Universidad Nacional de Loja

Facultad de la Energía, las Industrias y los Recursos Naturales no Renovables

Carrera de Ingeniería en Sistemas

Implementación de una arquitectura de microservicios y técnicas de DevOps a los servicios backend de la plataforma “GeoWorker”

Implementation of a microservices architecture and DevOps techniques to the backend services of the “GeoWorker” platform.

Trabajo de Titulación, previo a la obtención del título de Ingeniero en Sistemas.

AUTOR:

Manuel Alejandro Aguinsaca Medina

DIRECTOR:

Ing. Edwin René Guamán Quinche. Mg. Sc.

Loja – Ecuador

2024

Certificación

Loja, 30 de septiembre de 2024

Ing. Edwin René Guamán Quinche. Mg. Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

CERTIFICO:

Que he revisado y orientado en todo el proceso de elaboración del trabajo de titulación denominado: **Implementación de una arquitectura de microservicios y técnicas de DevOps a los servicios backend de la plataforma "GeoWorker"** de la autoría del estudiante **Manuel Alejandro Aguinsaca Medina**, con **cédula de identidad Nro. 1105947350**, previo a la obtención del título de **Ingeniero en Sistemas**, una vez que el trabajo cumple con todos los requisitos exigidos por la Universidad Nacional de Loja, para el efecto, autorizo la presentación del mismo, para su respectiva sustentación y defensa.

Firmado electrónicamente por:
EDWIN RENE GUAMAN QUINCHE

Ing. Edwin René Guamán Quinche. Mg. Sc.

DIRECTOR DEL TRABAJO DE TITULACIÓN

Autoría

Yo, **Manuel Alejandro Aguinsaca Medina**, declaro ser autor del presente Trabajo de Titulación y eximo expresamente a la Universidad Nacional de Loja y a sus representantes jurídicos de posibles reclamos y acciones legales por el contenido del mismo. Adicionalmente acepto y autorizo a la Universidad Nacional de Loja la publicación de mi Trabajo de Titulación en el Repositorio Digital Institucional – Biblioteca Virtual.

Firma:

Cédula: 1105947350

Fecha: 30 de septiembre de 2024

Correo electrónico: maaguinsacam@unl.edu.ec

Teléfono: (+593) 0985576629

Carta de autorización por parte del autor, para consulta, reproducción parcial o total y/o publicación electrónica del texto completo del Trabajo de Titulación

Yo, **Manuel Alejandro Aguinsaca Medina**, declaro ser autor del Trabajo de Titulación denominado: **Implementación de una arquitectura de microservicios y técnicas de DevOps a los servicios backend de la plataforma "GeoWorker"**, como requisito para optar el título de **Ingeniero en Sistemas**, autorizo al sistema Bibliotecario de la Universidad Nacional de Loja para que con fines académicos muestre la producción intelectual de la Universidad, a través de la visibilidad de su contenido de la siguiente manera en el Repositorio Institucional.

Los usuarios pueden consultar el contenido de este trabajo en el Repositorio Institucional, en las redes de información del país y del exterior, con las cuales tenga convenio la Universidad.

La universidad Nacional de Loja, no se responsabiliza por el plagio o copia del Trabajo de Titulación que realice un tercero.

Para constancia de esta autorización, suscribo, en la ciudad de Loja, a los treinta días del mes de septiembre, de dos mil veinticuatro.

Firmado electrónicamente por:
MANUEL ALEJANDRO
AGUINSACA MEDINA

Firma:

Autor: Manuel Alejandro Aguinsaca Medina

Cédula: 1105947350

Dirección: Loja - Ecuador

Correo electrónico: maaguinsacam@unl.edu.ec

Teléfono: (+593) 0985576629

DATOS COMPLEMENTARIOS

Director del Trabajo de Titulación: Ing. Edwin René Guamán Quinche. Mg. Sc.

Dedicatoria

A mi madre Angelita Medina, por todo el apoyo y amor incondicional que me ha brindado en este largo proceso y por ser el pilar de mi vida. A mi abuelita Imelda Paccha, mis hermanos Fabian, Sandra, Gaby y Melisa, que me han sabido guiar y aconsejar cuando lo he necesitado. A mi tío Johnson Medina, que, aunque no esté presente, sus palabras, su cariño y su memoria sigue presente en vida. A Janeth, por la paciencia y el cariño incondicional que me ha brindado.

A mis amigos Raisa, Jonathan, Danny y Alexander, por el apoyo y la amistad brindad en todos estos años.

Manuel Alejandro Aguinsaca Medina

Agradecimiento

Quiero agradecer a mi madre Angelita Medina, que con trabajo y esfuerzo diario logró apoyarme en todo momento, a mi abuelita Imelda Paccha, mi tío Johnson, mis hermanos Fabian, Sandra, Gabriela, Melisa.

Al Ingeniero René Guamán, que gracias a su guía y paciencia he podido alcanzar esta meta. A todos mis maestros que hasta el día de hoy han compartido su sabiduría y conocimientos para formarme como profesional.

Manuel Alejandro Aguinsaca Medina

Índice de Contenidos

Índice de tablas:

Índice de figuras:

Índice de Anexos:

1. Título

Implementación de una arquitectura de microservicios y técnicas de DevOps a los servicios backend de la plataforma GeoWorker

Implementation of a microservices architecture and DevOps techniques to the backend services of the “GeoWorker” platform.

2. Resumen

La empresa PuntoPymes, como cualquier empresa de tecnología, tiene diferentes retos al momento de ofrecer soluciones informáticas a sus clientes, uno de los más importantes y complicados es ir adaptándose a nuevas tecnologías que le permitan ofrecer sistemas rápidos, escalables y eficientes con los que pueda mantener su posición competitiva en el mercado, además, que estos sistemas sean provechosos para sus clientes automatizando procesos, eliminando errores y permitiendo que la información importante sea accesible en todo momento. En este contexto, el objetivo del Trabajo de Titulación de enfocó en implementar técnicas de DevOps a los servicios backend de la plataforma "GeoWorker", para este fin, se definió dos fases con actividades específicas con el propósito de cumplir el objetivo antes mencionado. i) En la primera fase mediante una entrevista se elaboró un documento con la información general de la plataforma, a partir de este documento, se obtuvo los requisitos Funcionales y no Funcionales mediante el estándar IEEE 830. Posteriormente, se diseñó la arquitectura de microservicios que se implementaría para la plataforma, este proceso se lo realizó a través del patrón de diseño Domain-Driven Design. Luego se codificó la arquitectura utilizando el framework Spring Cloud, y para los microservicios se implementó Java con Spring-Boot y NodeJS con NestJS. El almacenamiento de la información se lo realizó con PostgreSQL y MongoDB, así mismo, la comunicación se la implementó mediante RabbitMQ. Para finalizar esta primera fase, se procedió a implementar CI/CD, Monitoreo, Construcción, Control de Versiones de Código y Automatización de Pruebas como técnicas de DevOps a través del pipeline de GitLab a los servicios desarrollados, y al mismo tiempo se desplegó la arquitectura en EC2 de Amazon Web Services (AWS). ii) La segunda fase consistió en evaluar los servicios backend de la plataforma con el programa JMeter. Finalmente, los resultados demostraron que las técnicas de DevOps combinadas con una arquitectura de microservicios mejoraron los tiempos de entrega, la disponibilidad y la eficiencia de la plataforma "GeoWorker", permitiendo gestionar fácilmente altas cargas de trabajo.

***Palabras Claves**: DevOps, Microservicios, XP, DDD, Spring Cloud, RabbitMQ*

Abstract

The PuntoPymes company, like any technology company, has different challenges when they offer solutions to their clients, one of the most important and complicated is adapting to new technologies that allow them offer fast, scalable and efficient systems with which they can maintain their competitive position in the market; in addition, these systems are beneficial for clients by automating processes, eliminating errors and allowing important information to be accessible at all times. In this context, the objective of the Degree Work was focused on implementing DevOps techniques to the backend services of the "GeoWorker" platform; for this purpose, two phases were defined. i) In the first one, through an interview, a document was prepared with the general information of the platform. From this document, the Functional and Non-Functional requirements were obtained using the IEEE 830 standard. Subsequently, the microservices architecture was designed that would be implemented for the platform, this process was carried out through the Domain-Driven Design pattern. Then, the architecture was coded using the Spring Cloud framework, and the microservices Java with Spring-Boot and NodeJS with NestJS were implemented. The information was saved up with PostgreSQL and MongoDB, likewise, the communication was implemented using RabbitMQ. To complete this first phase, CI/CD, Monitoring, Construction, Code Version Control and Test Automation were implemented as DevOps techniques through the GitLab pipeline to the developed services, as well the architecture was deployed on EC2 from Amazon Web Services (AWS). ii) The second one consisted of evaluating the backend services of the platform with the JMeter program. Finally, the results demonstrated that DevOps techniques combined with a microservices architecture improved delivery times, availability and efficiency of the "GeoWorker" platform, allowing high workloads to be easily managed.

Keywords: *DevOps, Microservices, XP, DDD, Spring Cloud, RabbitMQ*

3. Introducción

Los Recursos Humanos (RRHH) y su administración siempre han sido un especto importante a considerar cuando una organización busca alcanzar los objetivos que se ha propuesto. En este sentido, las organizaciones constantemente buscan mejorar sus procesos e identificar posibles problemas que puedan interrumpir el alcance de las metas trazadas. En la encuesta realizada en [1], donde participaron 73 directores de empresas, se identificaron 11 factores que generalmente dificultan la gestión empresarial. Cinco de estos factores (contratación de personal calificado, falta de formación, personal poco dispuesto a asumir responsabilidades, seguimiento de tareas), es decir, aproximadamente el 45% están relacionados con los RRHH. Estos datos muestran la necesidad de las organizaciones de contar con sistemas de administración de recursos humanos bien diseñados y que permitan gestionar los recursos de manera fácil y eficiente. Es así, que en la industria manufacturera se valora mucho los sistemas con las características antes mencionadas, ya que no solo mejoran la gestión, sino que también, influyen y potencian a largo plazo la relación entre el rendimiento organizativo y la administración de recursos humanos ofreciendo grandes beneficios para las empresas [2].

En la actualidad las empresas han tenido que ir evolucionando y adaptándose a los nuevos sistemas y aplicaciones que los avances tecnológicos ofrecen, además, problemas como contingencias sanitarias han provocado que los protocolos y procesos de una empresa cambien, haciendo que los sistemas y arquitecturas monolíticas queden obsoletos frente a la alta demanda de usuarios y la poca capacidad de adaptación a nuevos cambios que las organizaciones requieren.

Por lo tanto, nace la interrogante en si la combinación de una arquitectura de microservicios y técnicas de DevOps, pueden otorgar a un sistema informático características como la modularidad, mantenibilidad, escalabilidad y eficiencia. Y así, construir una solución informática que permita a las empresas agilizar y mejorar los procesos de seguimiento, evaluación y control de sus colaboradores que se lleva internamente.

En este contexto, el presente Trabajo de Titulación (TT), tiene como objetivo principal implementar técnicas de DevOps a los servicios backend de la plataforma “GeoWorker”. Por lo tanto, la memoria se estructuró considerando el formato establecido por la Universidad Nacional de Loja y está conformada de las siguientes secciones. El Marco Teórico detalla los conceptos más relevantes que ayudaron a sustentar el TT, estos conceptos se enfocan en temas como la administración de recursos humanos, sistemas de administración de recursos humanos,

generalidades de la arquitectura de microservicios e información acerca del patrón de diseño Domain-Driven Design (DDD), sus conceptos básicos y el proceso para su implementación. Finalmente, en esta sección se aborda las diferentes técnicas de DevOps que se pueden implementar a un proyecto de desarrollo de software. En el apartado de la Metodología se establece el área de estudio, se listan las actividades a cumplir en cada objetivo específico, se detallan los recursos utilizados y los participantes que intervienen en el TT. En la sección de Resultados se presenta las evidencias del cumplimiento de los objetivos propuestos y en la Discusión se considera y analiza los resultados de la sección anterior. Por último, en las Conclusiones se presentan los hallazgos derivados del desarrollo del TT, y en las Recomendaciones se plantean sugerencias procedentes del desarrollo del proyecto e ideas para investigaciones futuras.

