos de dengue en la región de Lima.

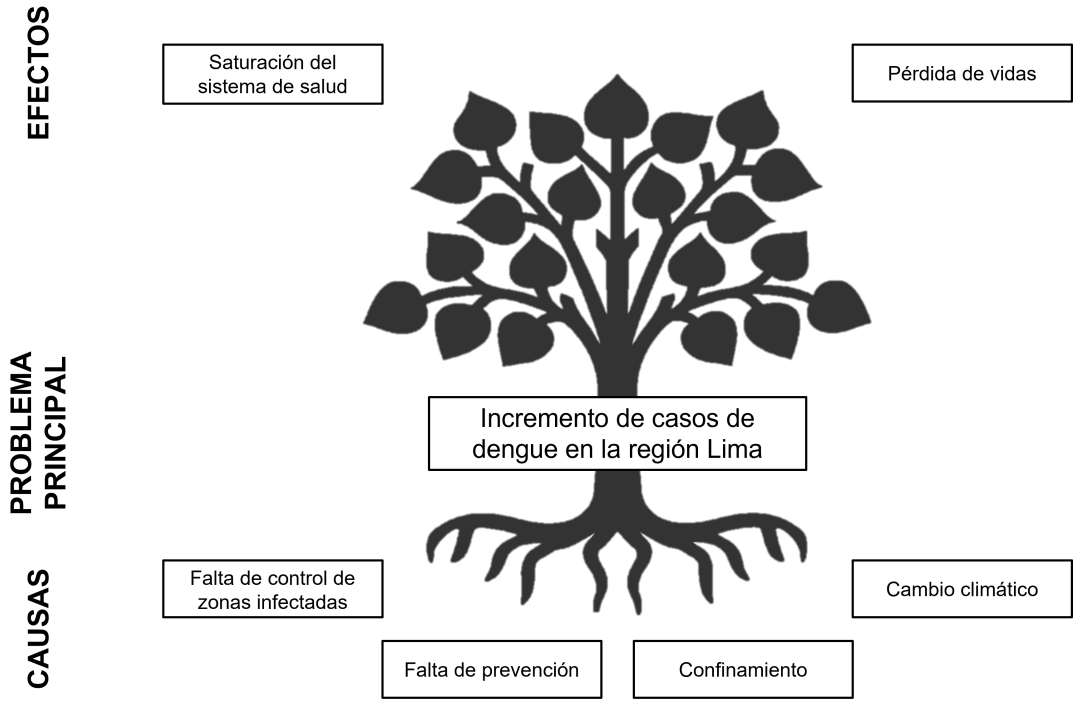


FIGURA 1.2: Árbol del problema de investigación.

1.2 Justificación

En este contexto, la importancia de desarrollar un sistema de monitorización en tiempo real de los casos de dengue en la región Lima, radica en la necesidad de poder tener un mejor control de las zonas afectadas con esta enfermedad. De esta manera, se pueden crear medidas que puedan frenar el avance de la enfermedad, y evitar su propagación hacia otras zonas.

Para este proyecto, se define tiempo real como la funcionalidad de poder registrar y visualizar los reportes de dengue en un rango de tiempo no mayor a 10 segundos desde que estos reportes son registrados en la aplicación móvil hasta que pueden ser visualizados en

la plataforma web. Se ha puesto como meta lograr el tiempo de actualización menor a 10 segundos, basado en la investigación realizada por Suznjevic [9], en donde se mencionan los tiempos de respuesta de diferentes sistemas computacionales dependiendo del uso de los mismos.

En cuanto a la parte computacional, la importancia radica en poder construir un sistema que pueda trabajar en tiempo real con los datos, desde que son enviados de la aplicación móvil hasta poder ser visualizados en la plataforma web. Para ello, será necesario crear una solución computacional que pueda procesar una elevada cantidad de solicitudes en simultáneo, permitiendo procesar y visualizar los datos.

1.3 Objetivos

1.3.1 Objetivo general

Esta investigación busca construir un sistema de software capaz de soportar una carga de 5,538 solicitudes por día[1], con el propósito de tolerar las peores condiciones de tráfico real que fueron las registradas durante la pandemia del COVID-19 en el Perú [10]. Además, se debe permitir la actualización de los reportes de dengue en tiempo real y mejorar la precisión de estos reportes a un nivel de coordenadas georreferenciadas.

1.3.2 Objetivos específicos

- Construir una aplicación móvil en iOS y Android que permita reportar nuevos casos de Dengue de manera rápida, automatizando el registro de datos básicos como la fecha, hora y ubicación. Además, permitir la posibilidad de agregar notas y/o imágenes en cada registro.

[1] Dato obtenido a partir de los 2,021,237 casos registrados en el Perú durante el año 2022, año en donde se registraron más casos en comparación con 1,087,692, 1,364,514 y 50,883 de los años 2020, 2021 y 2023 respectivamente [10].

- Construir un componente en un servidor que se encargue de recibir las peticiones, tanto por parte de la aplicación móvil, como de la plataforma web. Además de poder comunicarse con una Base de Datos en donde se guardarán todos los registros. Y finalmente, que tenga la posibilidad de mostrar patrones, en tiempo real, entre la ubicación de los reportes de dengue y diferentes instantes de tiempo.

- Construir una plataforma web para visualizar en tiempo real los nuevos casos de Dengue que se reporten y dar la posibilidad de trabajar con los datos ya sea en la misma plataforma web, así como también tener la posibilidad de descargarlos.

Los componentes que se construirán en cada uno de los objetivos específicos se muestran en la Figura 1.3.

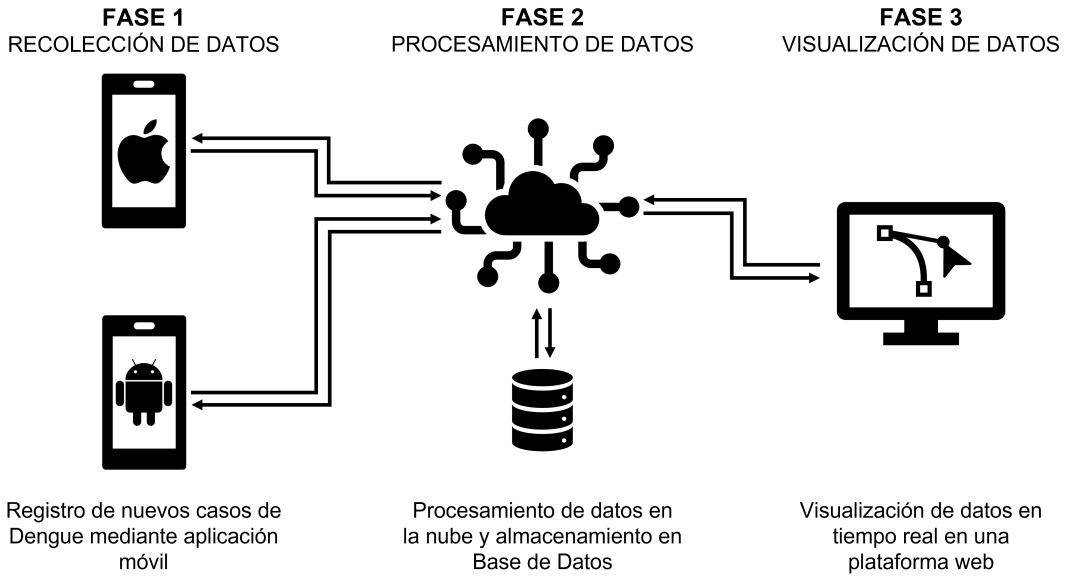


FIGURA 1.3: Diagrama de arquitectura del sistema de monitoreo en tiempo real.

1.4 Revisión de la Literatura

La masificación de los dispositivos móviles durante los últimos años ha permitido el desarrollo de aplicaciones móviles con diferentes funcionalidades, entre las cuales

se encuentra el monitoreo de enfermedades. Una de las primeras aplicaciones con este propósito fue la desarrollada por Reddy *et al*. [11], en donde se desarrolló una aplicación para móviles con el objetivo de verificar los síntomas de dengue, y de reportar casos nuevos de dengue en el país de Fiyi. En esta aplicación, los usuarios pueden responder si tienen o no una serie de síntomas para que al final la aplicación les dé un diagnóstico de si han contraído o no dengue. Además, si ya han contraído dengue, pueden reportar su caso en la aplicación rápidamente, automatizando el ingreso de la ubicación en donde se encuentran con la ayuda del GPS del dispositivo. Sin embargo, estos datos que son reportados solamente se limitan a unas coordenadas georreferenciadas, más no se puede agregar más detalles en cada reporte, tales como la fecha, hora, o algún dato adicional que permita realizar un análisis más profundo de los nuevos casos registrados.

Años después, Vutla *et al*. [12] propone una aplicación para el monitoreo de casos nuevos de dengue en India. En esta aplicación, los usuarios pueden verificar si la zona en donde se encuentran tiene algún brote de dengue o se han reportado varios casos. De esta manera, brinda información sobre qué tan segura es una zona y ver si se debe tomar precauciones adicionales. Al igual que en la investigación de Reddy *et al*. [11], la aplicación permite que los usuarios puedan saber si han contraído dengue, respondiendo una serie de preguntas sobre los síntomas que presentan. Además, también se permite realizar nuevos reportes de dengue, dependiendo del resultado del diagnóstico que da la aplicación. Sin embargo, en la ubicación de los nuevos reportes, sólo se podía colocar la ubicación en formato de texto, correspondiente a la ciudad donde se realiza el reporte. Esto afecta el análisis de datos, ya que no se tiene una buena precisión en la ubicación de los casos de dengue, además que tampoco se puede agregar datos adicionales cuando se reporta el nuevo caso de dengue.

Por otro lado, con respecto al procesamiento de datos georreferenciados, la investigación realizada por MacEachren *et al*. [13] se diseñaron tres métodos para poder

representar datos de salud georreferenciados. El primero es especializado en realizar comparaciones entre los datos de salud y la confiabilidad de los mismos todo mostrado en un mismo mapa. El segundo, es mostrar con una gama de colores la confiabilidad, para que de esta manera se pueda identificar los datos con menor confiabilidad. Y finalmente, un método que permite mostrar los conglomerados de los datos de salud en un mapa, con colores que representan la confiabilidad y coloreados por una leyenda. Sin embargo, este sistema de representación y procesamiento de datos georreferenciados sólo está limitado a mostrar los resultados por ciudades, más no en áreas mucho más específicas como a nivel de distritos o incluso a nivel de calles.

Así mismo, en la investigación realizada por Cao *et al*. [14] se buscaron relaciones entre espacios verdes y enfermedades cardiovasculares con el uso de datos georreferenciados. En esta investigación, se utilizó un modelo de regresión ponderado geográficamente y se cuantificaron las asociaciones espaciales entre espacios verdes y enfermedades cardiovasculares. Además, se proporcionan sugerencias sobre cómo planificar diferentes espacios verdes para el desarrollo de ciudades sostenibles, en base a los resultados obtenidos del procesamiento de estos datos. Sin embargo, este procesamiento está limitado a mostrar una especie de mapas de calor de las enfermedades cardiovasculares, y no permite la posibilidad de mostrar la información de alguna otra manera, o incluyendo otro tipo de análisis.

Por otro lado, respecto a la visualización de datos en tiempo real, la investigación realizada por Yang *et al*. [15] describe la construcción de una plataforma integrada mediante un entorno de clúster basado en Hadoop, Spark y un entorno de visualización y almacenamiento para la visualización de la calidad del aire y datos de enfermedades. Esta plataforma fue construida con el objetivo de resolver un problema de ineficiencia en el procesamiento de grandes cantidades de datos. Como resultado, esta plataforma permite visualizar la calidad de aire y los datos de enfermedades similares a la influencia en tiempo real, además de presentar análisis de asociación y discusión entre la calidad del aire y

el síndrome gripal. Sin embargo, esta plataforma está pensada más para trabajar con una gran cantidad de datos, que en tiempo real. Esto contrasta con la presente investigación, debido a que no se van a tener una gran cantidad de datos, pero sí se necesita que el tiempo de actualización sea el menor posible.

Por último, la investigación realizada por Guo *et al*. [16] se propone la construcción de un sistema de visualización de datos de la red eléctrica en tiempo real. En esta investigación, la plataforma de visualización de datos se encarga de estimar la frecuencia de oscilación y la amortiguación en tiempo real. Además, también se realizó la construcción de un sitio web de monitoreo en tiempo real que muestra la frecuencia dominante estimada y el amortiguamiento de las interconexiones de la red eléctrica de América del Norte. Si bien el sistema de comunicación para la transmisión de datos en tiempo real es muy relevante para la presente investigación, todo el procesamiento de los datos se realizan de forma local en cada navegador, y no centralizado en un servidor, lo cual disminuiría la cantidad de datos que se necesitarían transmitir a la plataforma web.

Como se puede observar, las investigaciones mencionadas anteriormente muestran deficiencias en dos puntos principales. Por un lado, la ubicación de los datos de incidencias no es precisa, ya que se limita, como máximo, a la ciudad donde se realiza el reporte de la incidencia. Por otro lado, las investigaciones mencionadas anteriormente muestran deficiencias en la visualización de los datos en tiempo real de las incidencias, ya que ninguna de ellas está pensada para mostrar los datos en tiempo real, sino que se limitan a mostrar los datos de forma estática. Estos dos puntos son los que se van a tratar de resolver en la presente investigación.

Capítulo 2

MARCO TEÓRICO

2.1 Reportes de dengue

Para realizar el reporte de nuevos casos de dengue y poder hacer el seguimiento adecuado al avance de esta enfermedad, en esta investigación, cada reporte incluirá datos como la ubicación geográfica (en coordenadas georreferenciadas), el detalle de la ubicación (dirección, distrito, código postal, etc.), la fecha y la hora en la que se realiza el reporte, y por último como datos adicionales se incluirá la posibilidad de adjuntar un comentario y/o fotografía en cada reporte. La importancia de cada uno de estos datos se detallan a continuación.

2.1.1 Sistema de Información Geográfica

Según la National Geographic Society [17], un Sistema de Información Geográfica (GIS por sus siglas en inglés) es un sistema informático para capturar, almacenar, verificar y mostrar datos relacionados con las posiciones en la superficie de la Tierra. Un GIS puede mostrar diferentes tipos de datos en un mapa, como calles, edificios y vegetación, lo cual permite analizar y comprender patrones y relaciones fácilmente. Estos datos pueden mostrarse en un solo mapa o de manera independiente, tal y como se muestra en la Figura 2.1.

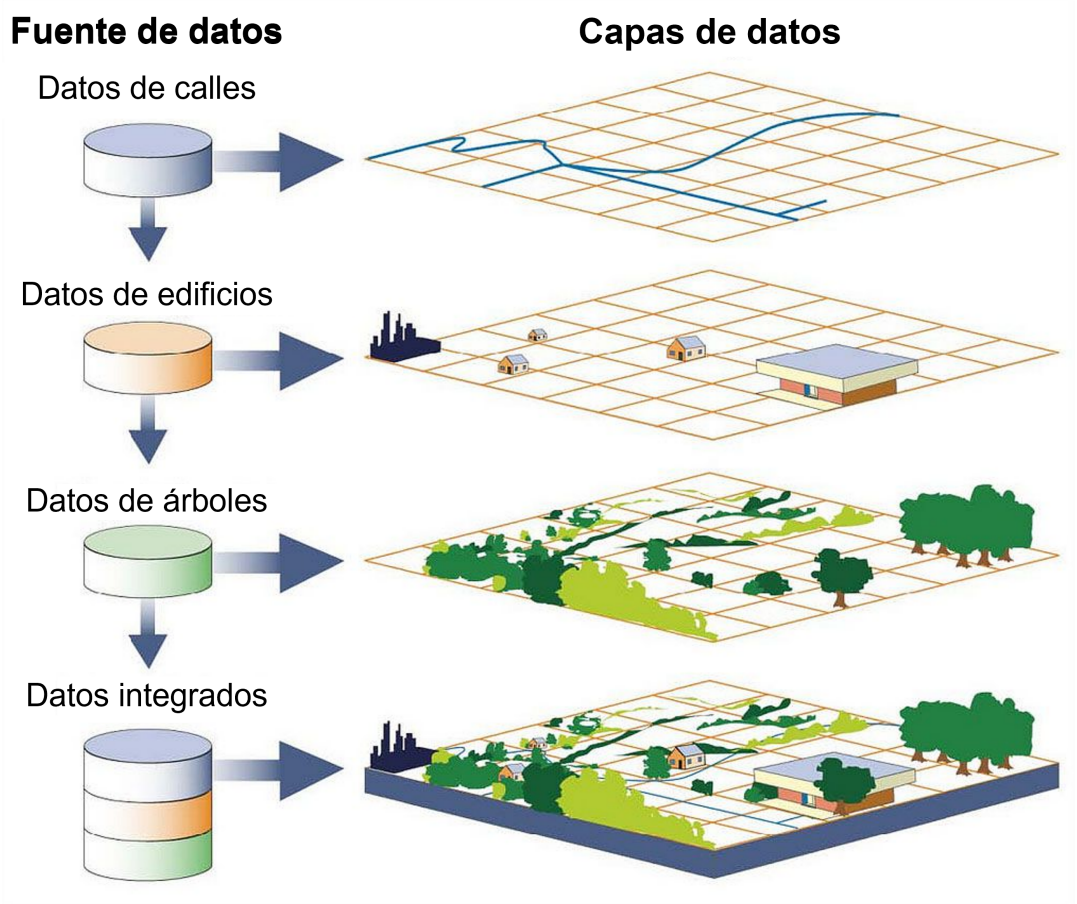


FIGURA 2.1: Capas de datos en un Sistema de Información Geográfica [17].