4. Marco Teórico

En esta sección, se detallan los conceptos, técnicas y herramientas en los que se fundamentó y las que se utilizó para el desarrollo del presente Trabajo de Titulación.

4.1. Recursos Humanos

El término recursos humanos se refiere al personal de una organización que tiene cualidades y competencias específicas para desempeñarse en niveles administrativos, operativos, técnico o gerencial dentro de la empresa, también, gracias a la importancia que ha tomado en la actualidad, se lo ha denominado como "Capital Humano". Los recursos humanos se distribuyen en los siguientes niveles:

- Nivel Institucional (Dirección)
- Nivel Intermedio (Gerencia y Asesoría)
- Nivel Operacional (Técnico, Empleados y Obreros) [3].

Figura 1. *Relación entre recursos humanos y la empresa*

4.1.1. Gestión de Recursos Humanos

La gestión de los recursos humanos de una empresa siempre ha tenido como objetivo guiar a los empleados con el fin de que ejecuten acciones comunes que lleven a cumplir las metas que la organización se ha planteado [4]. También se dice que es una práctica para la gestión de empleados de una empresa y un proceso estratégico que tiene que estar alineado a los objetivos establecidos por la dirección de la organización [5].

Para desarrollar un sistema de administración de recursos humanos, es fundamental conocer los principios y procesos de la empresa, ya que puede ocasionar que el sistema no aporte los beneficios esperados por la empresa [6].

4.1.2. Evaluación de Recursos Humanos

La evaluación de recursos humanos o evaluación de desempeño por competencias laborales es un aspecto importante dentro de la gestión de recursos humanos ya que permiten obtener de manera objetiva el rendimiento o desempeño de os empleados dentro de una empresa [7].

4.1.3. Sistemas de Gestión de Recursos Humanos

Los sistemas de gestión de recursos humanos deben estar apegados a las estrategias de la organización y debe contemplar dos filosofías de gestión de recursos humanos, las mismas que se detallan a continuación:

- **Sistema de Mercado:** Este tipo de sistemas permite que la organización adquiera empleados desde el mercado externo.
- **Sistema Interno:** En este tipo de sistemas, la organización forma a sus empleados y refuerza los conocimientos de sus empleados de forma interna, es decir, la organización invierte en sus empleados para tener un rendimiento y estabilidad a largo plazo [8].

4.1.4. Administración de Horarios o Asistencia

La gestión de asistencia es el proceso por el cual, se determina la hora de entrada o salida de un empleado. En el pasado las empresas llevaban este tipo de procesos de manera manual, haciendo que el mismo proceso se vuelva lento e ineficiente, además de provocar que los datos sean propensos a daños o perdidas ya que se plasmaban en papel [9].

En la **Tabla 1**, podemos observar una comparativa entre sistemas que se centran en la gestión de asistencias y tiene un pequeño enfoque en la gestión de recursos humanos.

Tabla 1. *Comparativa entre sistemas de gestión de asistencias*

Características	**Productos**		
	ClockIt	**ADP Workforce**	**Jibble**
Plataformas	Web & Móvil App	Web & Móvil App	Web & Móvil App
Organizador de Reuniones	No	No	No
Administración de comentarios	No	No	No
Generador de Reportes	Si	Si	Si

4.2. Arquitectura de Microservicios

Según [10], la arquitectura de microservicio o sistemas distribuidos es un estilo arquitectónico que trata de mitigar las prácticas de agrupar toda la lógica de negocio en un solo monolito, es decir, se centra en crear pequeños servicios escalables y distribuidos que provee a los sistemas

propiedades importantes como son: escalabilidad, modificabilidad y capacidad de despliegue. Además, que hacen que el código desarrollado se adapte fácilmente a automatización de infraestructura, entrega continua y DevOps.

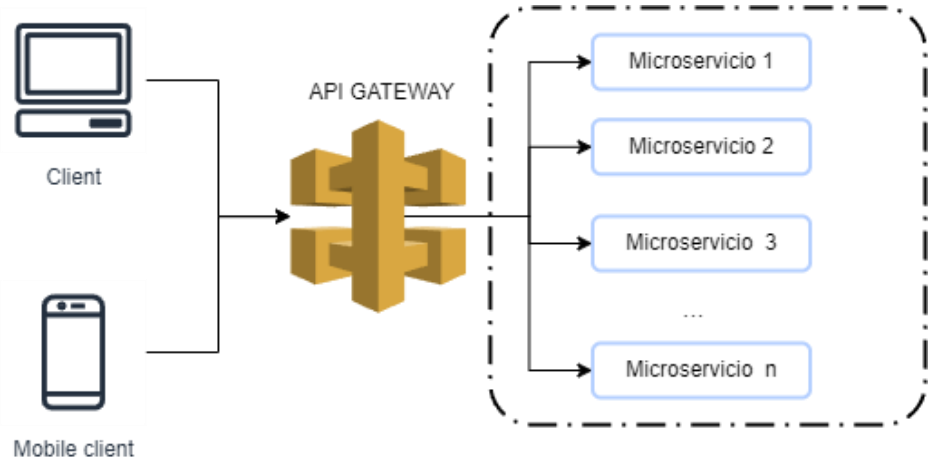


Figura 2. *Modelo básico de una arquitectura de microservicios*

4.2.1. Principios para el desarrollo de Arquitectura de microservicios

Con el tiempo y la experiencia de los desarrolladores, ha sido necesario establecer principios para el desarrollo de arquitecturas de microservicios o sistemas distribuidos, estos principios estandarizados o garantizan un producto escalable, reutilizables y fácil de usar, a continuación, se presenta una tabla resumen sobre estos principios [11].

Tabla 2. Principios para el desarrollo de arquitectura de microservicios

Nombre	Intención	Escenario de Uso	Beneficios
Diseño basado en el Dominio (DDD)	Diseñar y desarrollar sistemas de software basados en el modelo del dominio	Dominios complejos con una lógica empresarial rica y evolutiva	Lineamientos claros con el dominio, encapsulación del conocimiento del dominio, diseño mantenible y modular, mejora en la colaboración entre equipos
Patrón de detección de servicios	Facilitar un servicio dinámico de detección y comunicación	Arquitectura de Microservicios con interacción se servicios	Registro automático y dinámico de servicios, desacoplamiento entre servicios, balanceo de carga y tolerancia a fallos.
Diseño Basado en Datos (D-DD)	Diseñar sistemas de software en torno al comportamiento y estructura de datos	Aplicaciones donde la data es el principal componente	Se centra en la toma de decisiones basada en datos, la mejora de la integridad de los datos, la optimización del acceso a los datos y su manipulación, y la alineación con los objetivos de negocio

Patrón Backend para Frontend	Desarrollar servicios backend a adaptados para clientes frontend específicos	Aplicaciones con múltiples clientes frontend	APIs customizadas y optimizadas para cada cliente frontend, desacoplamiento mejorado backend – frontend, mejor experiencia de usuario
Patrón adaptador de microservicios	Adaptar y transformar datos o funcionalidades entre microservicios	Arquitectura de Microservicios con diferentes interfaces	Permitir una comunicación fluida entre microservicios con diferentes protocolos o formatos de datos, promover la interoperabilidad
Patrón de aplicación estranguladora	Migrar gradualmente de un sistema monolito a microservicios	Aplicaciones monolíticas heredadas	Migración gradual sin interrumpir el sistema existente, reduce el riesgo, mejora la mantenibilidad y escalabilidad
Patrón de diseño de microservicios con datos compartidos	Administrar y compartir datos entre microservicios	Arquitectura de Microservicios con requisitos de datos compartidos	Administración centralizada de datos, consistencia de datos, reducción de la duplicación y redundancia de datos, mayor integridad de datos
Patrón de diseño de microservicio agregador	Agregar datos o funcionalidades de múltiples microservicios	Aplicaciones que requieren consolidar información	Agregación centralizada de datos, menor complejidad para el cliente, rendimiento mejorado, reducir la sobrecarga de la red

4.2.2. Frameworks para Microservicios

Un marco de trabajo o framework, es una aplicación que tiene un conjunto de clases abstractas, permitiendo a un desarrollador crear aplicaciones reutilizables, haciendo subclases o creando instancias de las clases definidas por el framework [12].

En [13], nos señala que existen muchos marcos de trabajo para desarrollar arquitecturas de microservicios, sin embargo, por la degradación de rendimiento entre lenguajes se centran solo en KumuluzEE y Spring Cloud.

KumuluzEE

Es un framework que está escrito en Java con una licencia MIT10, creado por el esloveno Tilen Faganel en el año 2015, año en el que tuvo una gran aceptación por la comunidad. Para gestionar las dependencias este framework utiliza Maven, y como servidor web JBoss. Además, no hace uso de una clase main, ya que la gestión de los recursos y la aplicación esta englobada en acciones declarados en clases y funciones [13].

Spring Cloud

Este framework, así como el anterior también está escrito en Java, consta de una historia y soporte más extenso ya que está basado en Spring Boot. Para la gestión de dependencias, se puede optar por Maven o Gradle, tiene una licencia Apache 2.0. Es conocido por la migración a microservicios que realizó la empresa Netflix que en ese momento utilizaba Amazon Web Services y desde entonces, varias de las características de esa migración de adaptaron al Framework Spring, que hasta ese entonces no tenía soporte para microservicios [13].

En la **Tabla 3**, se muestra cifras de interacciones de los usuarios con los repositorios de KumuluzEE y Spring Cloud, mostrando que Spring Cloud tiene una mayor interacción y puntaje por parte de la comunidad.

Tabla 3. *Número de interacciones de KumuluzEE y Spring Cloud en GitHub*

Característica	KumuluzEE	Spring Cloud
Repositorios	1	75
Contribuidores Principales	4	6
Tareas (Issues)	12	1248
Pull Request	3	-
Commits	291	-
Releases	11	-
Stars	140	4015
Watches	33	4015
Forks	26	3275

4.2.3. Patrones de Diseño

4.2.3.1. Domain-Driven Design (DDD)

Es un patrón descrito por Erick Evans en [14], donde marca la importancia de comprender el Dominio, sus características y el trabajo en conjunto entre expertos en ese dominio y los desarrolladores.

Tabla 4*. Conceptos básicos de DDD*

Concepto	Definición
Entidad (Entity)	Representan elementos fundamentales del dominio, y cuya identidad es primordial para el modelo de negocio, es decir, es un objeto en el que su identidad perdura con el tiempo y puede variar sus estados.
Objetos de Valor (Value Object)	Permite modelar conceptos de dominio de una manera precisa, se distinguen por su "Inmutabilidad", en otras palabras, su estado no cambia, y si esto ocurre, se crea una nueva instancia.
Raíz del Agregado (Aggregate Root)	Es una entidad, constituida por un conjunto de objetos relacionados, esta entidad actúa como control para el acceso y modificaciones al resto de objetos que lo conforman.

Repositorio (Repository)	Proporciona una interfaz para acceder a un objeto de dominio, específicamente, a los Aggregate, además, encapsulan la persistencia y la obtención de objetos, de esta forma, se mantiene el código limpio.
Contexto Delimitado Bounded Context	Cada contexto delimitado tiene su propio modelo de dominio y su propia interpretación, los contextos delimitados permiten dividir sistemas grandes, en partes manejables reduciendo la complejidad.