Esto será de gran utilidad en esta investigación, debido a que se va a recolectar la información de la ubicación, que en este caso se va a expresar en forma de coordenadas georreferenciadas (latitud y longitud), así como también obtener la dirección, distrito, provincia, región, código postal, entre otros datos geográficos mediante la geocodificación inversa.

Debido a la naturaleza del Sistema de Información Geográfica, se puede incluir datos específicos, dependiendo del área donde de haga uso de un GIS. En esta investigación, aparte de los datos geográficos mencionados anteriormente, se incluirá datos temporales (fecha y hora) de cada nuevo reporte de dengue, así como también datos adicionales como los comentarios y/o fotografía que se incluya en cada reporte.

2.1.2 Datos temporales

Según Celko [18], los datos temporales son aquellos datos que están compuestos por hora y/o fecha dentro de un conjunto de datos. Este tipo de datos son omnipresentes, debido a que 1 de cada 50 registros en una base de datos involucra algún valor de hora o fecha. En esta investigación, se va a hacer uso de este tipo de datos temporales para el registro de la hora y fecha de los nuevos reportes de dengue, con el objetivo de analizar en profundidad el avance de la enfermedad en distintos intervalos temporales.

Para ello, se usará el término *paso de tiempo* para referirse al intervalo temporal existente entre un suceso observado y otro. Por ejemplo, en esta investigación como se tienen los datos temporales exactos de cada reporte, se puede realizar un análisis de los reportes con un paso de tiempo que puede ir desde minutos u horas, hasta meses o años; mientras que, por ejemplo, no se contara con la hora del reporte, sólo se podría realizar un análisis con un paso de tiempo que va desde días o semanas en adelante.

En este sentido, Buckley [19] desarrolló tres formas de poder analizar datos temporales. Estas formas de análisis pueden ser aplicadas a los datos temporales recolectados en cada uno de los nuevos reportes de dengue. Estas formas de analizar los datos temporales se detallan a continuación.

2.1.2.1. Análisis por separado

En este análisis de datos temporales, todas las herramientas de geoprocesamiento respetan la configuración de tiempo de las capas con tiempo habilitado, por lo que solo se procesarán aquellas entidades que se encuentren temporalmente dentro de la extensión de tiempo establecida en el control deslizante de tiempo. Luego, los resultados se muestran como una sola capa en un mapa. Este análisis se muestra en la Figura 2.2.

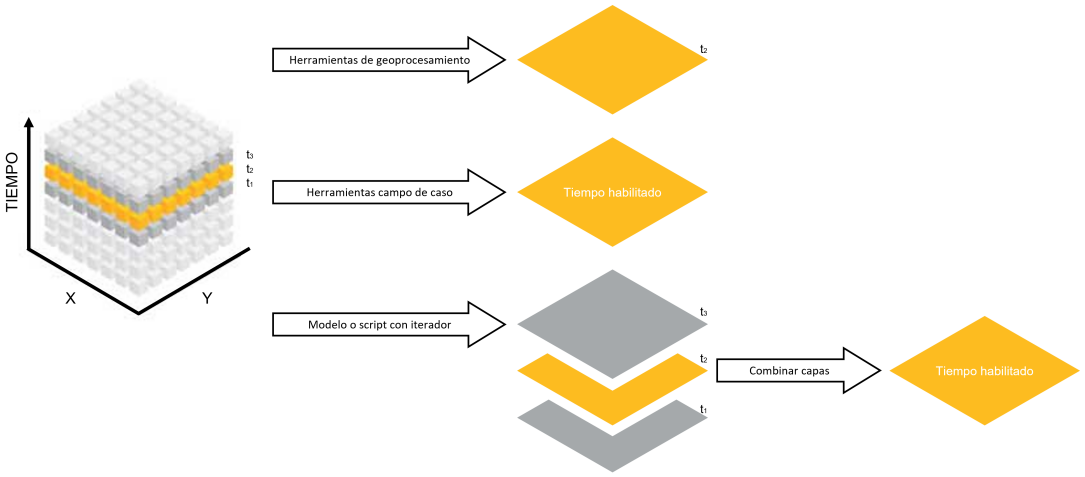


FIGURA 2.2: Análisis de datos temporales por separado [19].

2.1.2.2. Análisis con restricciones

En este análisis de datos temporales, el atributo tiempo se puede utilizar como una variable con el objetivo de repetir el análisis para cada paso de tiempo. De esta manera, el tiempo funcionaría como una variable de "enlace" para relacionar los eventos que han ocurrido en relación a las otras dos variables (coordenadas x, y). Por lo tanto, los resultados se presentan como una sola capa construida a partir de los enlaces realizados entre el atributo tiempo y las coordenadas x, y. Este análisis se muestra en la Figura 2.3.

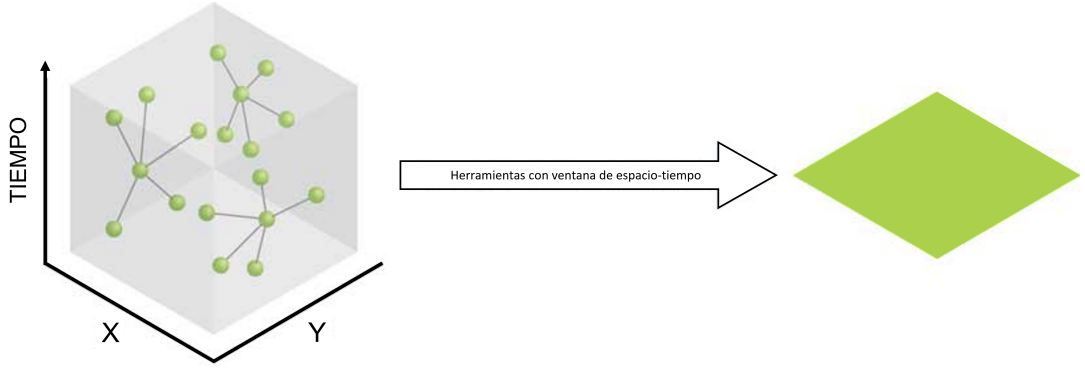


FIGURA 2.3: Análisis de datos temporales con restricciones [19].

2.1.2.3. Análisis mixto

En este análisis de datos temporales, se puede crear un modelo o script mediante un iterador para especificar que el análisis debe repetirse para cada paso de tiempo. Este análisis da como resultado salidas separadas para cada paso de tiempo. Luego, las salidas separadas se pueden combinar (por ejemplo, usando *Merge* o *Append* para clases de entidad o agregando un conjunto de datos que definen el espacio basado en celdas) para que el resultado final se pueda habilitar en el tiempo y visualizar mediante el control deslizante de tiempo. Este análisis se muestra en la Figura 2.4.

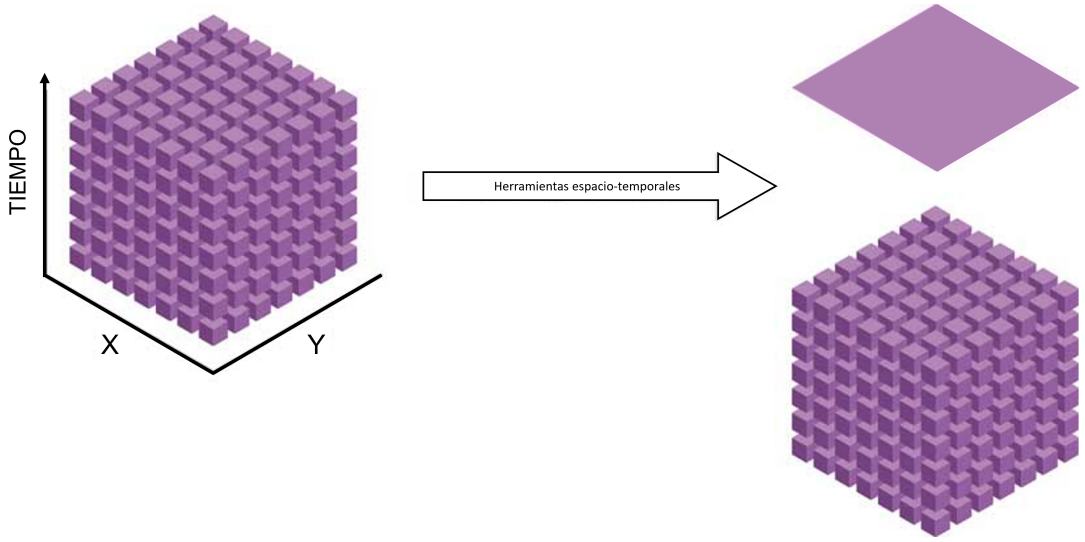


FIGURA 2.4: Análisis mixto de datos temporales [19].

En esta investigación se hará uso del primer tipo de análisis de datos temporales, el cual es el análisis por separado. Este análisis se realiza para poder visualizar los datos temporales en un mapa, y así poder realizar un análisis de los reportes de dengue en distintos intervalos temporales, denominados *ventanas de tiempo*.

2.1.3 Datos adicionales

Para complementar la información antes mencionada, en esta investigación, el registro de los nuevos reportes de dengue tendrá como datos extras la posibilidad de poder agregar un comentario y/o fotografía. Estos datos son agregados con el propósito de poder agregar información adicional en el registro de nuevos reportes de dengue. Estos datos adicionales ayudará a los especialistas en salud a poder identificar el estado y gravedad de cada caso, como una forma de complementar los datos espacio temporales obtenidos anteriormente.

2.2 Reportes en tiempo real

Según InetSoft Technology [20] los reportes en tiempo real consisten en recopilar datos actualizados a su versión más reciente y transmitir a los usuarios a medida que ocurren. De esta manera, la información se muestra en su forma más actual para que pueda ser procesada y analizada por los usuarios finales.

Sin embargo, este envío de datos actualizados siempre presentan una demora o "delay" que es el tiempo transcurrido desde que se registra los datos de un nuevo evento, hasta que esos datos son recibidos por el usuario final. Por ello, Suznjevic [9] desarrolló un resumen de los tiempos de demora en diferentes servicios que funcionan con transferencia de datos en tiempo real. Además, se mencionan las demoras tolerables en cada servicio, como comunicación por voz, con una demora menor a 150ms; videojuegos en línea, con una demora menor a 200ms; escritorios remotos, con una demora menor a 200ms; o mensajería instantánea, con una demora menor a 5s.

En este sentido, en esta investigación se mejorará el tiempo de actualización de los registros, pasando de una demora de 24 horas (tiempo actual usado) a una demora no

mayor a 10 segundos. Esto se logrará mediante la implementación de una arquitectura capaz de procesar una gran cantidad de datos en tiempo real.

Algunas de las funciones que se pueden utilizar en el manejo de reportes en tiempo real son las siguientes:

- Incorporar elementos gráficos como tablas, gráficos o mapas, así como también diferentes estilos gráficos, en los que se pueden incluir gráficos de barra, pastel, línea, dispersión, burbuja, radar, 3D, etc. Esto con el propósito de poder visualizar y analizar los datos de la mejor manera posible.
- Incorporar formatos de exportación de los datos, como PDF, Excel, CSV, etc. Esto con el propósito de poder compartir o realizar informes con los datos reportados hasta ese momento.

2.3 Visualización y seguimiento en tiempo real

Según Striim, Inc. [21], la diferencia que hace que la visualización de datos en tiempo real sea mejor que la visualización por lotes es la inmediatez en la toma de decisiones. En estas circunstancias, donde las medidas se deben tomar rápidamente, es donde la visualización de datos en tiempo real destacan, debido a que proporcionan los datos necesarios para poder tomar la mejor decisión ante un problema.

Por esta razón, el mayor valor de la visualización en tiempo real dentro del manejo de riesgos es que permite informar a las personas encargadas de tomar decisiones que necesitan actuar frente a eventos emergentes. En contraste, los análisis descriptivos entregados periódicamente hacen poco para una persona encargada de la toma de decisiones, debido a que estas situaciones son imprevistas y cambian constantemente.

En esta investigación, la inmediatez en la toma de decisiones estará enfocada en la inmediatez con la que la solución computacional propuesta pueda mostrar los datos de los

nuevos reportes que se realicen. De esta manera, las personas encargadas de las políticas públicas (realizar las fumigaciones o de aplicar algún tratamiento a las personas que ya han contraído dengue) puedan actuar con la mayor inmediatez posible y lograr contener la enfermedad en un área reducida, evitando su propagación.

2.4 Arquitectura de microservicios

Según *Microservices.io* [22], una arquitectura de microservicios es un enfoque para desarrollar una aplicación de software como un conjunto de pequeños servicios independientes, cada uno de los cuales se ejecuta en su propio proceso y se comunica con mecanismos ligeros, a menudo una API de recursos HTTP. Estos servicios se construyen a través de capacidades de negocios y se pueden implementar de manera independiente por equipos de desarrollo totalmente automatizados.

Un equipo es responsable de uno o más subdominios. Un subdominio es un modelo implementable de una porción de funcionalidad empresarial, también conocida como capacidad empresarial. Los subdominios implementan el comportamiento de la aplicación, que consta de un conjunto de operaciones (del sistema). Una operación se invoca de una de tres maneras: solicitudes sincrónicas y asincrónicas de los clientes; eventos publicados por otras aplicaciones y servicios; y el paso del tiempo. La Figura 2.5 muestra un ejemplo de una arquitectura de microservicios.

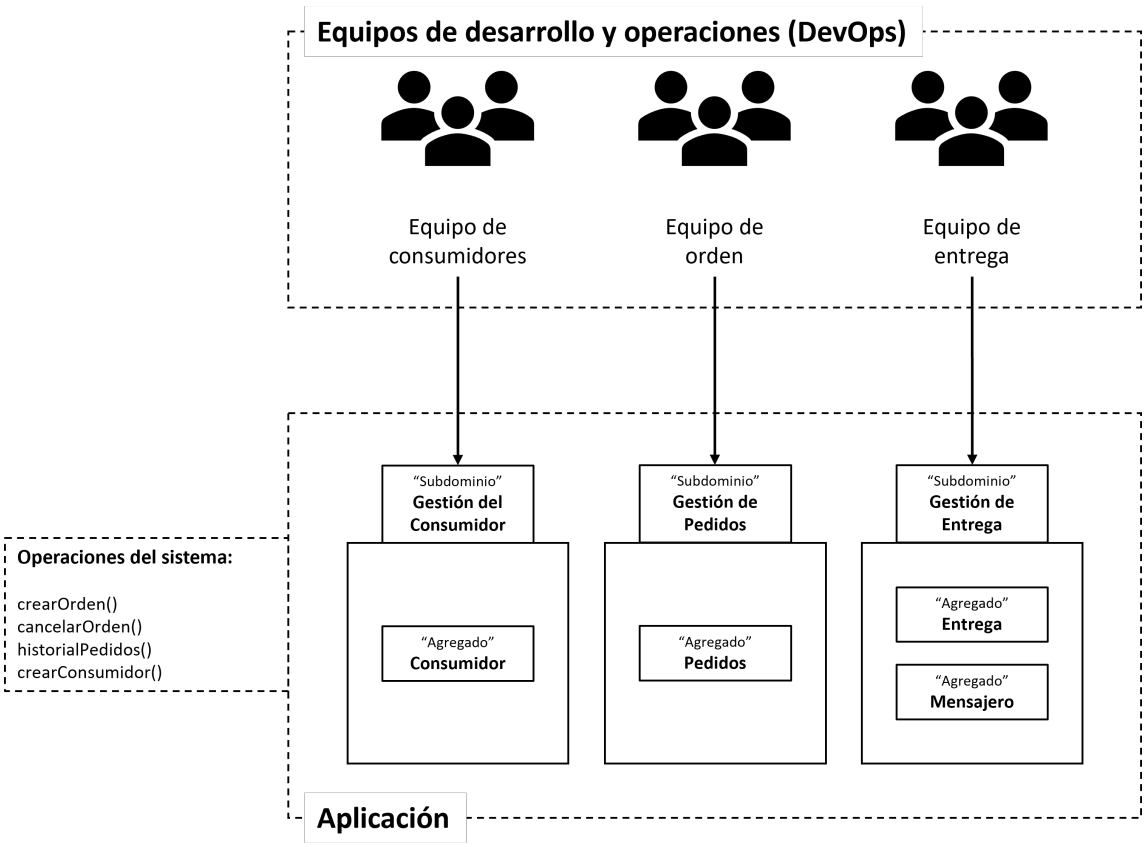


FIGURA 2.5: Ejemplo de una arquitectura de microservicios [22].

Capítulo 3

METODOLOGÍA

En este capítulo se detallará el proceso de desarrollo de esta investigación, centrada en la construcción de un sistema de monitorización de casos de dengue en la región Lima. Para ello, se ha dividido el desarrollo de esta investigación en 4 fases, las cuales se centran en la construcción de un backend en un servidor en la nube, un frontend para móviles y web y la implementación de funciones de clusterización de casos, y finalmente la realización de pruebas de escalabilidad y clusterización. Se culminará con un análisis de los resultados comparándolos con las pruebas iniciales y se llegarán a las respectivas conclusiones. El detalle de estas fases se muestran en la Figura 3.1.

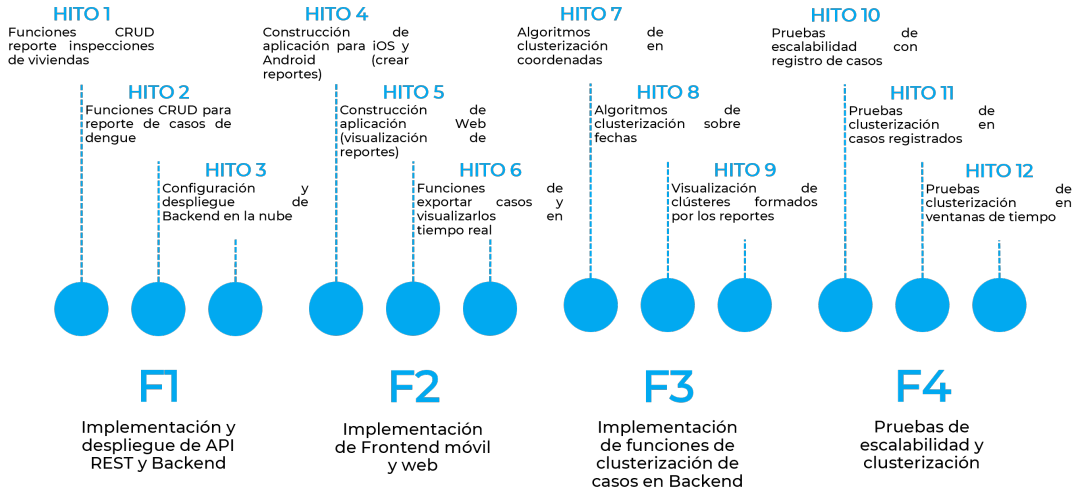


FIGURA 3.1: Diagrama de metodología de desarrollo de la investigación.

3.1 Fase 1: Implementación API REST

En esta sección se detallará la construcción, implementación y despliegue de un backend en un servidor en la nube que permitirá procesar solicitudes provenientes tanto desde la aplicación móvil como de la aplicación web. Por ello, con el motivo de que más adelante se puedan agregar más datos tanto en el reporte de nuevos casos de dengue así como también en el reporte de nuevas inspecciones de dengue, se va a hacer uso de una base de datos relacional, PostgreSQL.

Como este sistema permitirá registrar tanto inspecciones de vivienda así como también registrar casos de dengue, se va a manejar un total de 8 tablas en PostgreSQL, entre las cuales destacan dos llamadas *HomeInspection* y *CaseReport*. La primera dedicada específicamente al registro de las inspecciones de vivienda en donde se incluirán los atributos y tablas extras mostradas en la Figura 3.2. La segunda dedicada específicamente al registro de nuevos casos de dengue, en donde se incluirán los atributos y tablas extras mostradas en la Figura 3.2.

En este sentido, cada documento que se incluya en las tablas de *HomeInspection* o *CaseReport* va a hacer referencia al reporte de una inspección de vivienda o de un nuevo caso de dengue, respectivamente. Ahora que ya se tiene la estructura de la base de datos, se va a definir los *endpoints* que va a tener el backend para poder procesar las solicitudes provenientes de la aplicación móvil y la aplicación web.

Para ello, se va a hacer uso del lenguaje de programación Go para la implementación de la lógica del backend lo que incluye las operaciones CRUD para las tablas de *HomeInspection* y *CaseReport*, así como también la implementación de la clusterización detallada en la Fase 3.

En esta Fase 1, se va a describir la construcción de los *endpoints* encargados de crear, leer, editar y borrar documentos dentro de las tablas *HomeInspection* y *CaseReport*.

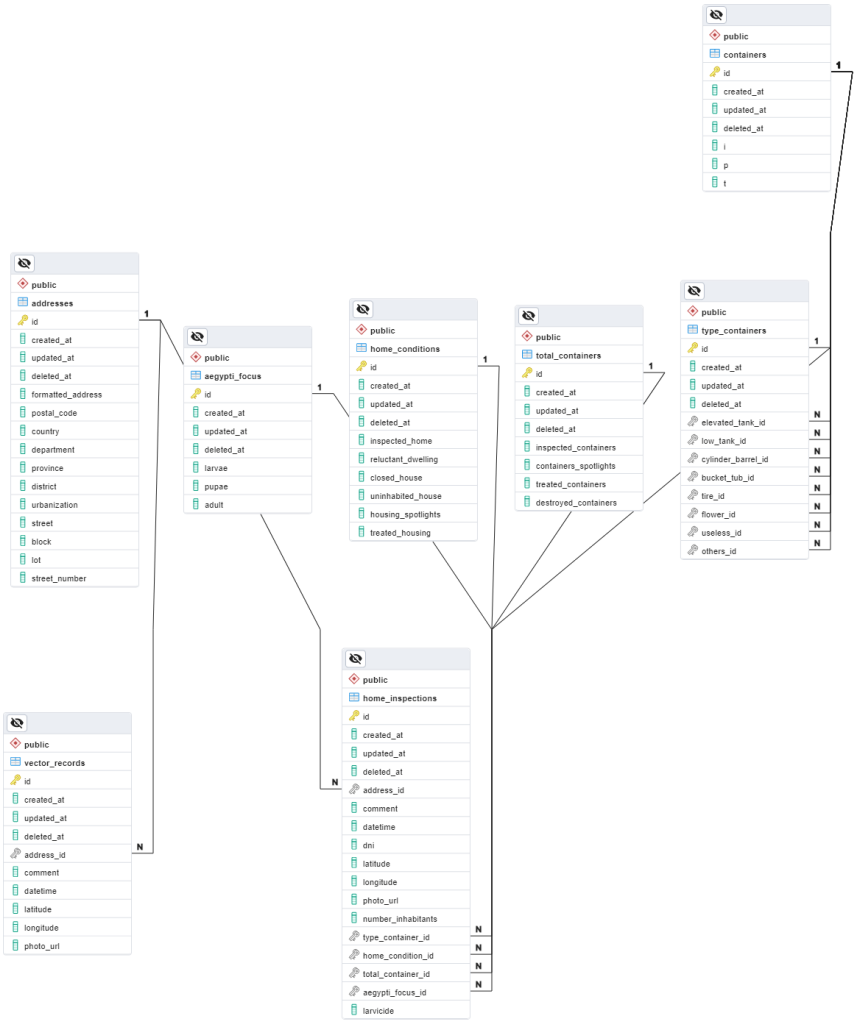


FIGURA 3.2: Diagrama de la estructura de la base de datos DENV.

Por esta razón, se va a tener un total de 7 *endpoints* para cada una de las 2 tablas principales, con el propósito de realizar las funciones antes mencionadas. Se tendrá un método *POST* para la creación de nuevos registros, 3 métodos *GET* para la lectura de registros, 2 métodos *DELETE* para la eliminación de registros y un método *PUT* para la actualización de registros.

A continuación, se muestran los diagramas de secuencia de algunos de los *endpoints* antes mencionados.

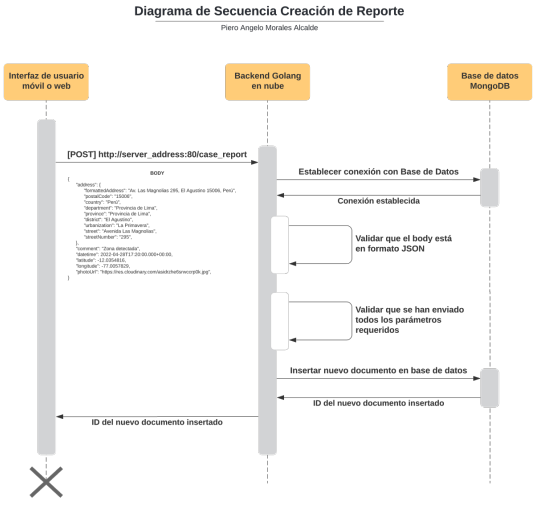


FIGURA 3.3: Diagrama de secuencia para la creación de un reporte. Fuente: Elaboración propia.

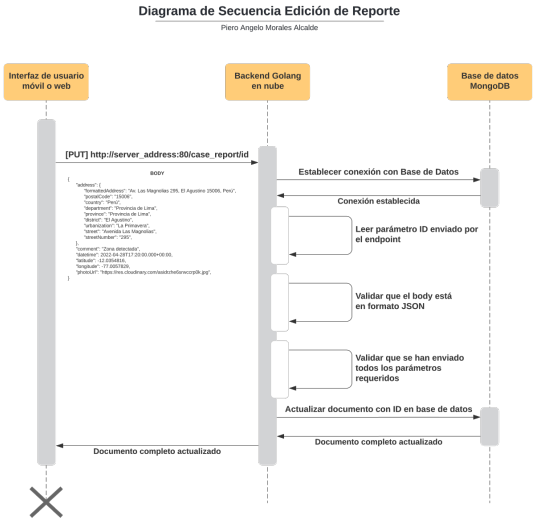


FIGURA 3.5: Diagrama de secuencia para la edición de un reporte. Fuente: Elaboración propia.

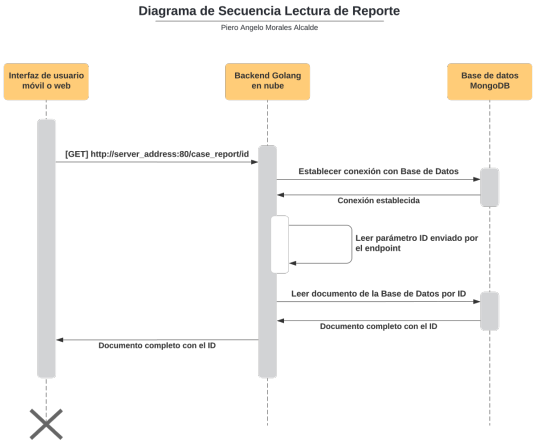


FIGURA 3.4: Diagrama de secuencia para la lectura de un reporte. Fuente: Elaboración propia.

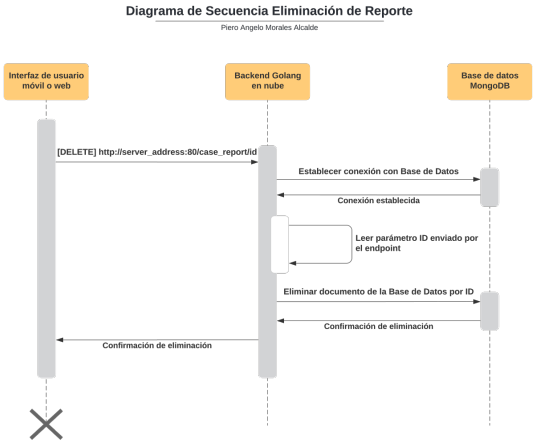


FIGURA 3.6: Diagrama de secuencia para la eliminación de un reporte. Fuente: Elaboración propia.

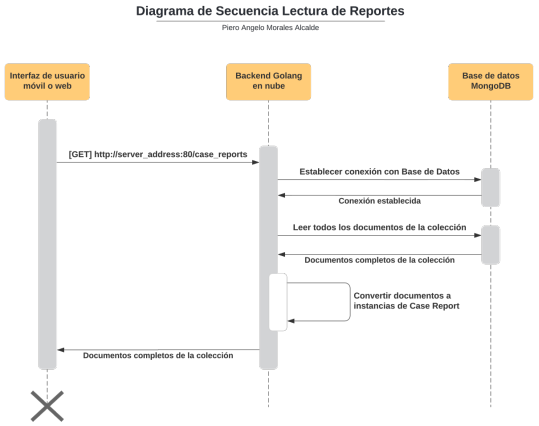


FIGURA 3.7: Diagrama de secuencia para la lectura de todos los reportes. Fuente: Elaboración propia.

Es necesario resaltar que los diagramas de secuencia descritos anteriormente son utilizados tanto para las tablas de *HomeInspection* y *CaseReport*, presentando sólo pequeñas variaciones en ciertas partes, pero con la misma funcionalidad. Por ello, sólo se diferencian en los *endpoints* y en la tabla de la base de datos sobre la que se hace la lectura y/o escritura.

3.1.1 Protección de datos personales

Para la protección de datos personales, se va a hacer uso de un *middleware* que se encargará de verificar si la solicitud que se está realizando tiene un *token* de autenticación válido. Para ello, se va a hacer uso de la librería *go-chi/jwtauth*[23] que permite la generación y validación de *tokens* de autenticación.

Este *middleware* se va a incorporar en todos los *endpoints* que se encuentren en el grupo de *endpoints* que se encargan de realizar las operaciones CRUD sobre las tablas *HomeInspection* y *CaseReport*. Además, se va a hacer uso de la librería *go-cors*[24] que permite la configuración de los *headers* de las solicitudes para permitir el acceso a los *endpoints* desde cualquier origen.

3.1.2 Seguridad en la transferencia de datos

Dado que el sistema va a manejar datos sensibles como la ubicación de los usuarios, se va a hacer uso de un certificado SSL para la transferencia de datos entre el servidor y el cliente. Para ello, la máquina virtual en la nube va a tener instalado el certificado SSL y se va a configurar el servidor para que acepte solicitudes HTTPS.

Por otra parte, se va a usar HS256 como algoritmo de cifrado para la generación de los *tokens* de autenticación. Este algoritmo de cifrado es uno de los más utilizados para la generación de *tokens* de autenticación, y es soportado por la librería *go-chi/jwtauth*[23]. De esta manera, se garantiza que los *tokens* de autenticación no puedan ser modificados por terceros y se garantiza la seguridad de los datos de los usuarios.

3.1.3 OWASP Top 10

Para la inclusión de OWASP Top 10, se ha realizado un análisis de las vulnerabilidades más comunes en aplicaciones web y se ha determinado que las vulnerabilidades más comunes son las siguientes:

- ***A1:2017-Injection:*** En esta vulnerabilidad se incluyen las inyecciones de código SQL, NoSQL, OS y LDAP. Para esta investigación, se ha determinado que la vulnerabilidad más común es la inyección de código SQL, por ello se ha realizado un análisis de las librerías que se van a utilizar para la conexión con la base de datos y se ha determinado que la librería *gorm.io/gorm*[25] es la más segura para la conexión con la base de datos PostgreSQL.
- ***A2:2017-Broken Authentication:*** En esta vulnerabilidad se incluyen las vulnerabilidades de autenticación y gestión de sesiones. Para esta investigación, se ha determinado que la vulnerabilidad más común es la gestión de sesiones, por ello se ha realizado un análisis de las librerías que se van a utilizar para la generación

y validación de *tokens* de autenticación y se ha determinado que la librería *go-chi/jwtauth*[23] es la más segura para la generación y validación de *tokens* de autenticación. De esta manera, cada vez que se cargue la página web o se abra la aplicación móvil, se generará un *token* de autenticación que tendrá una duración de 24 horas, y este *token* de autenticación se enviará en cada solicitud que se realice al *backend*. De esta manera, se garantiza que sólo los usuarios de la aplicación web y móvil podrán acceder a los *endpoints* que se encargan de realizar las operaciones CRUD sobre las tablas *HomeInspection* y *CaseReport*.

- ***A3:2017-Sensitive Data Exposure:*** En esta vulnerabilidad se incluyen las vulnerabilidades de exposición de datos sensibles. Para esta investigación, se ha determinado que la vulnerabilidad más común es la exposición de datos sensibles, por ello se ha realizado un análisis de las librerías que se van a utilizar para la generación y validación de *tokens* de autenticación y se ha determinado que la librería *go-chi/jwtauth*[23] es la más segura para la generación y validación de *tokens* de autenticación.