Según [15], los principios básicos de DDD son:

- Capturar el conocimiento del dominio, incluyendo los aspectos de comportamiento y estructurales.
- Modelar el dominio en colaboración con expertos en ese dominio e ingenieros de software.
- Conforme se va resolviendo las dudas sobre el dominio, se puede ir implementando diseños temporales e irlos mejorando con el avance del proyecto.
- Es importante que la comunicación entre los expertos en el dominio y los desarrolladores se mediante un lenguaje explicito y ubicuo.

El patrón de diseño DDD ofrece técnicas para la identificación eficiente de los conceptos de dominio y los patrones de modelado. Por esto, es que se ha hecho popular dentro de la ingeniería de software [15]. A continuación, se lista las fases que se emplean para el diseño de una arquitectura de microservicios.

1) Comprender el Dominio
2) Definir contextos delimitados
3) Diseñar microservicios
4) Implementación Interna [16].

En DDD, se establece que cada Contexto Delimitado (Bounded Context), sea manejado por un equipo, y si los cambios en el proyectos o nuevos requerimientos ameritan que se modifique la estructura, solo el equipo designado puede hacerlo. Una vez revisados los principios y las fases que diferentes estudios plantean en DDD, seguidamente se detalla el proceso implementado en [15] para el diseño de una arquitectura de microservicios.

- En la primera fase de este proceso, se crea un diagrama de clases UML, esto con el fin de conocer y detallar los atributos y métodos de las entidades o clases que van a

conformar el sistema. Para este fin, se deben tomar en cuenta conceptos como Asociaciones, Multiplicidad y Relaciones.

- La segunda fase consiste en mejorar o enriquecer el diagrama obtenido en la primera fase con ayuda del metamodelo UML 2.5 y agregando conceptos de DDD como Aggregation Root, Repositories, Entities y Value Object. El objetivo de esta fase es obtener un diagrama que permita profundizar en las funcionalidades, los roles de cada entidad y dependencias entra cada una de ellas.
- En la tercera y última fase, se construye un nuevo diagrama denominado, Diagrama de Contextos Delimitados, este presenta los Bounded Context agrupados, es decir, cada Contexto Delimitado contiene diferentes funcionalidades con el fin de otorgarle una independencia de los demás contextos. Al final, cada Contexto Delimitado representa un microservicio a desarrollarse y dentro del mismo las funciones que va a ofrecer.

4.2.4. Comunicación para Microservicios

Como se lo había mencionado en los apartados anteriores, una arquitectura de microservicios consiste en servicios pequeños e independientes que funcionan de manera autónoma y que trabajan para ofrecer en conjunto funcionalidades eficientes que los usuarios requieran. Sin embargo, el hecho de que cada servicio este separado, hace que la implementación de una comunicación entre estos servicios sea un reto para los ingenieros de software. Según [17], existen dos tipos de comunicaciones para arquitecturas de microservicios o sistemas distribuidos.

4.2.4.1. Comunicación síncrona

La comunicación síncrona se basa en que un servicio realiza una solicitud para obtener información de otro servicio, y este detiene sus procesos hasta que el receptor responda la solicitud. Y esta es la principal desventaja de este tipo de comunicación, ya que, el hecho de solicitar datos a otro servicio provoca que los procesos internos del servicio solicitante se interrumpan. Existen varias opciones para implementar comunicación síncrona en una arquitectura de microservicios, pero los más comunes son:

- ***REST API:*** Consiste en permitir que cada microservicio pueda recibir y retornar información por medio de funciones o endpoint expuestos a través de puertos, habitualmente estos puertos son configurados por un servidor web y pueden ser el 8080 y 443 [17].

- ***gRPC:*** Google desarrolló esta herramienta Open Source con el fin de crear un servicio potente y eficiente en un escenario donde la comunicación entre los servicios tenga que soportar un tráfico demandante [18].

4.2.4.2. Comunicación asíncrona

La comunicación asíncrona se realiza por medio de un *Broker Message*, el cual actúa como un intermediario administrando los mensajes y respuestas entre los servicios, usualmente cubre aspectos como balanceo de carga y manejo de errores. La implementación de un Broker Message es la principal diferencia entre la comunicación síncrona y asíncrona [17].

RabbitMQ

Es uno de los Broker Messages más populares actualmente, se beneficia de la infraestructura de Erlang Open Telecom (OTP) ya que esta desarrollado en el lenguaje de programación Erlang. RabbitMQ implementa la comunicación a través del Protocolo Avanzado de Colas de Mensajes (AMQP), esto le permite tener un comportamiento transaccional y un soporte de transferencia asíncrona por lotes, estas características lo convierten en uno de los servicios de mensajería más eficientes y escalables.

Los *Exchanges* y *Queues* son los conceptos que RabbitMQ propone para la comunicación asíncrona entre microservicios. Los Exchanges actúan como enrutadores basados en reglas y criterios que utiliza para la redirección de los mensajes a las respectivas colas, por otro lado, los *Messages Queues* son mensajes que tienen propiedades y configuraciones importantes que el Consumidor considera al momento de procesar dichos mensajes [19].

Características

- Su principal objetivo es el manejo de mensajes en memoria DRAM.
- Se encarga de la contabilización del consumo de recursos para ofrecer un mejor balanceador de carga.
- Si los mensajes se acumulan el rendimiento se empieza a degradar.

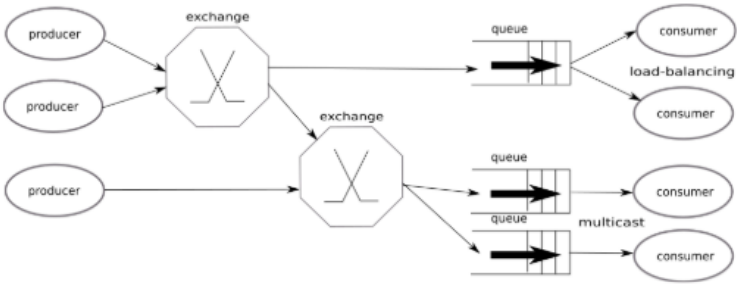


***Figura 3**. Arquitectura de RabbitMQ*

Apache Kafka

En el año 2011 LinkedIn lanza el proyecto Apache Kafka como una opción para servicios de mensajería, este proyecto se creó con el fin de dar solución al problema de manejo de eventos en su plataforma centralizada. Apache Kafka fue diseñado para sistemas con alta demanda de procesamiento en tiempo real, los datos se escriben en un conjunto de archivos de registro sin descarga inmediata en el disco para una mejor performance en operaciones de E/S, dando como resultado el poder manejar millones de mensajes con múltiples consumidores. Como la mayoría de los servicios de mensajería, Apache Kafka está conformado de publicadores y suscriptores que garantizan una gran eficiencia.

En la **Figura 4**, se señala los principales componentes que intervienen en la arquitectura de Apache Kafka y cómo se implementa la comunicación entre estos.

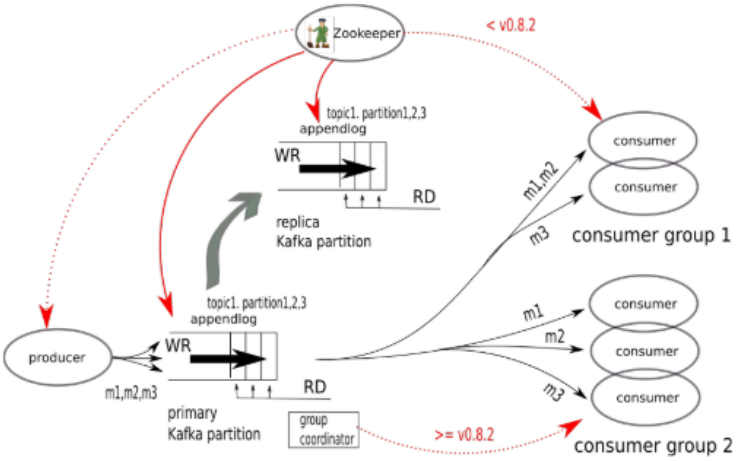


***Figura 4.** Arquitectura de Apache Kafka*

Ya que RabbitMQ y Apache Kafka son servicios de mensajería para comunicación asíncrona, a continuación, en la **Tabla 5**, se puede visualizar una comparativa entre estas dos opciones.

Tabla 5. *Tabla comparativa entre Apache Kafka y RabbitMQ*

Características	Apache Kafka	RabbitMQ
Desarrollado en	Scala	Erlang
Año de Inicio	2011	2007
Modelos de mensajería soportados	Pub/Sub Message queue	Pub/Sub Message Queue Request reply
Con intermediario	Si	Si
Rendimiento	Alto	Medio -Alto
Latencia	Baja	Baja -Media
Protocolos soportados	TCP	AMQP, STOMP, MQTT
Tamaños de mensajes	1MB máximo	2GiB
Envío de mensajes	Como mucho una vez, Exactamente una vez, Al menos una entrega	Como máximo una vez, Al menos una entrega
Lenguajes soportados	Alrededor de 17 lenguajes	Alrededor de 30 lenguajes
Ordenamiento de mensajes	Si	Si
Almacenamiento de mensajes	Disco	En memoria y Disco
Unidades distribuidas	Topics	Queues
Usados por	LinkedIn, Netflix, Facebook, Twitter, Chase Bank	Mozilla, AT & T, Reddit, Instagram, T-Mobile

4.3. DevOps

DevOps representa la interconexión entre los desarrolladores de software y el equipo de operaciones, es un conjunto de prácticas y actividades que permiten mejorar la eficiencia en el proceso de desarrollo de software, es decir, optimiza los tiempos de despliegue y pruebas al momento de lanzar un entregable o una actualización del sistema. Para mejorar el ciclo de vida de un proyecto de software, es recomendable que los procesos de DevOps de apliquen de manera constante y repetitiva [20].

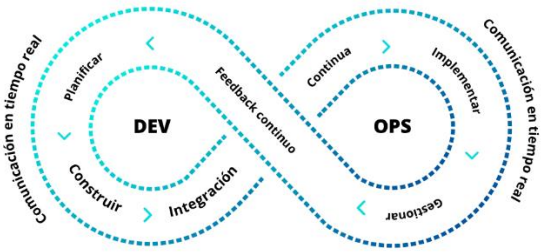


Figura 5. *Procesos que intervienen en DevOps*

Si bien la cultura DevOps ha beneficiado a la industria de software, mejorando aspectos importantes en el ciclo de vida del desarrollo, hay que tener en cuenta que los procesos y técnicas que se aplican al proyecto atraen también retos o dificultades, es por ello, que en la, se listan los beneficios y retos que se tiene que contemplar al momento de implementar DevOps.

Tabla 6. *Beneficios y retos de DevOps*

Beneficios	Retos
-Control de versiones de código proporcionado por Bitbucket. -Despliegue paralelo (Codeship). Habilitar despliegue programado, por ejemplo, Codeship. -Habilitar pruebas programadas (Jenkins, Codeship). -Proveer pruebas automatizadas de QA mediante comandos de prueba de Codeship o plugins de Jenkins. -Proveer integración continua (Codeship). -Proveer monitoreo automatizado en tiempo real. -Proveer comunicación en tiempo real mediante HipChat. -Usar servicios de AWS y MongoDB para proveer gestión en la nube. -Por medio de la Retroalimentación de QA se puede ofrecer reversión de código y mejor planificación. -Desarrollo de nuevas ideas e innovaciones continuas. -Realizar entregables de manera rápido gracias al ciclo, construcción, pruebas y despliegue. -Permitir escalar los recursos sin dar de baja a los servicios. -Mantener la visibilidad de los pipelines -Mejoras a partir de los logs del sistemas y monitoreo de recursos (Codeship, AWS, Bitbucket, etc.). -Asegurar el pipeline utilizando autenticación de herramientas. -Despliegue automatizado en la nube (AWS, Heroku). -Procesos escalables, repetibles y automatizados.	**-Superar la mentalidad de Dev vs. Operaciones** **-Migrar arquitecturas monolíticas a microservicios:** Las migraciones de monolitos a microservicios pueden representar un reto enorme para los ingenieros de software. La infraestructura como código es el futuro de los sistemas de alto rendimiento. **-Darles excesiva importancia a herramientas:** Para la implementación de un pipeline es necesario un gran número de herramientas. La dificultad en este punto es el aspecto de cómo se mantiene y la dificultad de integrar estas herramientas. **-Miedo al cambio:** Para una empresa puede parecer que la implementación de DevOps a sus procesos puede resultar en una tarea agotadora e interminable. La empresa debe saber cómo establecer los procesos y la mejor forma es presentarlo como una herramienta que permitirá un ahorro de tiempos. **-Dificultad al integrar herramientas Devs y Ops:** En algunos escenarios puede que el equipo de desarrolladores y el equipo de operaciones tenga herramientas totalmente separadas, haciendo que la integración entre estas sea un problema tedioso de resolver.