- ***A4:2017-XML External Entities (XXE):*** En esta vulnerabilidad se incluyen las vulnerabilidades de entidades externas XML. Para esta investigación, se ha determinado que esta vulnerabilidad no aplicaría para este sistema, debido a que no se va a hacer uso de archivos XML.

- ***A5:2017-Broken Access Control:*** En esta vulnerabilidad se incluyen las vulnerabilidades de control de acceso. Para esta investigación, se ha determinado que la vulnerabilidad más común es la falta de control de acceso, por ello se ha realizado un análisis de las librerías que se van a utilizar para la generación y validación de *tokens* de autenticación y se ha determinado que la librería *go-chi/jwtauth*[23] es la más segura para la generación y validación de *tokens* de autenticación. De esta manera sólo los usuarios de la aplicación web y móvil podrán acceder a los *endpoints*

que se encargan de realizar las operaciones CRUD sobre las tablas *HomeInspection* y *CaseReport*.

- ***A6:2017-Security Misconfiguration:*** En esta vulnerabilidad se incluyen las vulnerabilidades de configuración de seguridad. Para esta investigación, se ha determinado que la vulnerabilidad más común es la configuración de seguridad, por ello se ha realizado un análisis de las librerías que se van a utilizar para la generación y validación de *tokens* de autenticación y se ha determinado que la librería *go-chi/jwtauth*[23] es la más segura para la generación y validación de *tokens* de autenticación.

- ***A7:2017-Cross-Site Scripting (XSS):*** En esta vulnerabilidad se incluyen las vulnerabilidades de secuencias de comandos entre sitios. Para esta investigación, se ha determinado que tanto por parte de la aplicación web como de la aplicación móvil, no se va a permitir la entrada de código HTML, por lo que no se va a permitir la entrada de código JavaScript. Por ello, se ha determinado que esta vulnerabilidad no aplicaría para este sistema.

- ***A8:2017-Insecure Deserialization:*** En esta vulnerabilidad se incluyen las vulnerabilidades de deserialización insegura. Para esta investigación, se ha determinado que esta vulnerabilidad no aplicaría para este sistema, debido a que no se va a hacer uso de deserialización de objetos.

- ***A9:2017-Using Components with Known Vulnerabilities:*** En esta vulnerabilidad se incluyen las vulnerabilidades de uso de componentes con vulnerabilidades conocidas. Para esta investigación, se ha determinado que la vulnerabilidad más común es el uso de componentes con vulnerabilidades conocidas, por ello se ha realizado un análisis de las librerías que se van a utilizar para la conexión con la base de datos y se ha determinado que la librería *gorm.io/gorm*[25] es la más segura para la conexión con la base de datos PostgreSQL.

- ***A10:2017-Insufficient Logging & Monitoring:*** En esta vulnerabilidad se incluyen las vulnerabilidades de registro y monitoreo insuficiente. Para esta investigación, se ha determinado que la vulnerabilidad más común es el registro y monitoreo insuficiente, por ello se ha realizado un análisis de las librerías que se van a utilizar para la generación logs y se ha determinado que la librería *go-chi/middleware*[26] es la más segura para la generación de logs y monitoreo de solicitudes.

3.2 Fase 2: Implementación de interfaz móvil y web

En esta Fase 2 se realizará la implementación de la interfaz tanto móvil como web con el objetivo de realizar el registro de inspecciones de vivienda y nuevos casos de dengue, así como también de poder visualizar y/o exportar dichos datos. Por ello, es necesario destacar las funciones que van a cumplir cada una de estas dos interfaces que se van a construir. Estas funciones se describen a continuación.

3.2.1 Interfaz móvil

La interfaz móvil será la encargada de poder realizar nuevas inspecciones de vivienda, así como también registrar nuevos casos de dengue. Además, permitirá visualizar las inspecciones de vivienda y casos de dengue que ya se encuentren registrados en la base de datos mediante el backend construido en la Fase 1. Para ello, la aplicación móvil contará con una ventana inicial que permitirá al usuario elegir entre dos opciones, la primera enfocada en registrar una nueva inspección de vivienda y la segunda encargada de reportar un nuevo caso de dengue, ambos haciendo uso del backend de la Fase 1.

Al entrar en cualquiera de las dos opciones se mostrará una nueva vista con varios campos para llenar, dependiendo del tipo de registro. La ubicación en ambas opciones se registrará automáticamente con la ubicación actual del usuario, así como también los

campos de hora y fecha se registrarán automáticamente. Además, se le permitirá al usuario ingresar algún comentario o añadir una fotografía adjunta al registro (obtenida de la galería o tomada en ese momento usando la cámara). Una vez que el usuario termina de llenar los datos solicitados, la aplicación enviará una petición al *backend* para que se guarde el nuevo registro. El diseño de estas vistas se muestran en la Figura 3.8, Figura 3.9 y Figura 3.10.

FIGURA 3.8: Diseño de vista selección de tipo de registro en la interfaz móvil.

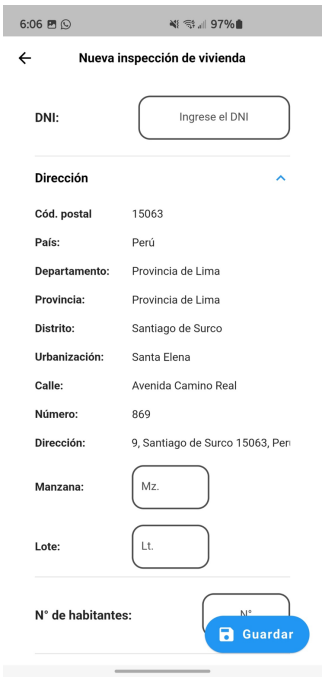


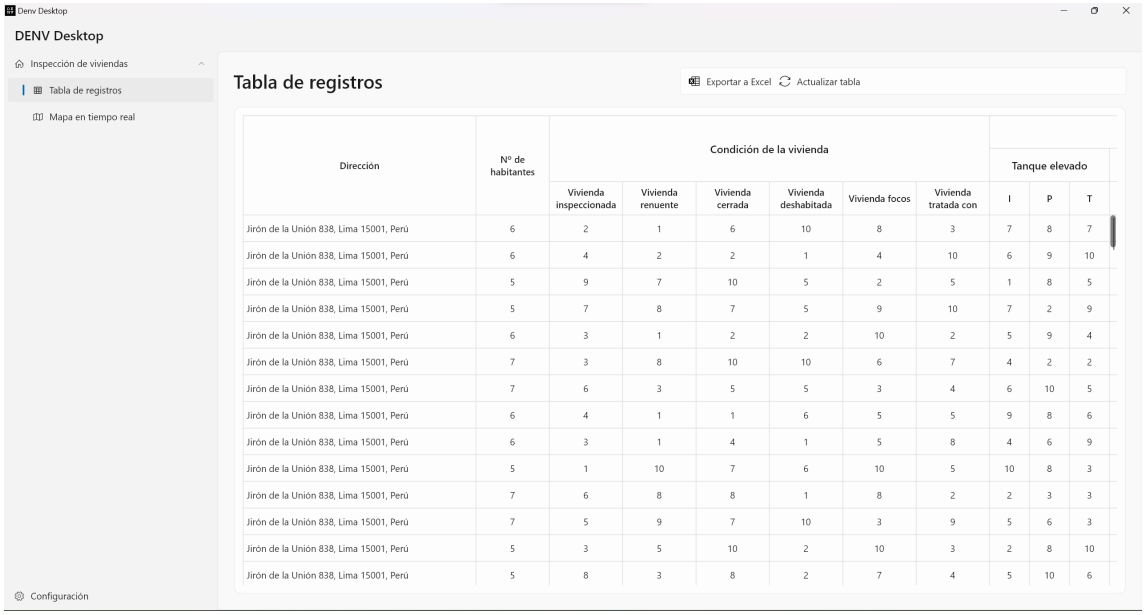
FIGURA 3.9: Diseño de vista formulario de inspección de vivienda en la interfaz móvil.

FIGURA 3.10: Diseño de vista formulario de caso de dengue en la interfaz móvil.

3.2.2 Interfaz web

La interfaz web tiene el objetivo de poder visualizar en tiempo real los nuevos casos de dengue e inspección de vivienda que se reporten mediante la aplicación móvil. Esto incluye la posibilidad de mostrar los registros, tanto en forma de tabla, como poder visualizarlos de forma gráfica en un mapa la ubicación de cada registro. Por ello, en esta Fase 2 se realizará la implementación de dos vistas, la primera centrada en poder visualizar en forma de tabla los reportes realizados, y la segunda encargada de mostrar los mismos reportes pero de forma gráfica en un mapa.

En la primera vista, se mostrará una tabla de los reportes de casos e inspección de viviendas que han sido registrados hasta ese momento en la base de datos. Además también se permitirá tener la posibilidad de editar algunos atributos incluidos en cada reporte Y guardarlos directamente en la base de datos usando el *endpoint PUT* implementado en la Fase 1. Por último en esta vista también se tendrá la posibilidad de poder exportar los registros mostrados en formato Excel con el objetivo de poder ser usado con otras herramientas para análisis más específicos. El diseño de esta vista se muestra en la Figura 3.11.



Dirección	Nº de habitantes	Condición de la vivienda						Tanque elevado		
		Vivienda inspeccionada	Vivienda renuente	Vivienda cerrada	Vivienda deshabitada	Vivienda focos	Vivienda tratada con	I	P	T
Jirón de la Unión 838, Lima 15001, Perú	6	2	1	6	10	8	3	7	8	7
Jirón de la Unión 838, Lima 15001, Perú	6	4	2	2	1	4	10	6	9	10
Jirón de la Unión 838, Lima 15001, Perú	5	9	7	10	5	2	5	1	8	5
Jirón de la Unión 838, Lima 15001, Perú	5	7	8	7	5	9	10	7	2	9
Jirón de la Unión 838, Lima 15001, Perú	6	3	1	2	2	10	2	5	9	4
Jirón de la Unión 838, Lima 15001, Perú	7	3	8	10	10	6	7	4	2	2
Jirón de la Unión 838, Lima 15001, Perú	7	6	3	5	5	3	4	6	10	5
Jirón de la Unión 838, Lima 15001, Perú	6	4	1	1	6	5	5	9	8	6
Jirón de la Unión 838, Lima 15001, Perú	6	3	1	4	1	5	8	4	6	9
Jirón de la Unión 838, Lima 15001, Perú	5	1	10	7	6	10	5	10	8	3
Jirón de la Unión 838, Lima 15001, Perú	7	6	8	8	1	8	2	2	3	3
Jirón de la Unión 838, Lima 15001, Perú	7	5	9	7	10	3	9	5	6	3
Jirón de la Unión 838, Lima 15001, Perú	5	3	5	10	2	10	3	2	8	10
Jirón de la Unión 838, Lima 15001, Perú	5	8	3	8	2	7	4	5	10	6

FIGURA 3.11: Vista de tabla de registros en la interfaz web.

En la segunda vista, se mostrará en un mapa los casos que estén siendo registrados en la base de datos, con el objetivo de tener la información en el mapa siempre actualizada a la versión más reciente. En esta vista también se podrá ver los detalles de cada registro en la misma vista del mapa teniendo la posibilidad de poder visualizar la imagen adjunta en caso se hubiera adjuntado una imagen, y todos los campos correspondiestes al registro mostrados en la Figura 3.2. El diseño de esta vista se muestra en la Figura 3.12.

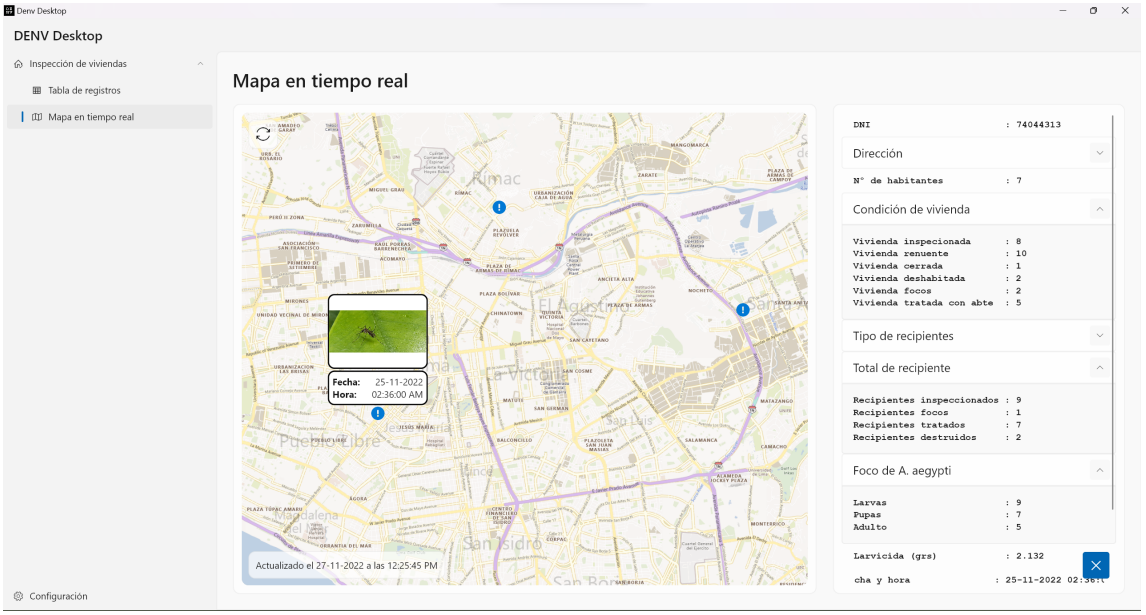


FIGURA 3.12: Vista de mapa en tiempo real de registros en la interfaz web.

3.3 Fase 3: Implementación de función de clusterización

En esta Fase 3 se implementará en el *backend* la función de clusterización de los registros con el objetivo de mostrar los puntos de infección basados en la ubicación de los registros que se encuentren en la base de datos. Cabe destacar que estas funciones de visualización de datos solamente serán utilizados por la interfaz web, debido a que cuenta con un área de visualización mucho más grande en comparación con un dispositivo móvil.

Además, el objetivo principal de estas funciones de clusterización es hacer una prueba exigente al *backend* para poder determinar el número de solicitudes que puede manejar el *backend*, la latencia de las solicitudes y el tiempo de ejecución de la clusterización. Por ello, se realizarán pruebas de escalabilidad y clusterización para poder determinar el número de solicitudes que puede manejar el *backend*, la latencia de las solicitudes y el tiempo de ejecución de la clusterización. Además, se realizarán pruebas de

clusterización incrementando la cantidad de registros en la base de datos para poder determinar el número de clusters que se pueden obtener con la cantidad de registros que se tienen en la base de datos.

Para la implementación de la clusterización se va a usar el algoritmo *BD-Scan*, ya que tiene la posiblidad de eliminar registros aislados y poder obtener zonas de infección más precisas. La implementación de clusterización será una modificación de la librería *go-DBSCAN* [27]. Se ha elegido esta librería porque cada punto puede tener n dimensiones, lo que permite que cada punto tenga una ubicación en el mapa, y además se puede definir un valor para la fecha y hora. Además, esta librería esta implementada para ejecutarse en paralelo, lo que permite que la clusterización se pueda realizar en un tiempo menor y aprovechar al máximo el poder de cómputo de la máquina donde se ejecuta el *backend*.

3.3.1 Clusterización sobre coordenadas

En una primera etapa, se modificará la librería *go-DBSCAN* para que pueda realizar la clusterización sobre las coordenadas de los registros. Para ello, se agregará el atributo para guardar el ID del registro junto con las coordenadas del registro (para este caso, sólo se usaran dos dimensiones para las coordenadas, correspondientes a la latitud y longitud). Por la naturaleza del algoritmo DB-Scan, se debe definir un valor para el radio de la esfera (en este caso círculo por tratarse sólo de dos dimensiones) y el número mínimo de puntos que debe tener una esfera para ser considerada como un cluster. Por ello, se ha preparado el endpoint encargado de calcular los clusters de los registros, y se le pasa como parámetro el radio de la esfera y el número mínimo de puntos para una esfera.

3.3.2 Clusterización sobre fechas

En una segunda etapa, se modificará la librería *go-DBSCAN* para que se pueda realizar la clusterización sobre las fechas de los registros aparte de las coordenadas. Para

ello, aparte de los atributos para guardar el ID del registro y las coordenadas del registro, se agregará un atributo para guardar la fecha y hora del registro. Además, el endpoint ahora recibe como parámetro la fecha de inicio y fin para poder realizar la clusterización sobre los registros que se encuentren en ese rango de fechas.