4.3.1. Técnicas

Las técnicas de DevOps individualmente tratan de resolver problemas específicos en los proyectos de software, es decir, procuran dar soluciones a aspectos que retrasan los proyectos [21]. A continuación, se describen las diferentes técnicas que la cultura DevOps ofrece.

4.3.1.1. Control de Versiones de Código

Si bien el control de versiones de código ya es una técnica ampliamente utilizada por las empresas de desarrollo de software, es considerada una técnica muy importante y tomada como una buena práctica de programación entre los desarrolladores. Para implementar el control de versiones la herramienta más utilizada es Git, que se encarga de registrar y dar seguimiento a cada una de las versiones de los archivos del proyecto, dando la posibilidad de revertir cambios a versiones anteriores. El control de versiones permite el trabajo colaborativo de varios desarrolladores en un mismo proyecto [22].

4.3.1.2. Integración y Despliegue Continuo (CI/CD)

La Integración Continua (CI) es una técnica que tiene como objetivo la creación de código de calidad, y permitiendo liberar al desarrollador de ciertas incertidumbres con respecto al comportamiento del código. CI propone que un desarrollador debe realizar fusiones de su código de manera contantes y con la mayor regularidad posible, esto con el fin de que el código se vaya probando como un todo y así poder reconocer problemas de funcionalidad de manera temprana [23].

El Despliegue Continuo (CD) es una práctica del desarrollo de software que utiliza el proceso de CI para permitir lanzamientos o entregables más frecuentes [24].

GitLab

Es una plataforma web para flujos de trabajo de DevOps, permite gestionar procesos como CI/CD, ofrece control de versiones de código y un módulo de para seguimiento de hitos y problemas en proyectos de software. Además de ofrecer los servicios antes mencionados, GitLab ofrece GitLab Runner, para gestionar la automatización de compilación y de pruebas, su instalación generalmente se lo realiza en servidores remotos que se encargan de procesar o ejecutar los comandos necesarios para CI/CD, se puede configurar múltiples instancias de este servicio en servidores separados [23].

GitHub

Plataforma que permite el control de versiones y trabajo colaborativo entre equipos de desarrolladores, con el tiempo ha ido adaptando soluciones a los problemas de compilación y

testeo en grandes proyectos de software por medio de herramientas de CI/CD. Su API basada en eventos conocida como GitHub Actions, permite ejecutar comandos tras producirse eventos como: commits, issues y pull request. Estos comandos se definen en archivos YAML y son ejecutados en servidores externos [25].

Jenkins

Software de código abierto con licencia MIT, se encuentra disponible como paquetes para sistemas nativos, Docker, y ejecutables utilizando Java Runtime Environment. Su pipeline es un conjunto de plugins desarrollados para soportar procesos de CI/CD, su archivo *Jenkinsfile* se tiene que construir con la sintaxis DSL. Entre sus principales ventajas tenemos, compilaciones automáticas en diferentes ramas, registros de auditoría [26].

4.3.1.3. Monitoreo

El monitoreo es un aspecto importante dentro de la cultura DevOps, ya que permite conocer el estado de los servicios o los contenedores que se estén ejecutando en el servidor. Los softwares de monitorización más comunes para contenedores son: Docker stats, cAdvisor, Sysdig, Prometheus.

Prometheus

Es un proyecto de código abierto desarrollado en GO, se basa en la captura periódica del estado de los componentes de un sistema a través del protocolo HTTP. También permite trabajar con otro tipo de herramientas como cAdvisor, que es un componente que recopila datos de los contenedores en ejecución, es decir, provee de métricas del estado se los servicios. Su escalabilidad, su lenguaje de alta flexibilidad, trabajo independiente y su interfaz gráfica configurable hacen que Prometheus sea una de las opciones más rentables para proyectos de software [27].

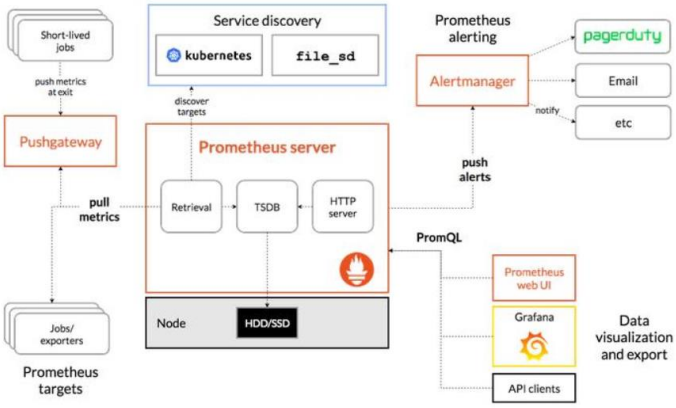


***Figura 6**. Proceso de trabajo de Prometheus*

Grafana

Plataforma de código abierto para recolección y visualización de métricas, además de permitir establecer alertas, generalmente Grafana se combina con otras herramientas que lo prevén de métricas, por ejemplo, Google Cloud Monitor, Elasticsearch, InfluxDB y Prometheus. Una de sus características más importantes es la creación de dashboards reutilizables y personalizables por medios de archivos JSON o desde su sitio web, permitiendo a sus usuarios compartir los dashboards de manera fácil y rápida. También permite en envió de alertas a otros sistemas como Microsoft Teams, Google, Discord y Email [28].

4.3.1.4. Testing

En DevOps, el testing consiste en validar las funciones codificadas en un proyecto de software en un escenario controlado, esta técnica permite automatizar las pruebas y ejecutarlas de una manera continua mientras el proyecto va avanzando. Esta técnica favorece aspectos como la seguridad, rendimiento, pruebas de carga y se puede comprobar que la infraestructura se encuentre en buen estado [29].

4.3.1.5. Construcción y Registro

La técnica de Construcción consiste en automatizar por medio de un pipeline la creación de los ejecutables de los servicios, por ejemplo, para la ejecución de una aplicación desarrollada en Java, se necesita obtener un .JAR a partir del código fuente. Y el Registro, permite almacenar información sobre posibles errores o eventos ocurridos durante la ejecución del pipeline, de esta forma se puede obtener datos sobre funciones que no estén ejecutándose de manera correcta.

4.3. Contenedores

Los contenedores son una tecnología creada con el fin de que a cada proceso (contenedor) se le pueda asignar recursos como procesador y memoria de forma individual. El Sistema Operativo (SO) ejecuta los contenedores en un conjunto de procesos espaciados, esto garantiza que la administración por parte del SO sea rápida y eficiente, además, la teoría indica que los contenedores pueden ejecutarse independientemente del SO [30].

4.3.2. Docker

Docker Inc en al año 2013 lanza su tecnología Docker con el fin de ofrecer una alternativa para crear y ejecutar contenedores en SO basados en Linux. Los contenedores Docker se pueden definir como un software en el cual se encuentra empaquetado el ejecutable de una aplicación y todos los recursos necesarios para su ejecución. Para la construcción de contenedores Docker

es necesario obtener un archivo al que se lo denomina *imagen*, este archivo es ligero e independiente y en la práctica, es el elemento principal para la ejecución de un contenedor [31].

4.3.2.1. Dockerfile

Antes de construir y ejecutar un contenedor Docker, se puede personalizar ciertos aspectos como comandos, instalación de dependencias y copia de recursos necesarios. Esta personalización se la realiza por medio de un archivo llamado *Dockerfile*, en este archivo se puede definir todos los comandos para construir una imagen, sin embargo, el archivo Dockerfile solo reconoce los siguientes comandos: ADD, CMD, ENTRYPOINT, ENV, EXPOSE, FROM, MAINTAINER, RUN, USER, VOLUME, WORKDIR [30].

4.3.3. Docker Compose

Una de las herramientas más comunes para organizar y gestionar contenedores Docker es Docker Compose, esta herramienta utiliza archivos YAML donde se definen los servicios y componentes a Dockerizar. Docker Compose permite que el despliegue de servicios o arquitecturas sea rápida y fácil, ya que el despliegue de lo puede realizar por medio de un solo comando, otorgando independencia y sin necesidad de clonar todos los repositorios de los servicios [32].

Como se mencionaba anteriormente, el archivo principal de Docker Compose, es el archivo YAML, el cual está conformado de 3 secciones esenciales:

- ***Servicios:*** En esta sección se define las propiedades que van a estructurar el contenedor, propiedades como imágenes, puertos y healthchecks.
- ***Redes:*** Esta sección es donde se define las redes que permitirán establecer separar y establecer la comunicación entre los servicios y hosts dentro de Docker Compose.
- ***Volúmenes:*** La sección de volúmenes es donde se establece los directorios que permitirán el almacenamiento de los datos que se creen y compartan entre los servicios, en otras palabras, es la sección de persistencia de datos [33].

4.4. Servicios Backend

A los servicios backend se los puede definir como la infraestructura o la parte que no se puede visualizar por parte del usuario final, en otras palabras, son los servicios que procesan la información de un sistema, generalmente se ejecutan en servidores remotos y que son accesibles desde cualquier parte del mundo. El desarrollo de estos servicios se lo puede hacer utilizando

lenguajes de programación como PHP, Ruby, Python, Java, etcétera. Un servicio backend está compuesto de tres partes esenciales:

- Servidor
- Aplicación
- Base de datos [34].

4.5. Metodologías Ágiles

Esta metodología hace frente a los problemas que las metodologías tradicionales pueden provocar, es decir, permite que el proceso de desarrollo de software sea adaptativo a cambios no previstos o problemas repentinos o no planificados. Una de sus principales características es realizar entregables frecuentes en tiempos cortos, así como también, mejora la comunicación entre clientes y desarrolladores. Todas las metodologías ágiles proporcionan principios o pasos con el fin de garantizar que la entrega del producto final sea menos complicada, con poca documentación [35].

Tabla 7. *Comparativa entre Metodologías Ágiles y Tradicionales*

Metodologías Ágiles	**Metodologías Tradicionales**
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas de normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente por el equipo	Impuestas externamente
Procesos menos controlados, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo por medio de reuniones
Grupos pequeños (< 10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Procos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura de software	La arquitectura de software es esencial y se expresa mediante modelos
Poca documentación	Documentación exhaustiva
Muchos ciclos de entrega	Pocos ciclos de entrega

4.5.1. Manifiesto Ágil

El manifiesto ágil consiste en un conjunto de principios del desarrollo ágil. Donde se prioriza:

- A las interacciones del equipo de desarrollo, sobre los procesos y herramientas.

- El obtener un producto final que sea funcional, y no una documentación extensa.
- Al cliente, sus interacciones y colaboraciones al equipo de desarrollo, más que un contrato.
- Responder a los cambios que va sufriendo el proyecto, más que seguir un plan [36].

4.5.2. Programación Extrema (XP)

Esta metodología fue desarrollada por Kent Beck en 1999, recibe su nombre por tomar las mejores prácticas y llevarlas al extremo, maneja los requisitos crecientes y cambiantes al largo del proceso de software [37].