3.3.3 Visualización de clusterización

En una tercera etapa, se implementará la visualización de los clusters en la interfaz web. Para ello, la aplicación web debe tener la posibilidad de enviar el radio de la esfera y el número mínimo de puntos que debe tener una esfera para ser considerada como un cluster, y la fecha de inicio y fin para poder realizar la clusterización sobre los registros que se encuentren en ese rango de fechas. La aplicación web debe tener la posibilidad de poder visualizar los clusters en el mapa, y al hacer click sobre un cluster, se debe mostrar los detalles de los registros que se encuentran en ese cluster.

3.4 Fase 4: Pruebas de escalabilidad y clusterización

Para esta Fase 4 se realizarán pruebas de escalabilidad y clusterización para poder determinar el número de solicitudes que puede manejar el *backend*, la latencia de las solicitudes y el tiempo de ejecución de la clusterización. Además, se realizarán pruebas de clusterización incrementando la cantidad de registros en la base de datos para poder determinar el número de clusters que se pueden obtener con la cantidad de registros que se tienen en la base de datos.

3.4.1 Pruebas de escalabilidad

Para las pruebas de escalabilidad se realizarán pruebas de carga con el objetivo de determinar el número de casos que puede registrar el *backend* y la latencia de las solicitudes. Para ello, se utilizará la librería *Locust* [28], la cual permite realizar pruebas de estrés de forma sencilla y rápida utilizando *Python*. En este sentido, se realizará una comparación entre tres arquitecturas de *backend*, la arquitectura inicial (monolítica), la primera arquitectura propuesta (microservicios) y la segunda arquitectura propuesta (híbrida microservicios y broker Kafka). Para ello, se realizarán pruebas de carga con la arquitectura monolítica, la arquitectura de microservicios y la arquitectura híbrida; y se compararán los resultados obtenidos con dos variables: la latencia de las solicitudes y la disponibilidad.

3.4.2 Pruebas de clusterización sobre coordenadas

Para la clusterización, se realizarán pruebas incrementando la cantidad de registros en la base de datos para poder determinar el número de clusters que se pueden obtener con la cantidad de registros que se tienen en la base de datos. Para ello, se ha escrito un programa en Python que permite generar registros aleatorios con coordenadas y fechas aleatorias que permite generar registros aleatorios y guardarlos en la base de datos. Además, se ha creado otro programa en Python que permite realizar la clusterización sobre los registros que se encuentren en la base de datos y que permite realizar la clusterización sobre los registros que se encuentren en la base de datos. Este programa permite ver la cantidad de clústeres formados, los IDs de los registros que se encuentran en cada cluster, y la cantidad de registros que se encuentran en cada cluster.

3.4.3 Pruebas de clusterización sobre fechas

En estas pruebas, se utilizará el mismo programa que se utilizó para las pruebas de clusterización sobre coordenadas, pero se modificará algunas características para que se pueda realizar la clusterización sobre las fechas de los registros. Para ello, se realizará la clusterización sobre diferentes rangos de tiempo, y se determinará el número de clusters que se pueden obtener con la cantidad de registros que se tienen en la base de datos. Por último, este programa permitirá visualizar en 3D los clusters obtenidos, dos dimensiones para las coordenadas y una tercera dimensión para las fechas.

Capítulo 4

RESULTADOS Y DISCUSIÓN

4.1 Sistemas actuales

Para poder comparar la aplicación desarrollada con los sistemas actuales, se ha realizado una investigación sobre los sistemas actuales de información con los que cuenta el Ministerio de Salud del Perú (MINSA). Para ello, se ha realizado una búsqueda en el portal de datos abiertos de la Defensoría del Pueblo del Perú [29] en donde se menciona que se cuenta con un total de 838 establecimientos de salud en Lima.

En la Sala Situacional de Dengue del MINSA [2], se puede observar que los datos de los nuevos casos de dengue se actualizan cada semana. Estas semanas se denominan como semanas epidemiológicas, y se definen como un periodo de 7 días que inicia el domingo y termina el sábado.

Por lo tanto, suponiendo que la carga de la datos de los nuevos casos de dengue se realiza el mismo día en todos los establecimientos de salud, el sistema actual de información cuenta con un soporte de carga de 838 solicitudes por día. Por ello, como el objetivo de esta investigación es desarrollar una aplicación que soporte 5,538 solicitudes por día, se puede concluir que la aplicación desarrollada es capaz de soportar una carga de datos mayor que la carga de datos actual.

4.2 Pruebas de escalabilidad

Para probar la escalabilidad de la aplicación se realizaron pruebas de carga usando la librería *Locust* sobre la aplicación. Las pruebas se realizaron sobre tres arquitecturas

diferentes, la arquitectura monolítica, la arquitectura de microservicios y la arquitectura híbrida (microservicios y broker Kafka).

Las estructuras lógicas de las arquitecturas: monolítica, microservicios e híbrida se muestran en las Figuras 4.1, 4.3 y 4.5 respectivamente. Las estructuras físicas de las arquitecturas: monolítica, microservicios e híbrida se muestran en las Figuras 4.2, 4.4 y 4.6 respectivamente.

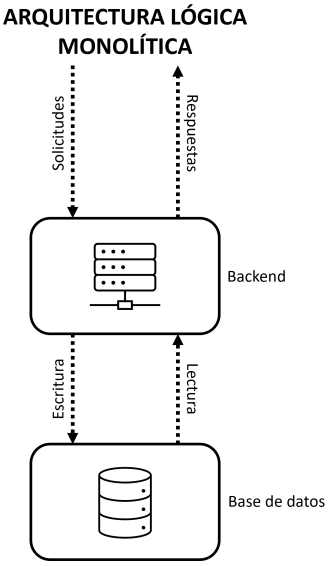


FIGURA 4.1: Estructura lógica de la arquitectura monolítica.

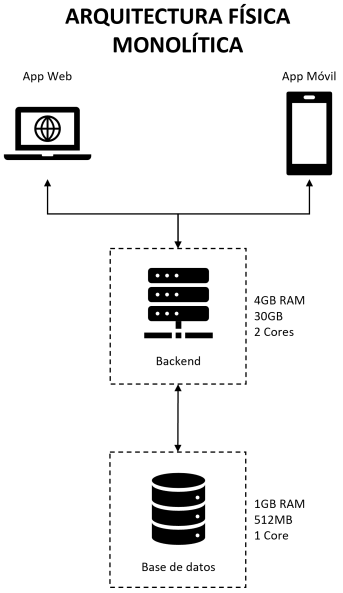


FIGURA 4.2: Estructura física de la arquitectura monolítica.

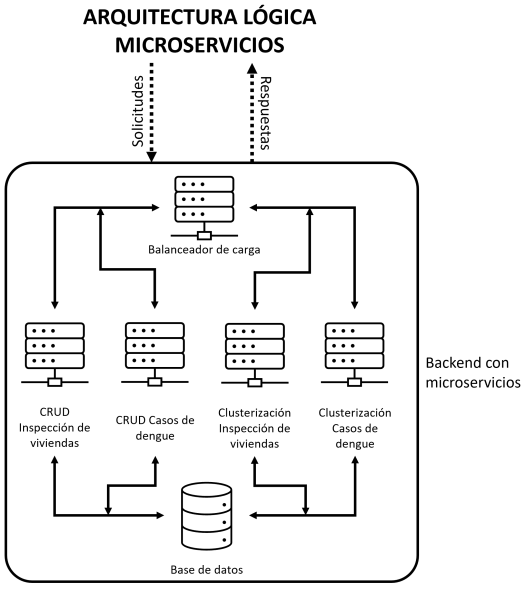


FIGURA 4.3: Estructura lógica de la arquitectura de microservicios.

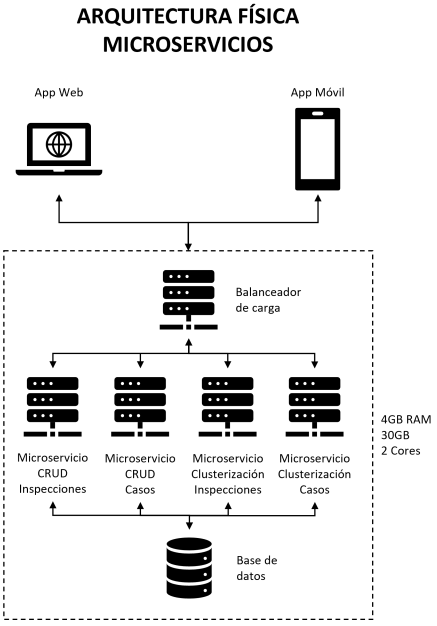


FIGURA 4.4: Estructura física de la arquitectura de microservicios.

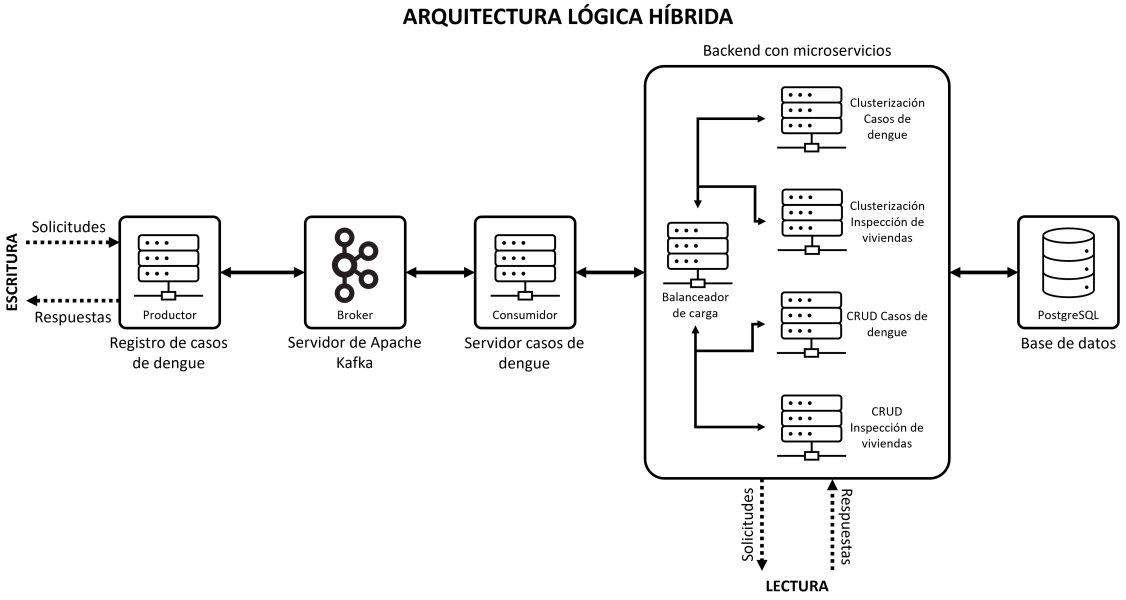


FIGURA 4.5: Estructura lógica de la arquitectura híbrida.

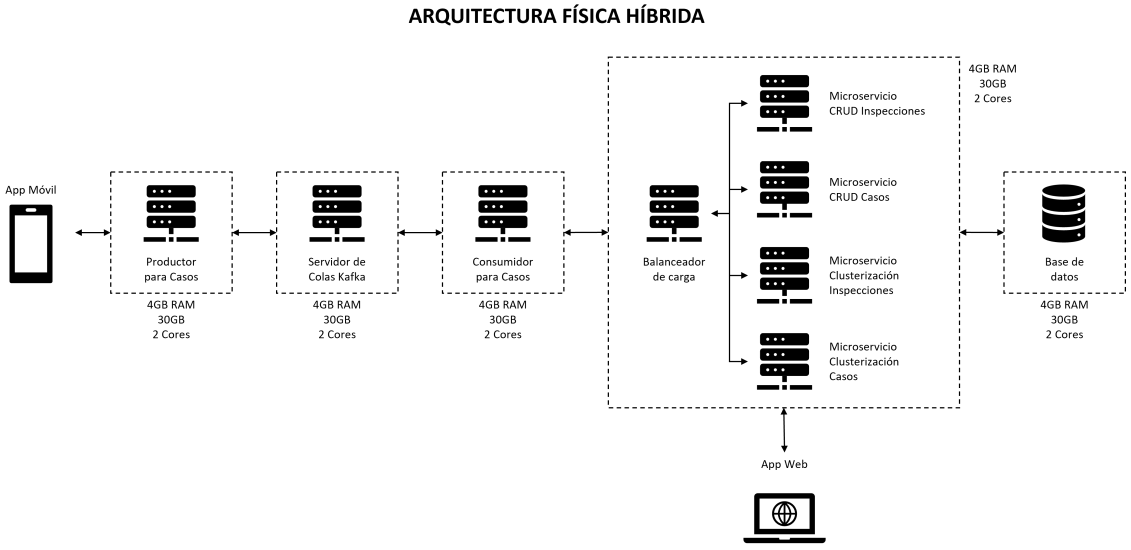


FIGURA 4.6: Estructura física de la arquitectura híbrida.

Se realizaron pruebas de carga con 100, 200, 400, 800, 1,600, 2,500, 2,800, 3,000 y 3,200 solicitudes por segundo para la arquitectura monolítica y la arquitectura de microservicios. Esto debido a que al llegar a 400 solicitudes por segundo, la disponibilidad de la arquitectura monolítica empeza a disminuir, y al llegar a 3,000 solicitudes por segundo, la disponibilidad de la arquitectura de microservicios empieza a disminuir. Por ello, se incrementó la tasa a 3,500, 3,800 y 4,000 solicitudes por segundo para la arquitectura híbrida, ya que esta arquitectura es una modificación de la arquitectura de microservicios, y se espera que la disponibilidad sea mayor que la arquitectura de microservicios. La aplicación se encuentra desplegada en una o varias máquinas virtuales (dependiendo de la arquitectura) de Google Cloud Platform (GCP). Cada una de las máquinas virtuales cuentan con 1 núcleo (2 threads) y 4 GB de RAM en la región de Oregón, Estados Unidos. En el caso de la arquitectura híbrida, todas las máquinas virtuales se encuentran en la misma red virtual de GCP.

La primera prueba se realizó sobre la arquitectura monolítica, la cual se encuentra desplegada en una máquina virtual de GCP. Se obtuvieron los resultados de latencia y disponibilidad mostrados en las Figuras 4.7 y 4.8.

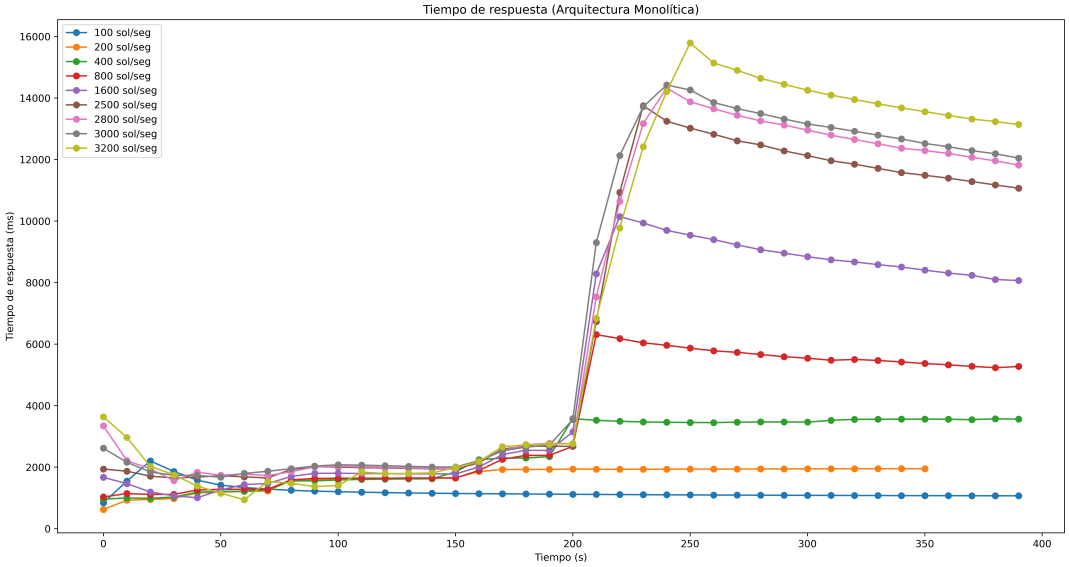


FIGURA 4.7: Tiempo de respuesta de solicitudes sobre la arquitectura monolítica.