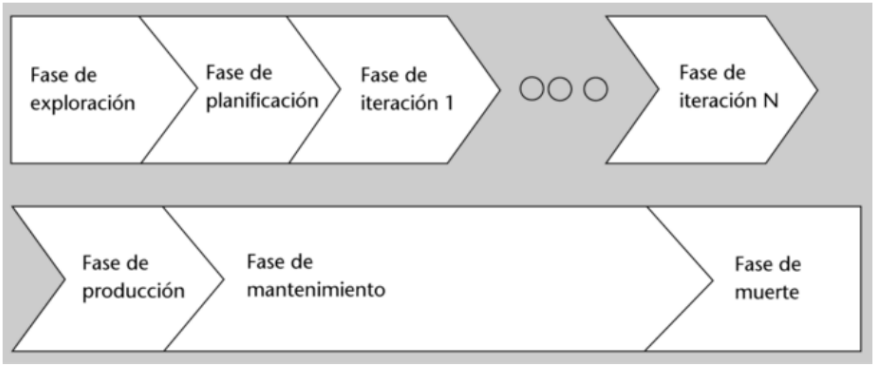


Figura 7. *Fase de Programación Extrema (XP)*

Fases de la metodología XP

En XP tenemos 4 fases, las cuales se describen a continuación [38]:

Fase 1. Planificación

Este es el punto donde se empieza a levantar los requisitos en conjunto con el cliente, es decir, el equipo de desarrollo también es donde se calcula el tamaño de las iteraciones y los tiempos según el tamaño y alcance del proyecto.

Fase 2. Diseño

En esta metodología, las historias de usuario son las que se diseñan, ya que, en XP se considera que no se puede diseñar el sistema completo y ya que el proyecto esta propenso a cambios, no es rentable realizar un diseño completo del sistema.

Fase 3. Codificación

Esta fase se la puede ir ejecutando a la par con la fase de diseño, y una de sus características, es que se vaya realizando en parejas de programadores.

Fase 4. Pruebas

En esta fase se toma en cuenta dos tipos de pruebas, las Pruebas Unitarias y las Pruebas de Aceptación. Las pruebas unitarias consisten en codificar métodos que verifiquen el correcto funcionamiento de los servicios del proyecto, en la metodología XP se considera como buena práctica que las pruebas unitarias se implementen antes de programar la funcionalidad. Finalmente, las pruebas de aceptación se las realiza con el cliente, revisando que los requerimientos propuestos o historias de usuarios se hayan alcanzado satisfactoriamente [38].

4.5.3. Scrum

La metodología Scrum es muy utilizada en proyectos de desarrollo de software, la implementación de esta metodología se considera un gran reto para una organización, ya que, es necesario definir nuevos roles y la entrega del código es diferente, presta menos atención a procesos, herramientas y prácticas, enfocándose o dándole más valor al software funcional [37].

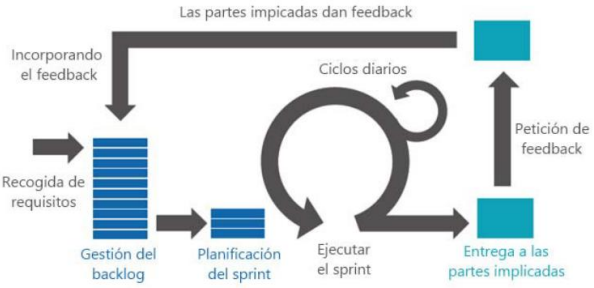


Figura 8. *Fase de Scrum*

Fases de la metodología Scrum

FASE 1. Planificación de la iteración

Se divide en dos partes fundamentales, la primera donde el cliente establece una lista de requisitos ordenado por prioridad, en la segunda parte, el equipo de desarrollo planifica como abordar estos requisitos y encargando a las personas más adecuadas para cada tarea.

FASE 2. Ejecución de la iteración

En Scrum, cada iteración (Sprint) tiene plazos cortos y fijos, generalmente de 2 a 4 semanas, en este tiempo, el equipo se compromete a realizar un entregable que completo y que sume al producto final.

FASE 3. Reunión diaria con el equipo

En las reuniones diarias interviene todos los miembros del equipo, esto se lo realiza con el fin de conocer los avances del día anterior, problemas que se hayan presentado, además, se puede dar respuesta de manera conjunta interrogantes del equipo.

FASE 4. Demostración de requisitos completados

En esta fase, el equipo presenta los requisitos completados al cliente en el Sprint, además, el cliente puede presentar adaptaciones al proyecto para una replanificación.

FASE 4. Retrospectiva

El equipo se reúne para debatir de una manera objetiva el por qué se están o no se están cumpliendo con los objetivos, como ha sido el rendimiento durante el Sprint, como mejorar la calidad y productividad [35].

5. Metodología

En esta sección, se detallan las actividades, métodos y materiales utilizados para el desarrollo del Trabajo de Titulación (TT).

5.3. Área de Estudio

El TT se lo realizó en el periodo de abril-julio de 2024 con la empresa Consultora PuntoPymes CIA. LTDA., que está localizada en la ciudad de Loja, en las calles Azuay 12-57 entre Bernardo Valdivieso y José Joaquín de Olmedo.

5.4. Procedimiento

Este apartado detalla los procedimientos y las actividades cumplidas en los objetivos propuestos en el TT.

5.4.1. Aplicar técnicas de DevOps en los servicios backend de la plataforma GeoWorker mediante CI/CD, Docker, Docker Compose y la metodología XP.

Para cumplir este objetivo se estableció los requisitos de la plataforma, se diseñó la arquitectura para su codificación y posteriormente se implementó las técnicas de DevOps a los servicios backend codificados. Las actividades ejecutadas son las siguientes:

- Establecer los requerimientos en con el encargado del departamento de desarrollo de software de la empresa PuntoPymes.
- Diseñar la arquitectura con el patrón de diseño Domain-Driven Design.
- Elaborar el documento de arquitectura de software.
- Codificar los servicios backend utilizando el framework Spring Cloud y combinando tecnologías como: Spring-Boot, NestJS.

- Implementar técnicas de DevOps tales como: Despliegue continuo, Control de versiones, monitoreo, testeo, construcción y registro.
- Desplegar los servicios backend en una IP y/o dominio público con ayuda de Docker Compose.

5.4.2. Evaluar los servicios backend en un escenario experimental

Esta fase tuvo como objetivo el testeo de las funcionalidades y servicios desarrollados de la plataforma con el fin de verificar su rendimiento y comprobar el correcto funcionamiento de la infraestructura implementada. Las actividades cumplidas fueron las siguientes:

- Codificar las pruebas unitarias para cada una de las funciones principales de los servicios backend de la plataforma.
- Verificar el rendimiento de los servicios en un ambiente simulado y controlado utilizando JMeter.

5.5. Recursos

Para alcanzar el objetivo propuesto en el TT, se utilizaron los siguientes recursos:

5.5.1. Recursos Científicos

- **Entrevista:** Por medio de esta técnica se recolecto la información necesaria para el presente Trabajo de Titulación, la entrevista se la realizó al Sr. Jonathan Zhunaula, Gerente de la empresa PuntoPymes y encargado del proyecto de la plataforma "GeoWorker", permitiendo identificar los requerimientos básicos, el modelo de negocio y el alcance del mismo.
- **Investigación Bibliográfica:** Esta técnica se utilizó para la recolectar de la información científica necesaria para sustentar el presente TT, las fuentes bibliográficas utilizadas fueron: artículos, libros, tesis y revistas científicas.

5.5.2. Recursos Técnicos

- **Modelo de arquitectura 4+1:** Este modelo permitió diseñar la arquitectura de la plataforma, y así, alcanzar el primero objetivo específico.
- **Metodología XP:** Se empleo para el desarrollo de los servicios de la plataforma, ejecutando sistemáticamente cada una de sus fases cumpliendo las iteraciones planeadas.

- **Patrón de diseño Domain-Driven Design:** Por medio de este patrón, se profundizo en las entidades de la plataforma y como resultado se obtuvo el diagrama de los microservicios a codificar.

5.5.3. Estándares

- **IEEE 830**: Este estándar fue utilizado para definir los requerimientos funcionales y no funcionales de los servicios.

5.5.4. Recursos de Hardware y Software

Hardware

- **Laptop Alienware Core i7:** Equipo utilizado para realizar la búsqueda de información, así como también el desarrollo de los servicios backend de la plataforma "GeoWorker"

Software

- **Mendeley Reference Manager:** Software utilizado para la gestión de citas bibliográficas.
- **Visual Studio Code:** Programa informático que brinda extensiones y plugins para la codificación de proyectos de desarrollo de software.
- **Amazon Web Services:** Plataforma de servicios en la nube que permitió el almacenamiento de archivos y contratar servidores para pruebas y despliegue de los servicios backend.

5.6. Participantes

En el presente TT, intervinieron los siguientes participantes:

- Manuel Alejandro Aguinsaca Medina como autor del presente TT. Ha estado involucrado desde la fase del planteamiento del tema, hasta alcanzar los objetivos planteados.
- Ing. Edwin René Guamán Quinche, como tutor académico y director del presente TT. Se encargó de la orientación, y de las revisiones en los avances del proyecto.
- Ing. Valeria del Rosario Herrera Salazar, en calidad de docente de la asignatura de Metodología de la Investigación, facilito los recursos didácticos para la estructuración del presente TT.

- Sr. Jonathan Fernando Zhunaula Angamarca, en su rol como Gerente de la empresa PuntoPymes, brindo la información necesaria de los requerimientos de la plataforma “GeoWorker”.

6. Resultados

En este apartado se detalla los resultados obtenidos de implementar una arquitectura de microservicios y técnicas de DevOps a los servicios backend en la plataforma GeoWorker.

6.1. Aplicar técnicas de DevOps en los servicios backend de la plataforma GeoWorker mediante CI/CD, Docker, Docker Compose y la metodología XP.

Para cumplir con este objetivo, se aplicó todas las fases de Ingeniería de Software según la metodología XP. La planificación, diseño, codificación, pruebas y despliegue fueron las fases que aplicadas en el desarrollo del TT. Por último, se implementó técnicas de DevOps a los servicios de autenticación, de gestión de empresas, usuarios, proyectos y tareas, así como también, a los servicios que administran los horarios, geocercas y la auditoria del sistema. Las técnicas aplicadas a los servicios backend fueron: CI/CD, Monitoreo, Construcción, Control de versiones de código y automatización de pruebas. A continuación, se detallan las actividades que se llevaron a cabo en este objetivo:

6.1.1. Definir los requisitos y expectativas de los usuarios de la plataforma GeoWorker a través de entrevistas.

En esta actividad, se ejecutó la fase de planificación de la Metodología XP, se entrevistó al Gerente de la Empresa PuntoPymes, con el fin de profundizar en los requisitos del sistema, conocer el modelo de negocio y las entidades y funcionalidades principales que la plataforma va a ofrecer a sus Usuarios. La empresa entregó el documento de requisitos establecidos en la entrevista (Ver **Anexo 1.** Requerimientos entregados por la empresa PuntoPymes CIA. LTDA.) y se agruparon las actividades en épicas para cada iteración. En la primera iteración, con una duración de dos semanas, se desarrolló la gestión de Usuarios y Empresas, seguidamente, en la segunda iteración centrada en la gestión de proyectos, tareas, roles y horarios, esto en un lapso de 2 semanas, y, por último, se desarrolló la gestión de permisos y logs del sistema en un plazo de 1 semana. La **Tabla 8**, muestra el resultado de la planificación de iteraciones a efectuarse para el cumplimiento de los requisitos solicitados.