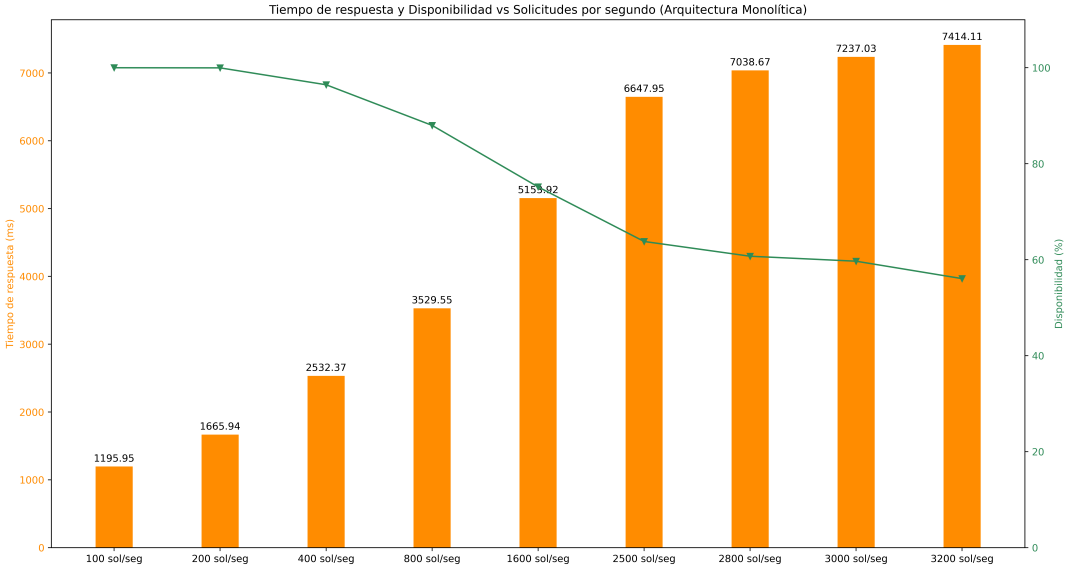


FIGURA 4.8: Tiempo de respuesta y disponibilidad sobre la arquitectura monolítica.

En la Figura 4.7 se puede observar que la latencia de las solicitudes aumenta a medida que aumenta la tasa de solicitudes por segundo. Además, se puede observar que

para 100 y 200 solicitudes por segundo, el tiempo de respuesta se mantiene estable con un máximo de 2 segundos durante los 400 segundos de prueba. Sin embargo, a partir de 400 solicitudes por segundo, el tiempo de respuesta empieza a aumentar incluso hasta 14 segundos a partir de los 200 segundos de prueba. Y para el caso de 2,800, 3,000 y 3,200 solicitudes por segundo, el tiempo de respuesta llega a alcanzar los 16 segundos a partir de los 200 segundos de prueba.

Por otra parte, en la Figura 4.8 se puede observar que la disponibilidad del servidor disminuye a medida que aumenta el número de solicitudes por segundo. Además, se puede observar que para 100 y 200 solicitudes por segundo, la disponibilidad del servidor se mantiene estable en 100 % durante toda la prueba. Sin embargo, a partir de 400 solicitudes por segundo, la disponibilidad del servidor empieza a disminuir, hasta estar por debajo del 60 % para 3,200 solicitudes por segundo.

Esto se debe a que la arquitectura inicial no está preparada para soportar un alto número de solicitudes por segundo. Por ello, se identificó el cuello de botella de la arquitectura inicial, que es durante la comunicación entre el servidor y la base de datos, como se muestra en la Figura 4.9.

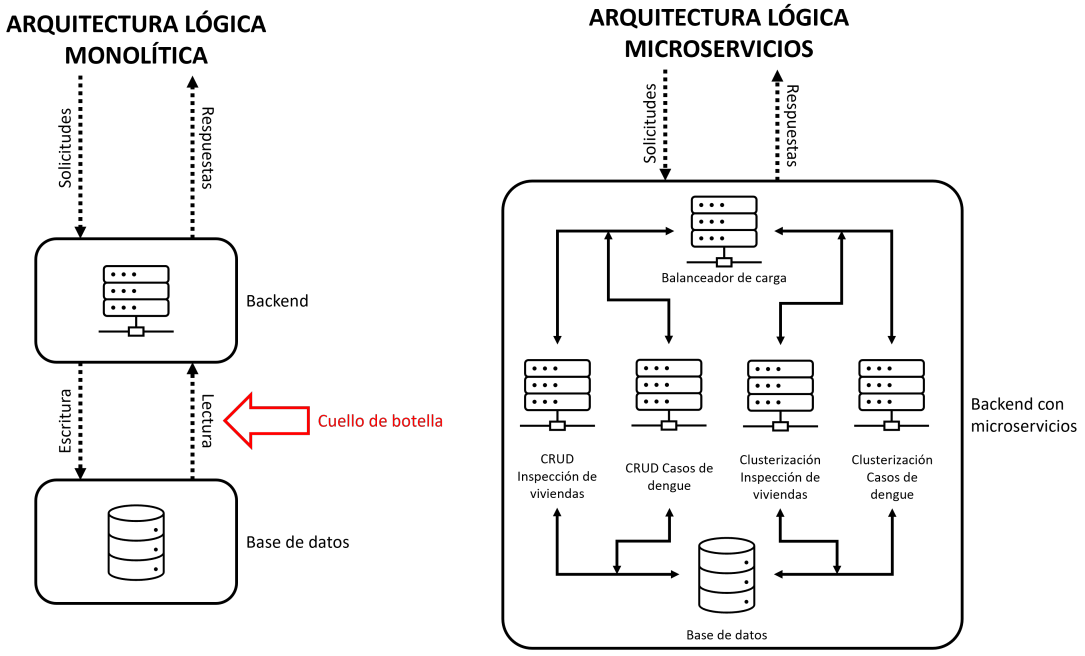


FIGURA 4.9: Estructura interna de la arquitectura monolítica y la arquitectura de microservicios identificando el cuello de botella.

Por lo tanto, se propuso la arquitectura de microservicios para mejorar la escalabilidad de la aplicación. Además, se detectó que la arquitectura inicial no aprovecha los recursos de la máquina virtual de GCP, por lo que se propuso la arquitectura de microservicios para mejorar la eficiencia de la aplicación y así aprovechar de mejor manera los recursos de la máquina virtual.

La segunda prueba se realizó sobre la arquitectura de microservicios, la cual se encuentra desplegada en la máquina virtual de GCP. Se obtuvieron los resultados de latencia y disponibilidad mostrados en las Figuras 4.10 y 4.11.

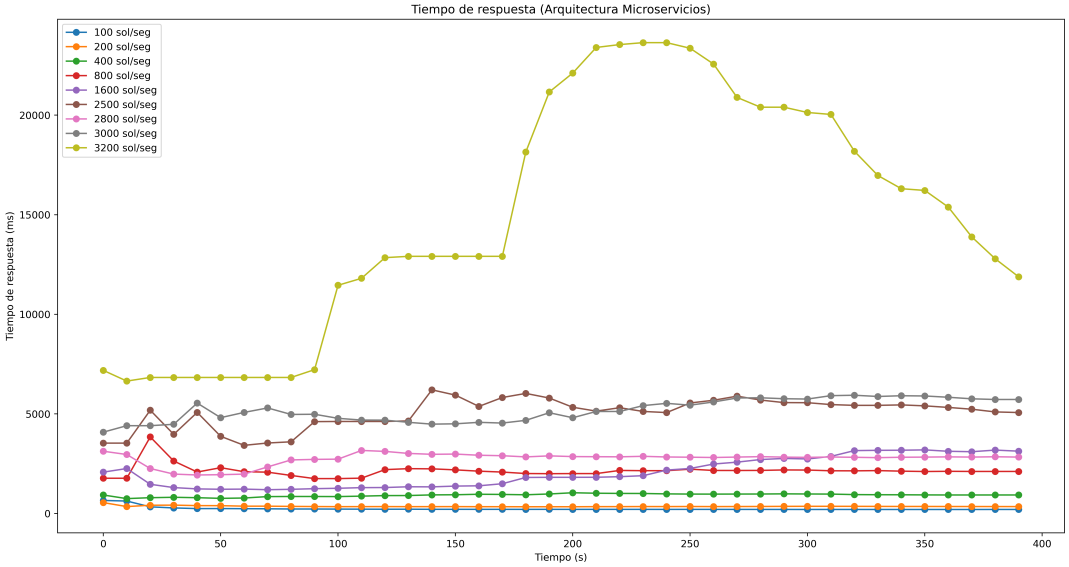


FIGURA 4.10: Tiempo de respuesta de solicitudes sobre la arquitectura de microservicios.

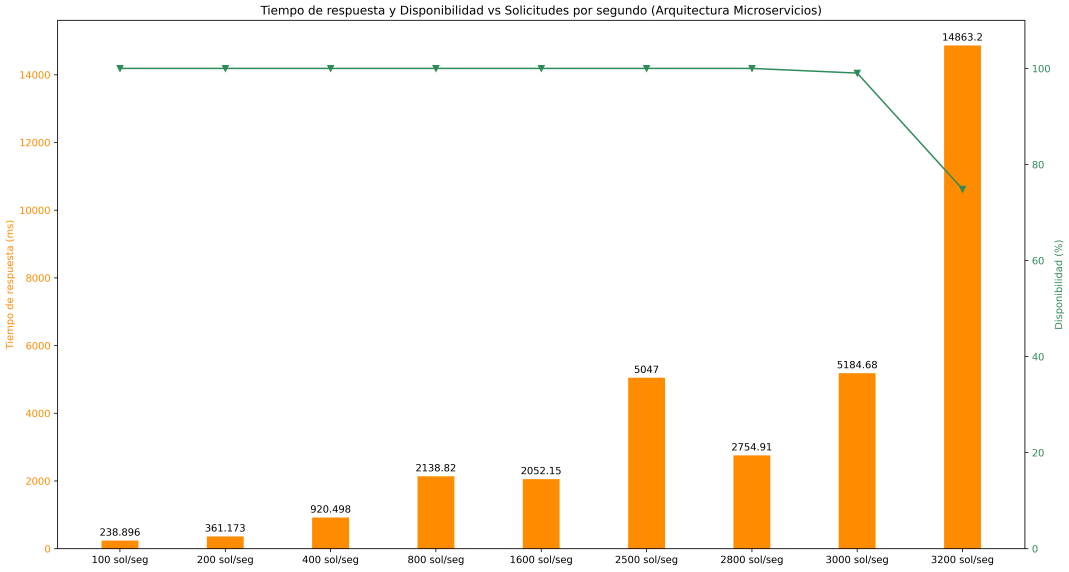


FIGURA 4.11: Tiempo de respuesta y disponibilidad sobre la arquitectura de microservicios.

En la Figura 4.10 se puede observar que la latencia de las solicitudes aumenta a medida que aumenta el número de solicitudes por segundo. Sin embargo, se puede observar que para 100 hasta 3,000 solicitudes por segundo, el tiempo de respuesta se mantiene estable durante los 400 segundos de prueba. Además, el tiempo de respuesta durante toda la prueba no supera los 6 segundos. Sin embargo, para 3,200 solicitudes por segundo, el tiempo de respuesta llega a superar los 20 segundos a partir de los 200 segundos de prueba, lo que indica que la arquitectura de microservicios no es capaz de soportar 3,200 solicitudes por segundo y ya ha llegado a su límite.

Por otra parte, en la Figura 4.11 se puede observar que la disponibilidad del servidor se mantiene estable en 100 % hasta 2,800 solicitudes por segundo, bajando a 99 % para 3,000 solicitudes por segundo y a menos del 80 % para 3,200 solicitudes por segundo. Además, el tiempo de respuesta aumenta de manera lineal a medida que aumenta el número de solicitudes por segundo, superando por poco los 5 segundos para 3,000 solicitudes por segundo, pero llegando a casi 15 segundos para 3,200 solicitudes por segundo.

Por lo tanto, se hizo una mejora sobre la arquitectura de microservicios, agregando un broker Kafka para mejorar la escalabilidad de la aplicación. Por ello, la tercera prueba se realizó sobre esta nueva arquitectura (híbrida), la cual se encuentra desplegada en 5 máquinas virtuales de GCP. Cada una de las máquinas virtuales tiene ejecutando un componente de la arquitectura híbrida, como se muestra en la Figura 4.5. Se obtuvieron los resultados de latencia y disponibilidad mostrados en las Figuras 4.12 y 4.13.

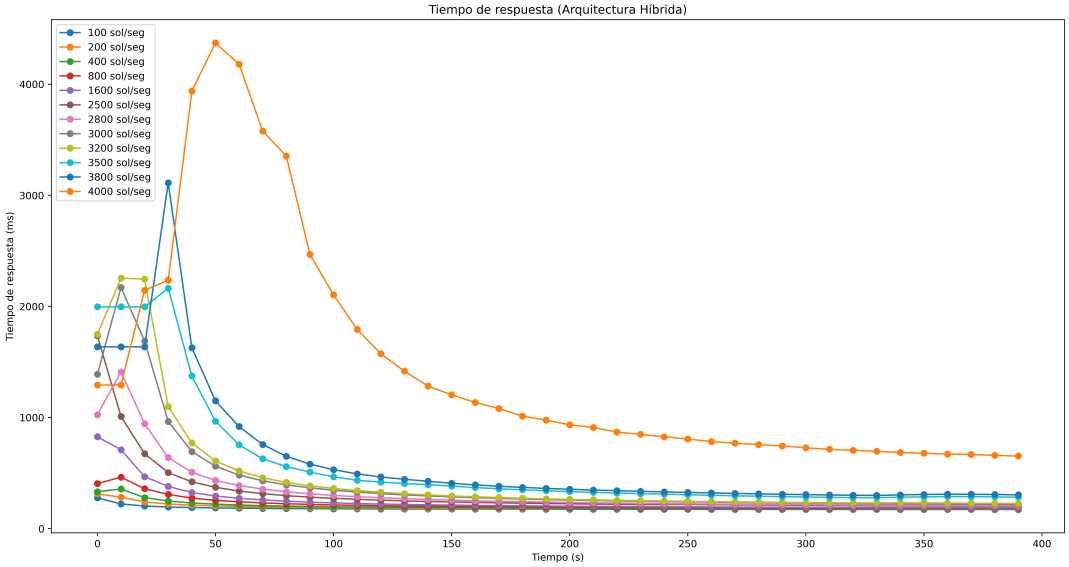


FIGURA 4.12: Tiempo de respuesta de solicitudes sobre la arquitectura híbrida.

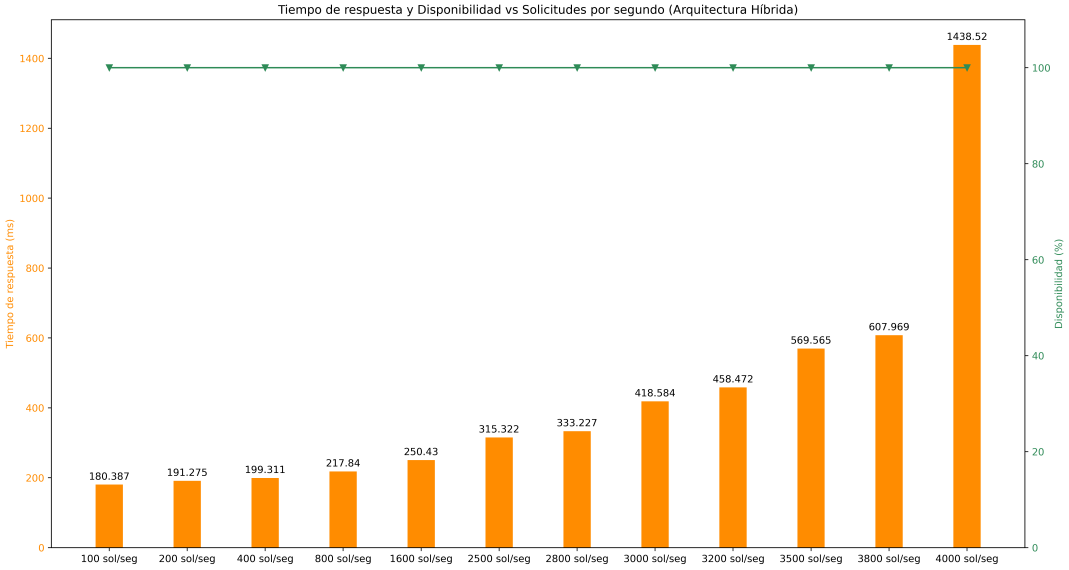


FIGURA 4.13: Tiempo de respuesta y disponibilidad sobre la arquitectura híbrida.

En la Figura 4.12 se puede observar que la latencia de las solicitudes aumenta a medida que aumenta el número de solicitudes por segundo. En esta arquitectura, el tiempo

de respuesta se mantiene estable durante los 400 segundos de prueba desde las 100 hasta las 4,000 solicitudes por segundo. Además, el tiempo de respuesta durante toda la prueba no supera los 1.5 segundos para la tasa máxima de 4,000 solicitudes por segundo. En este sentido, se puede observar que el tiempo de respuesta de la arquitectura híbrida tiene una tendencia logarítmica, lo que indica que la arquitectura híbrida es capaz de soportar una carga de datos mayor que la arquitectura monolítica y la arquitectura de microservicios.