Tabla 8. *Planificación de iteración de épicas*

Iteración	Fecha Inicio	Fecha fin	Épica
1	2024-04-15	2024-05-03	Gestión de Usuarios
2			Gestión de Empresas
3	2024-05-06	2024-05-24	Gestión de Proyectos
4			Gestión de Tareas
5			Gestión de Roles

6			Gestión de Horarios
7	2024-05-27	2024-06-03	Gestión de Permisos
8			Gestión de Logs

En la siguiente etapa, se elaboró el documento de requisitos de Software mediante el estándar IEEE 830 (Ver **Anexo 2.** Especificación de Requisitos de software), aquí se obtuvo los Requisitos Funcionales y No Funcionales del sistema, y, a partir de estos, se generó las Historias de Usuarios.

Requerimientos Funcionales

***Tabla 9**. Requerimientos Funcionales de la Plataforma*

Requerimientos Funcionales	
Código	**Requisito**
RF001	El sistema permitirá registrar empresas
RF002	El sistema permitirá modificar la información de las empresas
RF003	El sistema permitirá listar las empresas
RF004	El sistema permitirá eliminar empresas
RF005	El sistema permitirá al encargado de la empresa registrar usuarios
RF006	El sistema permitirá al encargado de la empresa modificar usuarios
RF007	El sistema permitirá al encargado de la empresa listar sus usuarios
RF008	El sistema permitirá al encargado de la empresa eliminar sus usuarios
RF009	El sistema permitirá al encargado de la empresa cargar usuarios desde un archivo .csv
RF010	El sistema permitirá al encargado de la empresa crear proyectos
RF011	El sistema permitirá al encargado de la empresa listar proyectos
RF012	El sistema permitirá al encargado de la empresa modificar proyectos
RF013	El sistema permitirá al encargado de la empresa eliminar proyectos
RF014	El sistema permitirá al encargado de la empresa asignar un usuario como líder del proyecto
RF015	El sistema permitirá al líder del proyecto listar los usuarios que pertenecen al proyecto
RF016	El sistema permitirá al líder del proyecto crear tareas a los usuarios asignados al proyecto
RF017	El sistema permitirá al líder del proyecto eliminar tareas
RF018	El sistema permitirá a los usuarios modificar los datos de una tarea
RF019	El sistema permitirá a los usuarios visualizar los datos de una tarea
RF020	El sistema permitirá al líder del proyecto listar las tareas que pertenecen al proyecto
RF021	El sistema permitirá al encargado de una empresa crear geocercas
RF022	El sistema permitirá al encargado de una empresa modificar geocercas
RF023	El sistema permitirá al encargado de una empresa listar las geocercas
RF024	El sistema permitirá al encargado de una empresa asignar una geocerca a un usuario

RF025	El sistema permitirá visualizar los usuarios que estén dentro de una geocerca
RF026	El sistema permitirá a un usuario solicitar permiso o vacaciones
RF027	El sistema permitirá responder solicitudes de permisos o vacaciones
RF028	El sistema permitirá al encargado de la empresa definir cuantas solicitudes puede crear un usuario mensualmente
RF029	El sistema permitirá registrar la entrada o salida de la jornada laboral
RF030	El sistema permitirá modificar el registro de entradas y salidas de un usuario
RF031	El sistema permitirá visualizar el reporte de horas trabajadas de un usuario
RF032	El sistema almacenará las acciones que realicen los usuarios
RF033	El sistema permitirá al usuario iniciar sesión mediante correo y contraseña

Requerimientos No Funcionales

Tabla 10. *Requerimientos No Funcionales de la Plataforma*

Requerimientos No Funcionales		
Número	**Nombre**	**Descripción de Requerimiento**
RNF001	Seguridad	Los servicios de la plataforma GeoWorker, serán capaces de controlar el acceso a las funcionalidades por medio de permisos y roles.
RNF002	Disponibilidad	Los servicios de la plataforma deben funcionar las 24 horas del día y los 7 días de la semana, a menos que se encuentren en mantenimiento o actualización.
RNF003	Usabilidad	Los servicios deben retornar los respectivos mensajes de error e información, de esta forma, ayudar al usuario a guiar en los procesos que este ejecutando.

Seguidamente, se procedió a determinar los tipos de usuarios que van a interactuar con la plataforma, los cuales se detallan en la **Tabla 11**.

Tabla 11*. Usuarios que utilizarán la plataforma*

Usuarios del Sistema	
Tipo de Usuario	Descripción
Usuario administrador	Administración y soporte de plataforma
Encargado de la Empresa	Administración de los datos de su empresa, usuarios, horarios, roles, proyectos y tareas
Usuario del Sistema	Gestión de tareas, permisos, y horarios

6.1.2. Diseñar una arquitectura de microservicios para los servicios backend.

En esta actividad se implementó el patrón Domain-Driven Design (DDD) para diseñar la arquitectura e identificar los microservicios que la van a conformar. El patrón DDD permitió definir el diagrama de Contextos Delimitados (Ver **Figura 9**), con sus servicios, entidades, relaciones y repositorios. Para visualizar el procedimiento completo ver **Anexo 3.** Diseño de la Arquitectura de Microservicios.

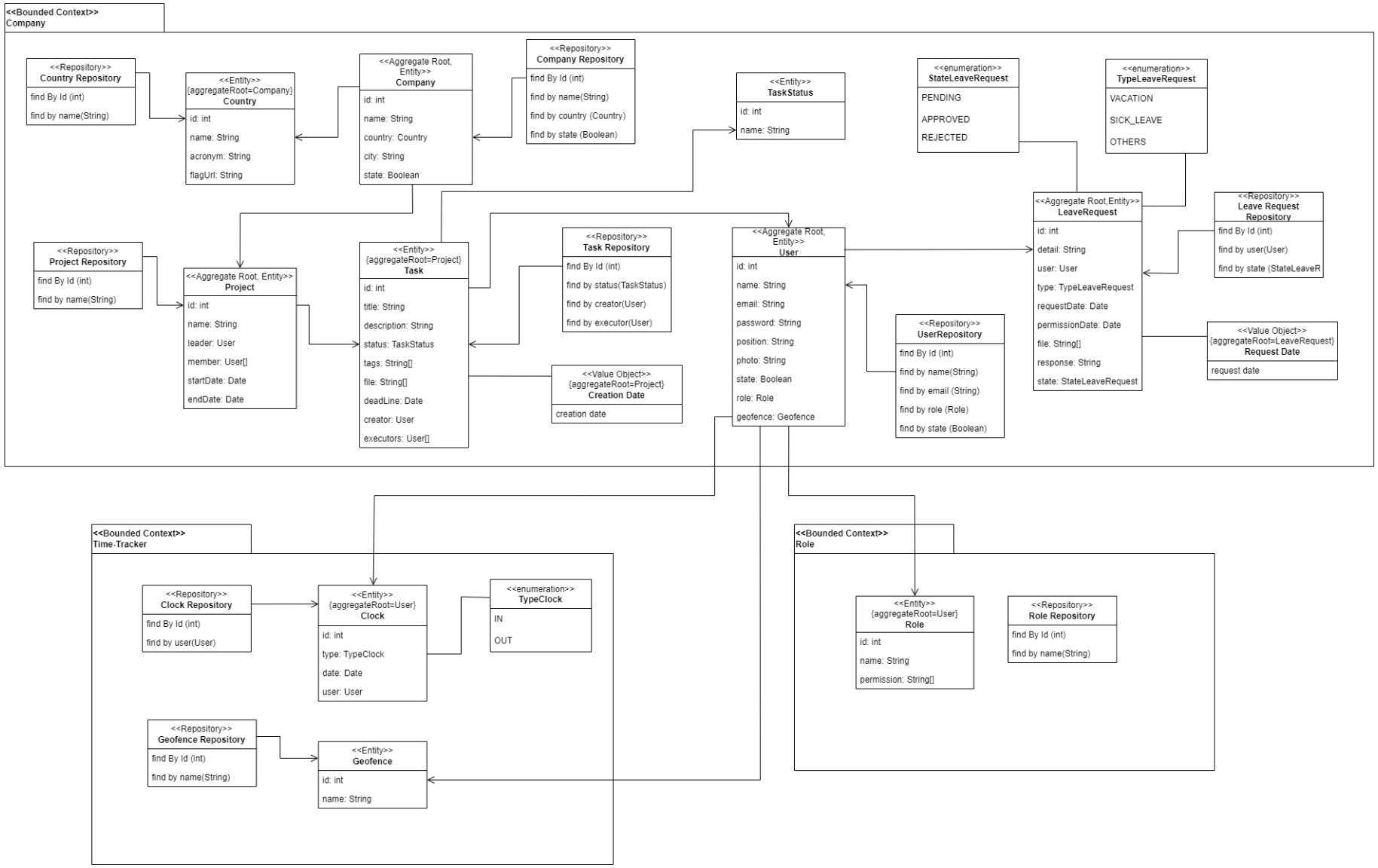


Figura 9. *Diagrama de Contextos Delimitados de la plataforma "GeoWorker"*

6.1.3. Codificar los servicios backend en base a los requerimientos establecidos.

Para la codificación de los servicios backend, fue necesario elaborar el documento de Arquitectura de Software basado en el modelo 4+1 (Ver **Anexo 4.** Documento de Arquitectura de Software), las diferentes vistas que componen el modelo antes mencionado ayudaron a tener una mejor perspectiva de las entidades que componen el sistema, sus procesos y funcionalidades. En cada iteración, se realizaron los diagramas de clases respectivos, al mismo tiempo, se fueron implementando las entidades relacionadas y las funcionalidades de mayor relevancia en cada fase. En la **Figura 10**, se puede visualizar el diagrama de Casos de Uso, que permitió visualizar las interacciones entre los usuarios y las funcionalidades que ofrece la plataforma "GeoWorker".

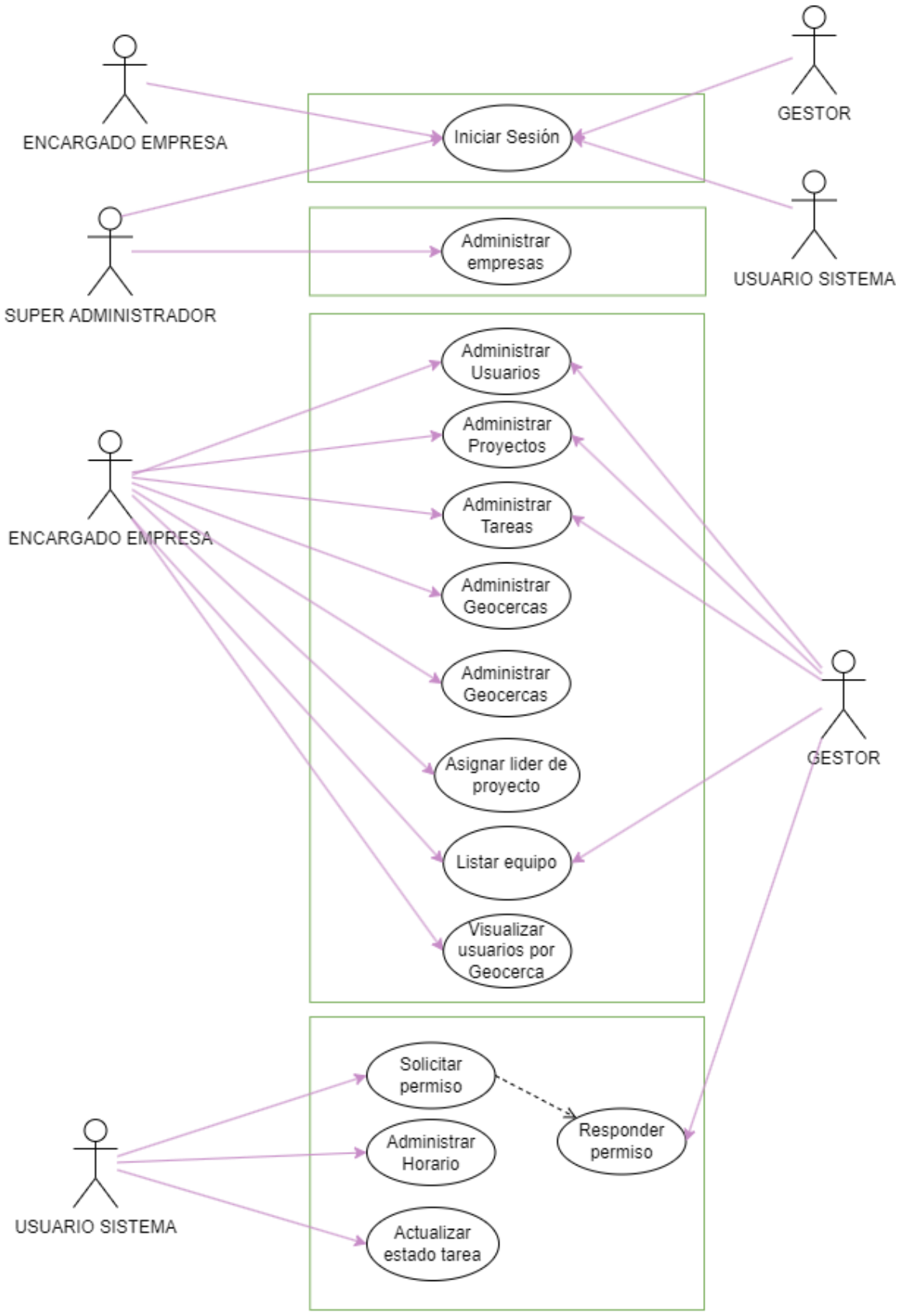


Figura 10. *Diagrama de Casos de Uso*

De la misma forma, en la **Figura 11**, se presenta un diagrama de la arquitectura final del sistema, que es el resultado obtenido de la elaboración del documento de Arquitectura de software que ese encuentra en el **Anexo 4.** Documento de Arquitectura de Software. En este diagrama se puede visualizar todos los componentes de la arquitectura implementada, es decir, los microservicios y las tecnologías que se usó para desarrollarlos, las bases de datos utilizadas, los componentes de Spring Cloud para el control de la arquitectura y como el servicio de mensajería RabbitMQ se conecta a todos los componentes antes mencionados.

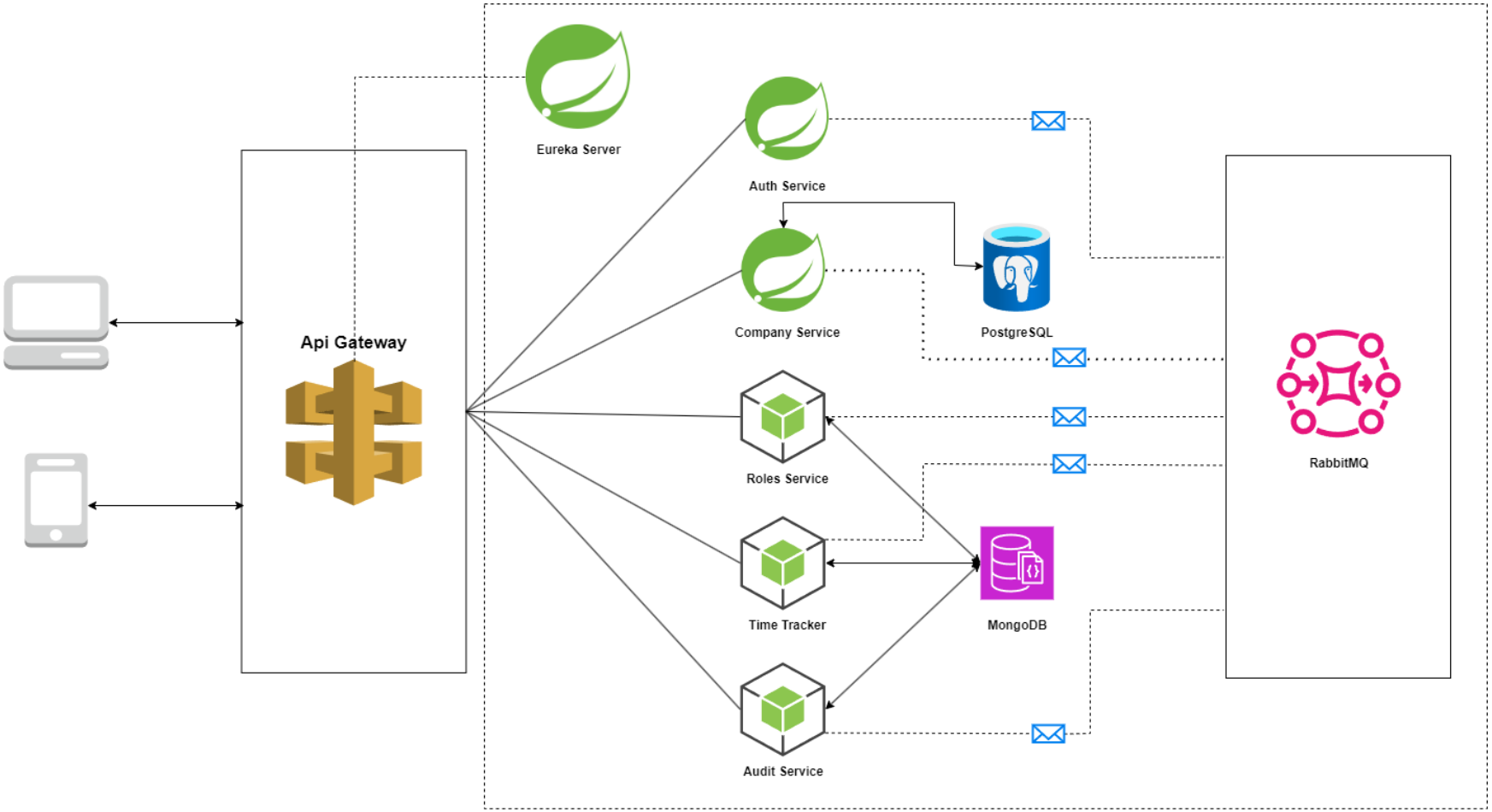


Figura 11. *Arquitectura diseñada para la plataforma "GeoWorker"*

Una vez elaborado el documento de arquitectura de software, se procedió con la codificación y configuración de los servicios backend utilizando Spring Cloud como framework para la implementación de la arquitectura, además, se utilizó Java y Typescript como lenguajes de programación para el desarrollo de los microservicios, con los frameworks Spring-Boot y NestJS respectivamente. Por otro lado, la gestión de los datos de Empresas, Usuarios, Proyectos y Tareas se la realizó por medio de PostgreSQL, dejando la gestión de Roles, Horarios, Geocercas y Logs del sistema a cargo de MongoDB.
En el siguiente apartado, se presenta las configuraciones esenciales y estructura de los microservicios más relevantes dentro de la arquitectura:

Naming Service

En este repositorio denominado "*naming-service*", basado en Spring-Boot, se instaló las dependencias del *Eureka Server* de Spring Cloud, estas dependencias permiten al servicio ofrecer las siguientes funcionalidades:

- Registro de servicios
- Descubrimiento de servicios
- Monitoreo y salud
- Escalabilidad

```java
@SpringBootApplication
@EnableEurekaServer
public class NamingServiceApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(NamingServiceApplication.class, args);
    }

}
```

Figura 12. *Clase principal del naming-service*

Una vez instalada las dependencias, se configuró el archivo *application.properties*, el cual contiene las variables y configuraciones principales del servicio (Véase **Figura 13**).

```
spring.application.name=naming-server
server.port=6005
eureka.client.registerWithEureka=false
eureka.client.fetchRegistry=false
eureka.client.service-url.defaultZone=http://localhost:6005/eureka/
eureka.server.maxThreadsForPeerReplication=0
eureka.server.enableSelfPreservation=false
#***************RABBIT MQ CONFIG***************#
spring.rabbitmq.host=${RABBITMQ_HOST:localhost}
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

***Figura 13**. Archivo de configuraciones del naming-service*

La estructura de las carpetas dentro del proyecto del naming-service se muestra en la **Figura 14.**

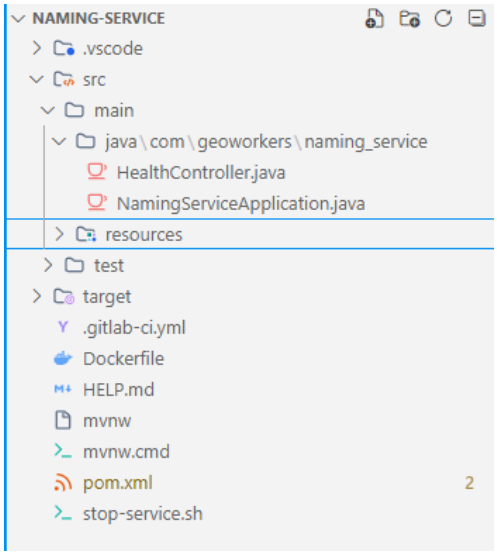


***Figura 14**. Estructura de carpetas naming-service*

Api Gateway

El puerto de este servicio es el único que se expone y, por ende, es por donde se reciben todas las solicitudes desde el cliente. Cada solicitud que ingresa se enruta al servicio correspondiente, sin embargo, los servicios deben estar previamente registrado en el servicio de descubrimiento *naming-service.* El Api Gateway está construido en Spring-Boot y es otro componente esencial dentro del entorno de Spring Cloud, ofreciendo las siguientes funciones:

- Enrutamiento

- Filtros de Gateway
- Seguridad

Para configurar el enrutamiento de los microservicios a través del api-gateway, se creó una clase única para cada servicio. En la **Figura 15**, se muestra la clase que realiza el enrutamiento del servicio *company-service*, además, en esta clase se añadieron los *Histryx* propios del api gateway y un filtro de autorización, el cual actúa como el primer filtro de seguridad en la arquitectura y permite verificar que cada solicitud que entra al api gateway tenga un JWT en sus cabeceras.

```
@EnableHystrix
public class CompanyServiceRouteConf {

	@Autowired
	AuthorizationFilter authorizationFilter;

	@Bean
	public RouteLocator companyServiceRoutes(RouteLocatorBuilder builder) {
		Config config = new Config();
		GatewayFilter gatewayFilter = authorizationFilter.apply(config);

		return builder.routes()
				.route(r -> r.path(...patterns:"/company-service/create").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/company-service/update/**").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/company-service/list").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))

				.route(r -> r.path(...patterns:"/user-service/user/create-member")
					.filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/user-service/user/create-admin-company")
					.filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/user-service/user/profile").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/user-service/user/get-all").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))      You, a month ago • add routes
				.route(r -> r.path(...patterns:"/user-service/user/update/**").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/user-service/user/get-by-company/**")
					.filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))

				.route(r -> r.path(...patterns:"/task-service/create").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/task-service/list").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/task-service/get-file").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/task-service/add-file/**").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/task-service/remove-file/**").filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
				.route(r -> r.path(...patterns:"/task-service/update-status/**")
					.filters(f -> f.filter(gatewayFilter))
					.uri(uri:"lb://COMPANY-SERVICE"))
```

***Figura 15**. Configuración de rutas de los microservicios*

La **Figura 16**, la estructura del proyecto api-gateway, como se mencionó anteriormente, cada microservicio tiene su carpeta y clase separadas para el enrutamiento.

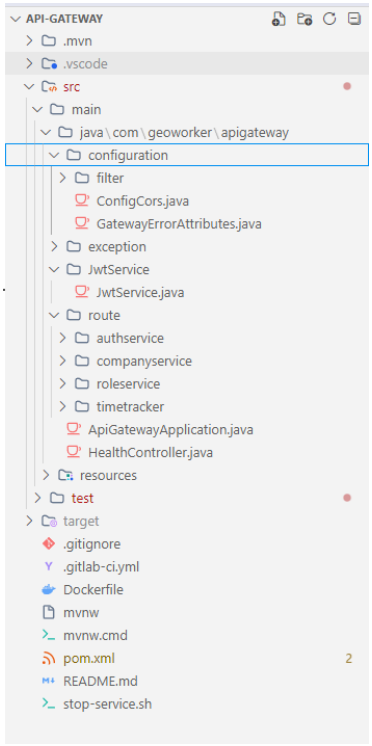


Figura 16. *Estructura de carpetas api-gateway*

Company Service

Desarrollado en Spring-Boot, este servicio es uno de los más importantes y que tiene más lógica de negocio, las entidades que maneja son: Empresa, Usuario, Proyecto, y Tarea. También, es el único servicio conectado a una base de datos relacional.

El archivo que se muestra en la **Figura 17**, contiene las configuraciones de los siguientes aspectos:

- Conexión a la Base de Datos
- Conexión al Eureka Server (ubicado en el *naming-service*)
- Conexión al Bucket de AWS S3
- Exchanges, Colas y RPCs necesarios para la comunicación a otros servicios a través de RabbitMQ.