Por otra parte, en la Figura 4.13 se puede observar que la disponibilidad del servidor se mantiene estable en 100 % hasta 4,000 solicitudes por segundo. Además, el tiempo de respuesta aumenta a medida que se aumenta el número de solicitudes por segundo, hasta mantenerse estable en menos de 1.5 segundos al llegar a las 4,000 solicitudes por segundo.

Para poder ver de manera más clara la mejora, tanto de la arquitectura de microservicios como de la arquitectura híbrida, se compara el tiempo de respuesta de estas tres arquitecturas en la Figura 4.14.

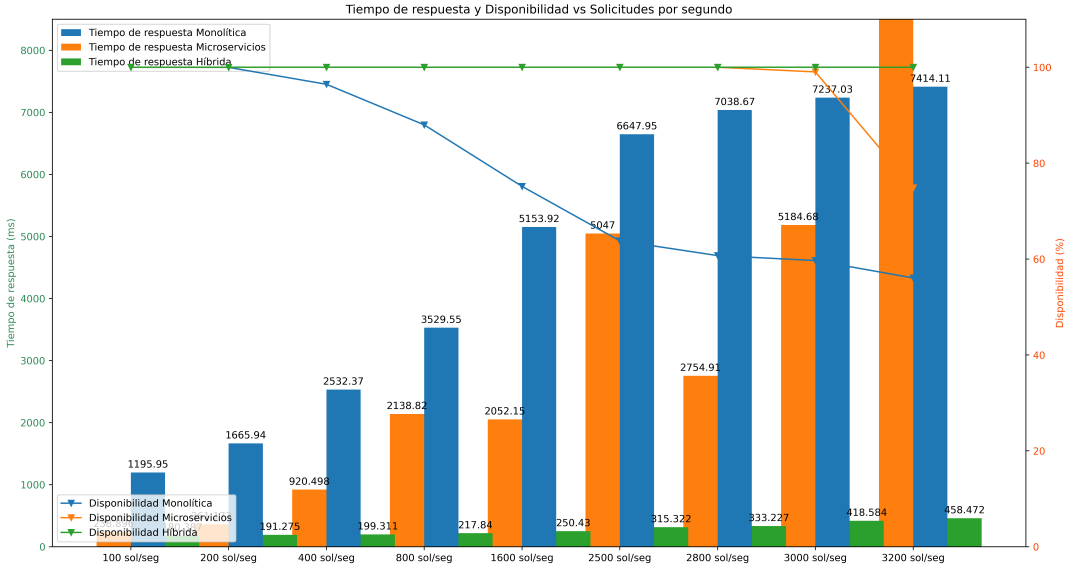


FIGURA 4.14: Tiempo de respuesta de solicitudes sobre la arquitectura inicial, la arquitectura de microservicios y la arquitectura híbrida.

En resúmen, en la Figura 4.14 se puede observar que el tiempo de respuesta de la arquitectura de microservicios es menor que el tiempo de respuesta de la arquitectura monolítica. Además, se puede observar que en promedio el tiempo de respuesta de la arquitectura de microservicios disminuye en un 35.843 % respecto a la arquitectura monolítica. Y para el caso de la disponibilidad, se puede observar que la arquitectura de microservicios aumenta la disponibilidad en un 24.857 % respecto a la arquitectura monolítica.

Por otro lado, se puede observar que el tiempo de respuesta de la arquitectura híbrida es menor que el tiempo de respuesta de la arquitectura de microservicios. Además, se puede observar que en promedio el tiempo de respuesta de la arquitectura híbrida disminuye en un 90.575 % respecto a la arquitectura de microservicios. Y para el caso de la disponibilidad, se puede observar que la arquitectura híbrida aumenta la disponibilidad en un 2.975 % respecto a la arquitectura de microservicios.

4.3 Pruebas de clusterización sobre coordenadas

Como se mencionó en el capítulo anterior, se ha creado un programa en Python que permite ingresar registros de manera aleatoria a la base de datos. Estos registros son después utilizados para realizar pruebas de clusterización utilizando las coordenadas georreferenciadas de los registros. Para ello, se ha creado un endpoint en el *backend* que usa la librería *GoDBSCAN* para realizar la clusterización. Luego de realizar la clusterización, se crea una imagen con los resultados de la clusterización para comprobar que los resultados obtenidos son correctos. Una prueba de ello, se muestra en la Figura 4.15.

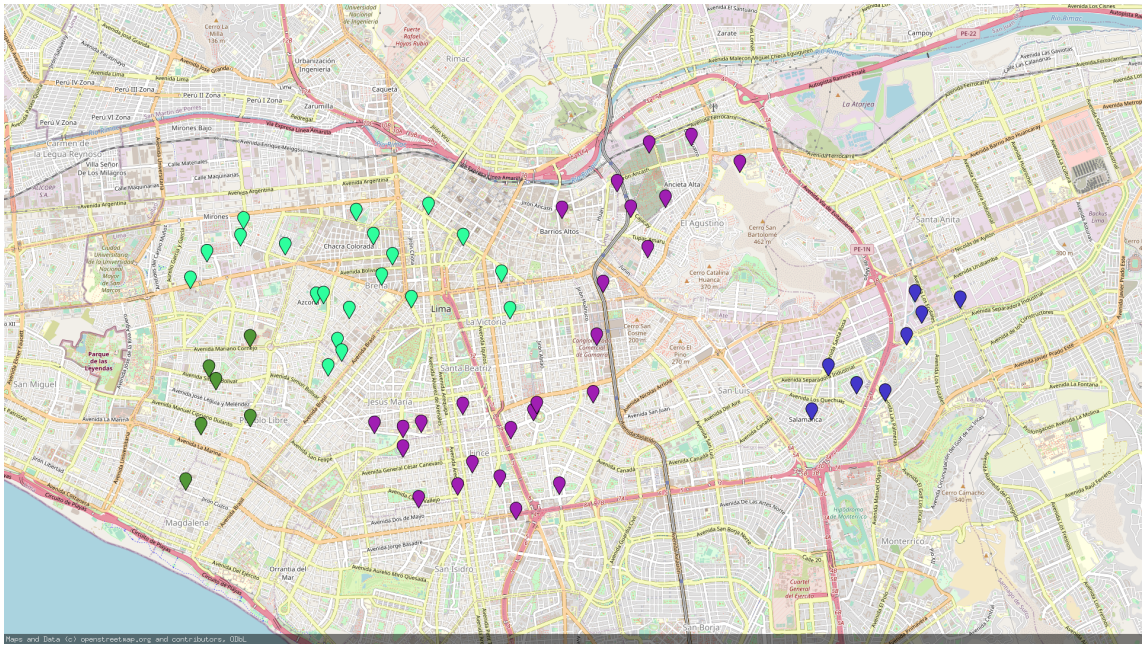


FIGURA 4.15: Clusterización con DBScan usando coordenadas georreferenciadas.

Una vez comprobado que el algoritmo de clusterización funciona correctamente, se procede a realizar pruebas de clusterización agregando la variable fecha a los puntos dentro del algoritmo de clusterización.

4.4 Pruebas de clusterización con ventanas de tiempo

Para realizar las pruebas de clusterización con ventanas de tiempo, se ha creado un programa en Python que permite darle un rango de fecha al *backend* (fecha de inicio y fecha de fin) para que este devuelva los registros que se encuentran dentro de ese rango. A partir de ello, se ha creado un programa en Python que permite construir un gráfico en 3D con los resultados de la clusterización, agregando la fecha como una dimension más, teniendo como dimensiones latitud, longitud y fecha, mostrados en los ejes X, Y y Z respectivamente. El resultado de la clusterización agregando la fecha como una tercera dimensión, se muestra en la Figura 4.16.

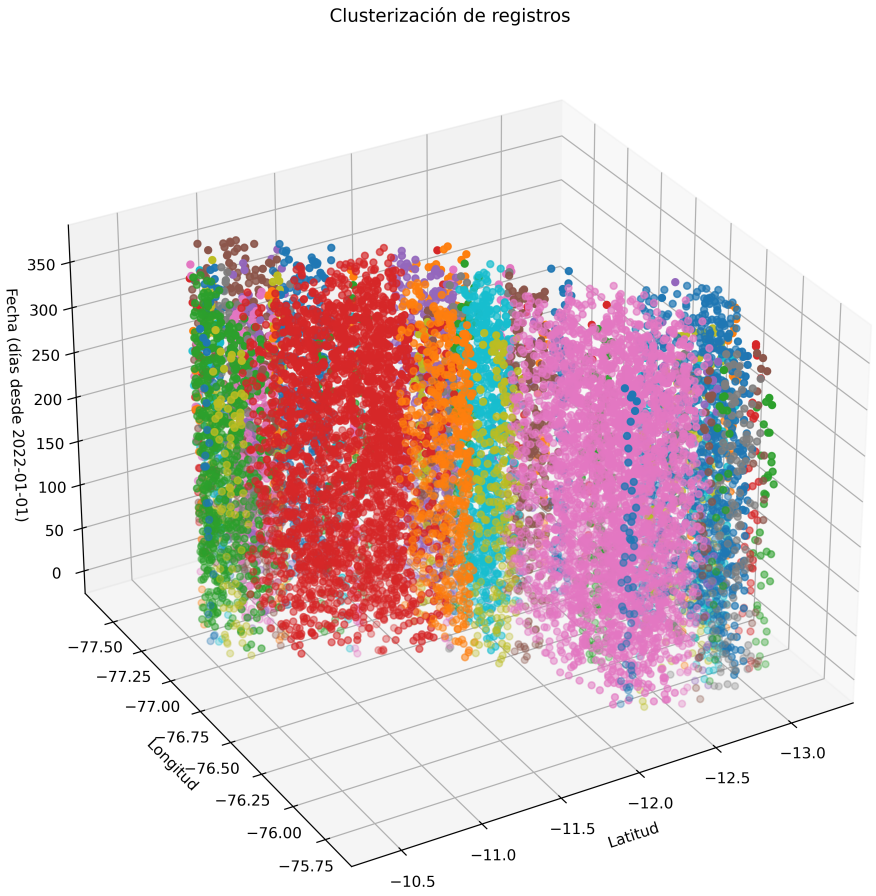


FIGURA 4.16: Clusterización con DBScan usando coordenadas georreferenciadas y fecha.

En la Figura 4.16 se puede observar que se ha ejecutado el algoritmo de clusterización para los registros que se encuentran entre el 1 de enero del 2022 y el 31 de diciembre del 2022. Por ello, se puede observar que cada uno de los clusters identificados tiene un color diferente, y que cada uno de los clusters se encuentra a lo largo de todo el eje Z, que representa la fecha. Esto se debe a que el algoritmo de clusterización se ha ejecutado para todos los registros que se encuentran entre el 1 de enero del 2022 y el 31 de diciembre del 2022.

Este rango de fecha es configurable, es decir, se puede cambiar el rango de fecha para que el algoritmo de clusterización se ejecute para un rango de fecha diferente, dependiendo de la necesidad del análisis. Para el caso de dengue, el tiempo de incubación del virus es de 2 días antes de que aparezcan los síntomas, y 2 días después de la resolución de la fiebre [30]. Por ello, se puede configurar el rango de fecha para que el algoritmo de clusterización se ejecute para un rango de fecha de 4 días.

CONCLUSIONES

Se ha logrado construir el *backend* con los *endpoints* necesarios para leer y escribir registros de casos de dengue sobre la base de datos.

Además, se ha logrado implementar una arquitectura de microservicios, la cual incluye un balanceador de carga y una base de datos PostgreSQL en el mismo servidor en donde se encuentra el *backend*. Todo esto, con el objetivo de eliminar el cuello de botella encontrado en un inicio, y que el *backend* pueda soportar una carga mayor de solicitudes.

Por otra parte, también se ha logrado arquitectura híbrida, la cual incluye un broker de mensajería de Apache Kafka y un productor y consumidor de mensajes, con el objetivo de que el *backend* pueda soportar una carga mayor de solicitudes, disminuya el tiempo de respuesta y la disponibilidad se mantenga en un 100 %. Debido a esto, el tiempo de respuesta tiene un comportamiento logarítmico, es decir, a medida que aumenta el número de solicitudes por segundo, el tiempo de respuesta disminuye hasta un punto en donde se mantiene constante.

Tambien se ha logrado implementar el algoritmo de clusterización para los registros, con el objetivo de identificar patrones de propagación de la enfermedad. Finalmente, se ha logrado implementar un *frontend* para visualizar los datos en forma de tabla o usando un mapa interactivo y que se actualice en tiempo real.

- Se ha realizado la construcción de un *backend* con la arquitectura de microservicios, que cuenta con la capacidad de escribir y leer registros de casos e inspecciones de dengue y realizar clusterización sobre dichos registros.
- La arquitectura de microservicios presenta una clara mejoría de disponibilidad y tiempo de respuesta con respecto a la arquitectura monolítica. Así como también, la arquitectura híbrida presenta una clara mejoría de disponibilidad y tiempo de respuesta con respecto a la arquitectura de microservicios.
- El tiempo de respuesta con la arquitectura de microservicios no supera los 6 segundos con 3,000 solicitudes por segundo, pero supera los 20 segundos con 3,200 solicitudes por segundo. Sin embargo, con la arquitectura híbrida, el tiempo de respuesta se mantiene en promedio en 0.3 segundos con hasta 3,200 solicitudes por segundo.
- La disponibilidad de la arquitectura de microservicios se mantiene en 100 %, bajando a 99 % con 3,000 solicitudes por segundo, pero a menos de 80 % con 3,200 solicitudes por segundo. Sin embargo, con la arquitectura híbrida, la disponibilidad se mantiene en 100 % con hasta 3,200 solicitudes por segundo.
- Con la arquitectura de microservicios, el tiempo de respuesta de las solicitudes bajó en un 35.843 %, y la disponibilidad aumentó en un 24.857 % con respecto a la arquitectura monolítica.
- Con la arquitectura híbrida, el tiempo de respuesta de las solicitudes bajó en un 90.575 % y la disponibilidad aumentó en un 2.975 % con respecto a la arquitectura de microservicios.
- El algoritmo de clusterización puede ejecutarse de manera simultánea junto con la lectura y escritura de registros.
- El algoritmo de clusterización permite separar los registros en ventanas de tiempo para un análisis minucioso y poder encontrar patrones más acotados.

RECOMENDACIONES

4.5 Alcances y limitaciones

El alcance de este proyecto es poder recibir registros de inspecciones de viviendas y casos de dengue en toda el área de la región de Lima. Sin embargo, más adelante se puede llegar a incrementar el área de cobertura a otras regiones del Perú, llegando incluso a cubrir todo el territorio nacional. Del mismo modo, este sistema de monitorización puede ser usado para el registro de casos de otras enfermedades que también necesiten ser monitorizadas, ya que las bases utilizadas para la construcción de este sistema no están limitadas a una sola enfermedad como es el dengue. Ya que cada enfermedad sigue un patrón de propagación diferente, se puede llegar a agregar un módulo de análisis de datos para poder detectar patrones de propagación de enfermedades y así poder predecir la propagación de una enfermedad en un área determinada.

Por otro lado, la limitación encontrada en el desarrollo de este proyecto es que solo se ha usado simulaciones para realizar las validaciones del comportamiento de las funciones de visualización y la actualización de datos en tiempo real. Esto es debido a que aún no sé cuenta con una gran cantidad de usuarios dentro del área de estudio que permitan realizar los reportes de casos mediante la aplicación móvil.

4.6 Trabajos futuros

En futuras investigaciones se pueden incluir nuevos servicios de análisis más especializados, tales como mapas de calor, modelos de predicción de casos, entre otros. Debido a las arquitecturas de microservicios e híbrida utilizadas para la construcción de esta investigación, se pueden agregar funciones de análisis adicionales como nuevos microservicios o módulos de análisis en el servidor para realizar análisis más específicos para otras enfermedades.

ANEXOS

A continuación se presentan los anexos del presente trabajo de investigación.

Anexo 1: Código fuente del proyecto

Código fuente del productor de Kafka

. El código del productor de Kafka se encuentra en GitHub en el siguiente enlace: https://github.com/Piero16301/DENV_Register_Producer.

Código fuente del consumidor de Kafka

El código del consumidor de Kafka se encuentra en GitHub en el siguiente enlace: https://github.com/Piero16301/DENV_Cases_Consumer.

Código fuente de los microservicios

El código de los microservicios se encuentra en GitHub en el siguiente enlace: https://github.com/Piero16301/DENV_Backend.

Código fuente del cliente web

El código del cliente web se encuentra en GitHub en el siguiente enlace: https://github.com/Piero16301/DENV_Desktop.

Código fuente de la aplicación móvil

El código de la aplicación móvil se encuentra en GitHub en el siguiente enlace: https://github.com/Piero16301/DENV_Mobile.

Anexo 2: Clusterización con Neo4j

Se han utilizado datos de prueba para realizar la clusterización de los nodos de la base de datos de Neo4j. El resultado de la clusterización se puede observar en la Figura 4.17.

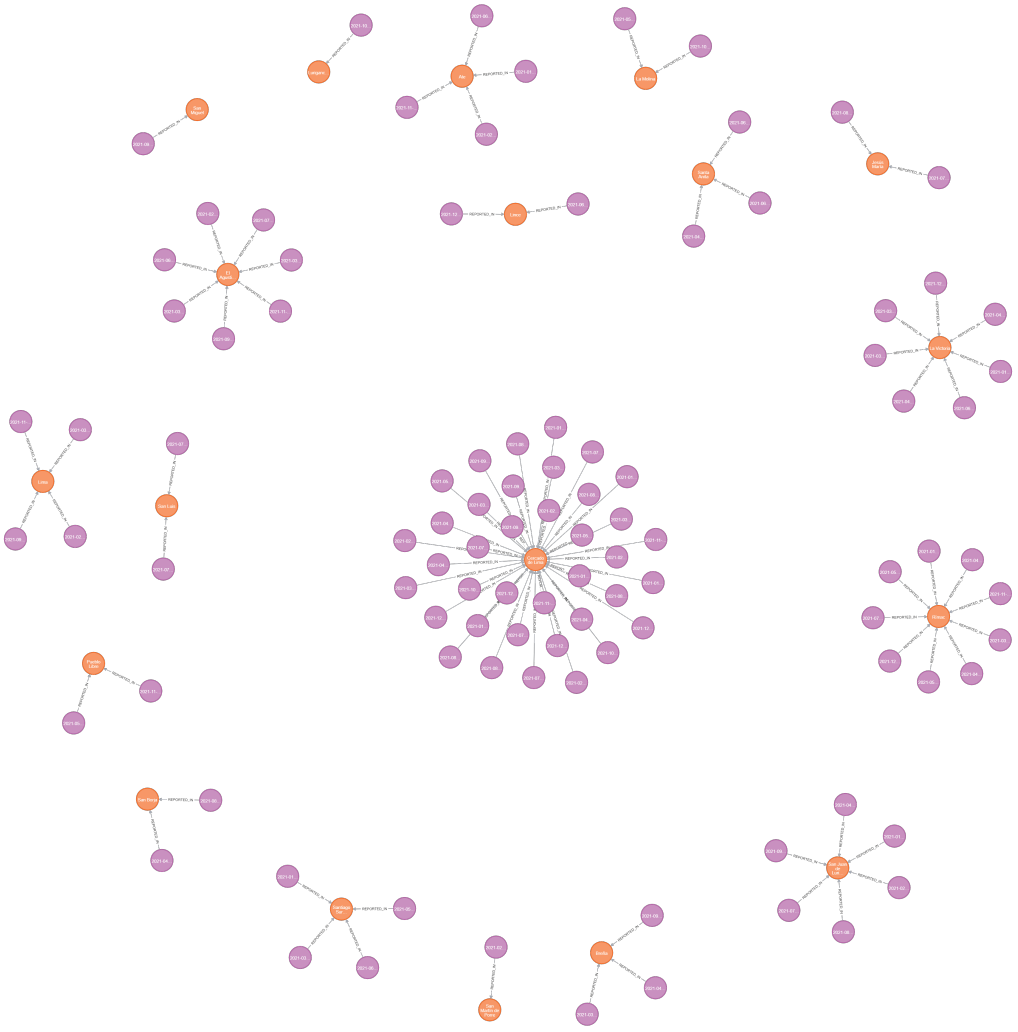


FIGURA 4.17: Clusterización de los nodos de la base de datos de Neo4j. Fuente: Elaboración propia.

Anexo 3: Clusterización por ventanas de tiempo

Se ha utilizado el algoritmo de DBSCAN para realizar la clusterización de los casos de dengue por ventanas de tiempo. El resultado de la clusterización se puede observar en la Figura 4.18.

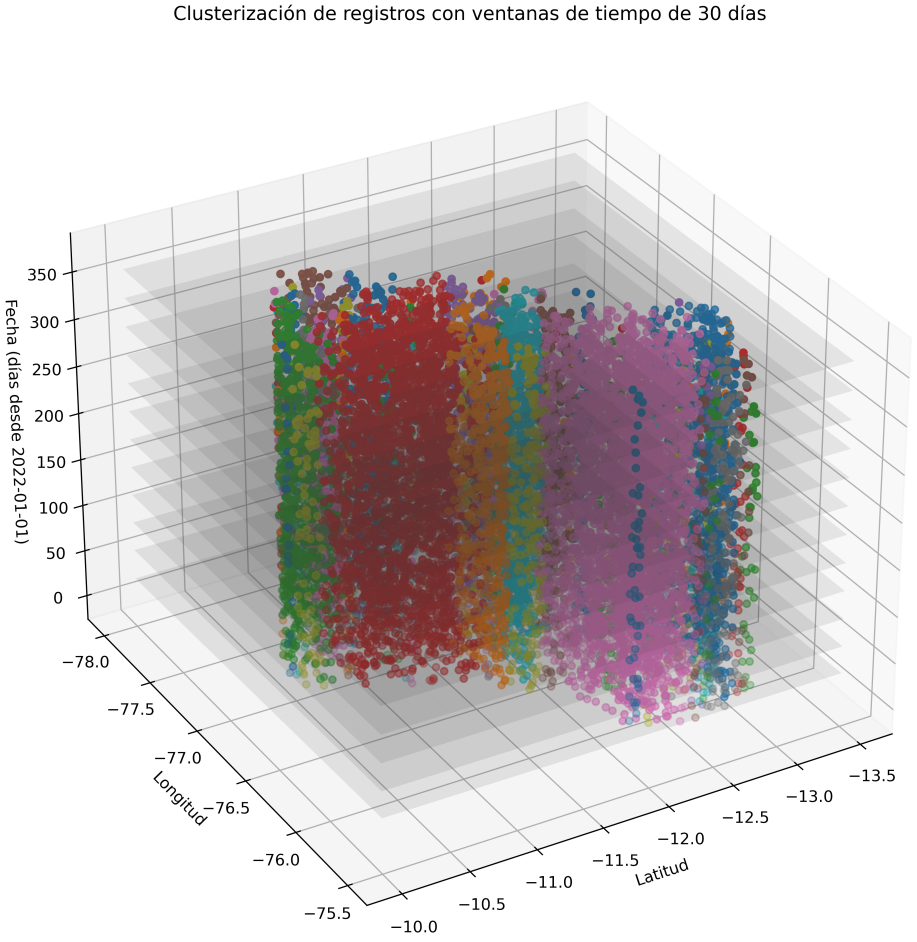


FIGURA 4.18: Clusterización de los casos de dengue por ventanas de tiempo. Fuente: Elaboración propia.

Anexo 4: Uso de recursos del broker de Kafka

Se ha monitorizado el uso de recursos del broker Kafka utilizando el comando `htop` de Linux. El monitoreo se ha realizado en el momento en el que se estaban registrando los casos de dengue en la base de datos. El resultado de la monitorización se puede observar en la Figura 4.19.

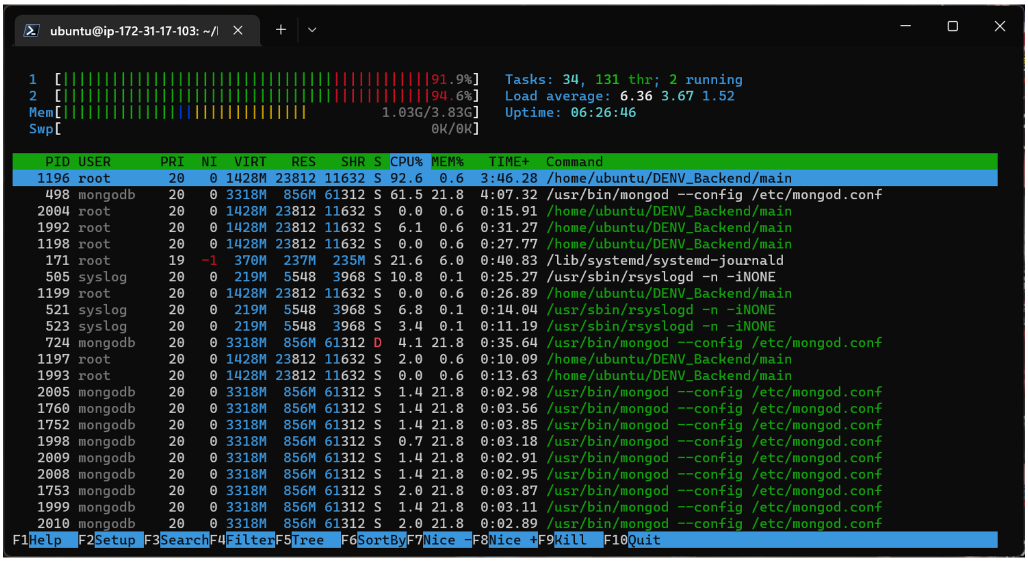


FIGURA 4.19: Uso de recursos del broker de Kafka. Fuente: Elaboración propia.

Anexo 5: Componentes desplegados en GCP

Se ha desplegado los componentes de la arquitectura híbrida en GCP. Los 4 componentes de la arquitectura híbrida se pueden observar en la Figura 4.20.

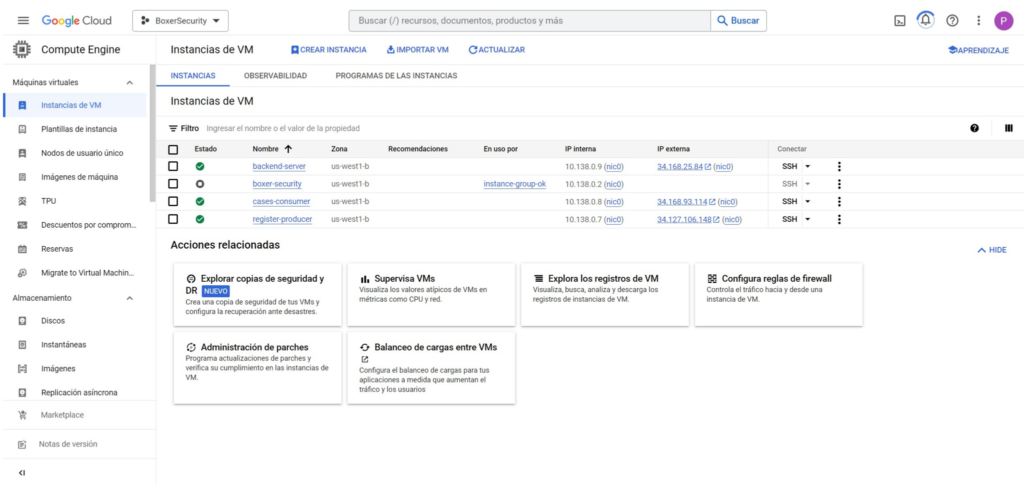


FIGURA 4.20: Componentes desplegados en GCP. Fuente: Elaboración propia.

REFERENCIAS BIBLIOGRÁFICAS

[1] M. de Salud del Perú. Sala de situación de la vigilancia epidemiológica de dengue. [Online]. Available: https://www.dge.gob.pe/salasituacional/sala/index/SALA_VIGILA/141

[2] ——. Sala situacional de dengue. [Online]. Available: https://www.dge.gob.pe/sala-situacional-dengue/#grafico01

[3] I. N. de Estadística e Informática. La población de lima supera los nueve millones y medio de habitantes. [Online]. Available: http://m.inei.gob.pe/prensa/noticias/la-poblacion-de-lima-supera-los-nueve-millones-y-medio-de-habitantes-12031/

[4] A. P. de Noticias—ANDINA. Dengue: activan cerco epidemiológico en san juan de lurigancho tras primer caso autóctono. [Online]. Available: http://tinyurl.com/CasoAutoctono

[5] E. M. Delmelle, H. Zhu, W. Tang, and I. Casas, "A web-based geospatial toolkit for the monitoring of dengue fever," *Applied Geography*, vol. 52, pp. 144–152, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0143622814001027

[6] M. de Salud del Perú. Minsa: efectividad de la fumigación llega al 98 por ciento para eliminar al zancudo que transmite el dengue. [Online]. Available: https://web.ins.gob.pe/es/prensa/noticia/minsa-efectividad-de-la-fumigacion-llega-al-98-para-eliminar-al-zancudo-que

[7] ——. Dengue — digesa. [Online]. Available: http://www.digesa.minsa.gob.pe/material_educativo/dengue.asp#:~:text=%C2%BFCu%C3%A1l%20es%20el%20tratamiento%20del,en%20reposo%20e%20ingiera%20l%C3%ADquidos.

[8] J. Fowks. Perú se enfrenta al avance de la pandemia sin suficientes camas de uci, oxígeno ni médicos en los hospitales. [Online]. Available: http://tinyurl.com/AvancePandemia

[9] M. Suznjevic and J. Saldana, "Delay limits for real-time services," *IETF draft*, 06 2016.

[10] M. de Salud del Perú. Sala de situación de covid. [Online]. Available: https://www.dge.gob.pe/covid19.html

[11] E. Reddy, S. Kumar, N. Rollings, and R. Chandra, "Mobile application for dengue fever monitoring and tracking via gps: case study for fiji," *arXiv preprint arXiv:1503.00814*, 2015.

[12] B. Vutla, Y. S. Vemulapalli, S. S. Kurra, and C. R. Madhuri, "Dengue alert system–a life saviour," in *2020 7th International Conference on Smart Structures and Systems (ICSSS)*. IEEE, 2020, pp. 1–6.

[13] A. M. MacEachren, C. A. Brewer, and L. W. Pickle, "Visualizing georeferenced data: representing reliability of health statistics," *Environment and planning A*, vol. 30, no. 9, pp. 1547–1561, 1998.

[14] Z. Cao, Z. Wu, G. Guo, W. Ma, and H. Wang, "Quantifying spatial associations between effective green spaces and cardiovascular and cerebrovascular diseases by applying volunteered geo-referenced data," *Environmental Research Letters*, vol. 17, no. 1, p. 014055, 2022.

[15] C.-T. Yang, C.-J. Chen, Y.-T. Tsan, P.-Y. Liu, Y.-W. Chan, and W.-C. Chan, "An implementation of real-time air quality and influenza-like illness data storage and processing platform," *Computers in Human Behavior*, vol. 100, pp. 266–274, 2019.

[16] J. Guo, H. Liu, D. Zhou, J. Chai, Y. Zhang, and Y. Liu, "Real-time power system electromechanical mode estimation implementation and visualization utilizing synchrophasor data," in *2016 IEEE/PES Transmission and Distribution Conference and Exposition (T&D)*. IEEE, 2016, pp. 1–5.

[17] N. G. Society. Gis (geographic information system). [Online]. Available: https://www.nationalgeographic.org/encyclopedia/geographic-information-system-gis/

[18] J. Celko, "Temporal Data," *Joe Celko's Data, Measurements and Standards in SQL*, pp. 253–269, 2010.

[19] A. Buckley. Working with temporal data in arcgis. [Online]. Available: https://www.esri.com/about/newsroom/arcuser/working-with-temporal-data-in-arcgis/

[20] I. Technology. Real-time reports definition. [Online]. Available: https://www.inetsoft.com/business/solutions/real_time_reports_definition/

[21] I. Striim. 5 uses for real-time visualization. [Online]. Available: https://www.striim.com/blog/5-uses-for-real-time-visualization/

[22] C. Richardson. Microservices architecture pattern. [Online]. Available: https://microservices.io/patterns/microservices.html

[23] Go-Chi. jwtauth - jwt authentication middleware for http services. [Online]. Available: https://github.com/go-chi/jwtauth

[24] Go-Rs. Go cors handler. [Online]. Available: https://github.com/rs/cors

[25] Gorm.io. Gorm - the fantastic orm library for golang. [Online]. Available: https://gorm.io

[26] Go-Chi. Go-chi middleware. [Online]. Available: https://github.com/go-chi/chi/tree/master/middleware

[27] sjbog. Dbscan (go package). [Online]. Available: https://bitbucket.org/sjbog/go-dbscan/src/master/

[28] Locust. Locust - a modern load testing framework. [Online]. Available: https://locust.io/

[29] D. del Pueblo Perú. Centros de salud de lima registran graves problemas de infraestructura y falta de personal médico. [Online]. Available: http://tinyurl.com/DefensoriaPueblo

[30] O. M. de la Salud. Dengue y dengue grave. [Online]. Available: https://www.who.int/es/news-room/fact-sheets/detail/dengue-and-severe-dengue