```
spring.application.name=${APP_NAME:company-service}
server.port=${PORT:6020}
#**********NAMING SERVICE**********#
eureka.client.service-url.defaultZone=http://${EUREKA_HOST:localhost}:${EUREKA_PORT:6005}/eureka
#**********POSTGRESQL**********#
spring.datasource.url= jdbc:postgresql://${DB_HOST:localhost}:${DB_PORT:5432}/company_service
spring.datasource.username=${DB_USERNAME:geoworker}
spring.datasource.password=${DB_PASSWORD:geoworker}
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation= true
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto= update

spring.servlet.multipart.max-file-size=50MB
spring.servlet.multipart.max-request-size=50MB
#..............RABBIT CONFIG..............#
spring.rabbitmq.host=${RABBITMQ_HOST:localhost}
spring.rabbitmq.port=${RABBITMQ_PORT:5672}
spring.rabbitmq.username=${RABBITMQ_USERNAME:guest}
spring.rabbitmq.password=${RABBITMQ_PASSWORD:guest}

aws.accessKeyId=AKIAQR37ZEDNPNEJQAN4
aws.secretKey=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
aws.s3.region=us-east-1
aws.s3.bucket.name=geoworker

aws.task.files.folder=task-files
aws.leave.request.files.folder=leave-request-files

availables.status.options=active, inactive, all
availables.languages=en, es
default.language=es

#..............EXCHANGES..............#
authentication.direct.exchange=authentication
role.direct.exchange=role
audit.logs.direct.exchange=auditLogs
company.direct.exchange=company
#..............RPC..............#
retrive.user.information.rpc=retrieveUserInformation
retrieve.role.information.rpc=retrieveRoleInformation
retrieve.role.admin.company.rpc=retrieveRoleAdminCompany
retrieve.company.info.rpc=retrieveCompanyInfo
#..............QUEUES..............#
save.audit.logs.queue=saveAuditLogs
create.role.company.queue=createRoleCompany
deactivate.role.company.queue=deactivateRoleCompany
```

Figura 17. *Configuración principal de company-service*

La estructura del proyecto, es decir, sus carpetas, los modelos, servicios, repositorios, dtos, y configuraciones generales se puede visualizar en la **Figura 18**.

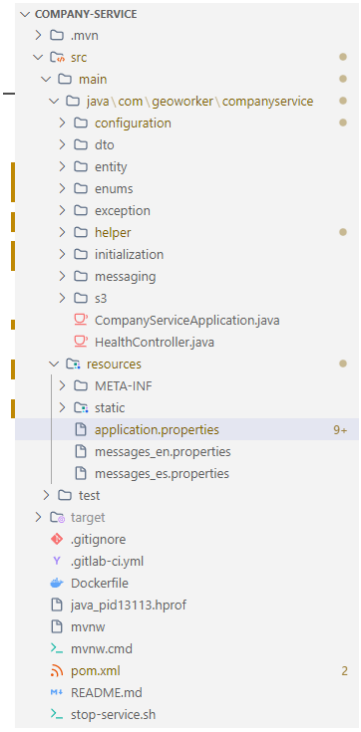


Figura 18. *Estructura de carpetas company-service*

Como último paso en esta actividad, se procedió a subir los microservicios desarrollados a diferentes repositorios dentro de la plataforma GitLab, que actuó como gestor de versiones del código fuente (Ver **Figura 19**).

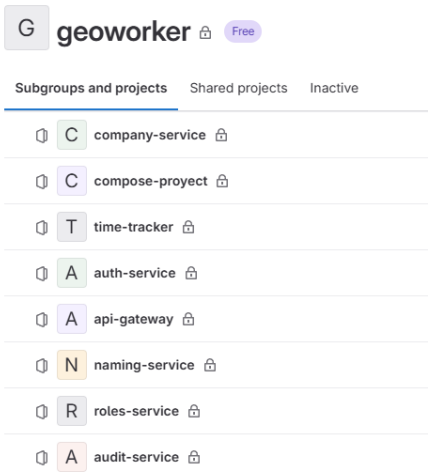


Figura 19. *Repositorios de la plataforma "GeoWorker"*

6.1.4. Configurar los archivos necesarios para la Dockerización de los servicios backend.

Para la Dockerización, se creó el archivo Dockerfile para cada uno se los microservicios. Dado que la plataforma se desarrolló en Java y TypeScript, se generó dos tipos de archivos que se muestran a continuación:

```
You, a day ago | 1 author (You)
1  FROM openjdk:17-jdk-alpine      You, a day ago • add pipeline
2  RUN apk --no-cache add curl
3  ADD target/*.jar app.jar
4  ENV JAVA_OPTS=""
5  ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
6
```

***Figura 20**. Dockerfile para servicios JAVA*

```
You, 9 hours ago | 1 author (You)
1  FROM node:14       You, 9 hours ago • change
2  RUN apt-get install -y curl
3  COPY ./node_modules node_modules/
4  COPY ./dist dist/
5  CMD ["node", "dist/main.js"]
6
```

***Figura 21.** Dockerfile para servicios TypeScript*

6.1.5. Implementar los pipelines que permitan integración y el despliegue continuo

Para esta actividad, se configuro GitLab CI como plataforma para la integración y despliegue continuo. El proceso empezó configurando los archivos *.gitlab-ci.yml* en cada uno de los servicios backend de la plataforma. Al igual que los archivos de Dockerización, se generó dos tipos de archivos para cada lenguaje de programación implementando.

El primer archivo de los servicios Java se estructuro con las siguientes fases (stages): *compile, test, package, dockerize, publish image, deploy*.

```
1  stages:
2      - compile
3      - test
4      - package
5      - dockerize
6      - publish-image
7      - deploy
```

***Figura 22.** Fases del pipeline para servicios en Java*

En el primer stage se realiza el proceso de compilación del servicio, el comando utilizado de lo puede visualizar en la siguiente imagen.

```
compile:
    stage: compile
    script:
        - mvn clean compile
    tags:
        - dev
    only:
        - development
```

Figura 23. *Fase de compilación para servicios en Java*

La segunda fase consiste en aplicar y automatizar las pruebas que se hayan codificado para el servicio, tal como se muestra en la imagen, esta fase depende de la fase de compilación.

```
tests:
    stage: test
    dependencies:
        - compile
    script:
        - mvn test
    tags:
        - dev
    only:
        - development
    when: manual
```

Figura 24. *Fase de pruebas en el servicio en Java*

La fase de empaquetado es otra fase que se ha configurado en el pipeline, esto permite empaquetar el servicio y crear el .jar del código desarrollado.

```
package:
    stage: package
    dependencies:
        - tests
    script:
        - mvn package -Dmaven.test.skip=true
    artifacts:
        expire_in: 1h
        paths:
            - target/*.jar
    tags:
        - dev
    only:
        - development
```

Figura 25. *Fase de Empaquetado del servicio en Java*

La cuarta fase consiste en Dockerizar el servicio, es decir, en esta fase se construye la imagen basándose en el archivo Dockerfile previamente establecido, con los recursos necesarios para que el contenedor de ejecute.

```
dockerize:
    stage: dockerize
    dependencies:
        - package
    script:
        - sudo docker build -t $CI_REGISTRY_IMAGE:dev .
    tags:
        - dev
    only:
        - development
```

Figura 26. *Fase de Dockerización del servicio en Java*

En la fase de publicación, se sube la imagen obtenida previamente, al servicio Container Registry de GitLab, de esta forma se puede almacenar las imágenes asignándoles un tag como identificador, y así, almacenarlas y obtenerlas de manera segura .

```
publish-image:
    stage: publish-image
    dependencies:
        - package
    script:
        - sudo docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
        - sudo docker push $CI_REGISTRY_IMAGE:dev
    tags:
        - dev
    only:
        - development
```

Figura 27. *Fase de publicación del servicio en Java*

El despliegue es la última fase configurada en el pipeline, aquí detiene y se elimina el servicio que se va a actualizar, de esta forma se garantiza que el servicio se levante con las últimas actualizaciones. Todo este proceso consiste en detener el servicio, descargar la imagen para el contenedor y posteriormente ejecutar el proyecto de Docker Compose, así, también se controla que, si hay algún servicio que este caído, se levante de nuevo.

```
deploy:
    stage: deploy
    dependencies:
        - dockerize
    before_script:
        - chmod +x stop-service.sh
        - sudo ./stop-service.sh company-service
    script:
        - sudo docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
        - sudo docker pull $CI_REGISTRY_IMAGE:dev
        - sudo docker-compose -f /home/compose-proyect/docker-compose.yml up -d
    tags:
        - dev
    only:
        - development
```

Figura 28. *Fase de despliegue del servicio en Java*

Para los servicios desarrollados en NestJS (TypeScript), se implementó las mismas fases en el pipeline, sin embargo, las tres primeras fases cambian ya que los comandos no son los mismos que para un servicio en Java.

La primera fase denominada *install*, es la fase donde se instala las dependencias declaradas en el archivo *package.json*.

```
12  install_dependencies:
13      stage: install
14      script:
15          - npm install
16      artifacts:
17          paths:
18          - node_modules/
19      tags:
20          - dev
21      only:
22          - development
```

Figura 29. *Fase de instalación de dependencias en un servicio de NodeJS*

La segunda fase implica ejecutar las pruebas escritas para el servicio.

```
23  testing_testing:
24      stage: test
25      dependencies:
26          - install_dependencies
27      script: npm run test-server
28      tags:
29          - dev
30      only:
31          - development
32      when: manual
```

Figura 30. *Fase de ejecución de pruebas en los servicios de NodeJS*

La fase de compilación para los servicios en NodeJS consiste en crear la carpeta dist con el código compilado y comprimido.

```
33  build_project:
34      stage: build
35      dependencies:
36          - testing_testing
37      script:
38          - npm run build
39      artifacts:
40          paths:
41              - dist/
42      tags:
43          - dev
44      only:
45          - development
```

Figura 31. *Fase de compilación para un servicio en NodeJS*

Las fases restantes para el pipeline de un servicio de NodeJS, consiste en prácticamente lo mismo que el pipeline descrito anteriormente para los servicios de Java, pues para las fases de Dockerización, Publicación y Despliegue se utiliza los mismos comandos de Docker.

6.1.6. Emplear Docker Compose para facilitar el despliegue de los servicios backend y de monitoreo.

El despliegue de los servicios backend se lo realizó utilizando Docker Compose, el código generado en esta actividad se subió a la plataforma GitLab para tener un mejor control sobre las versiones de este proyecto. Dentro del archivo *docker-compose.yml* generado existen cuatro partes importantes que se detallan a continuación:

- **Configuración de las bases de Datos**

  Esta configuración permitió detallas los servicios de PostgreSQL y MongoDB, el código se lo muestra en la **Figura 32**.

```yaml
postgres-service:
  image: postgres:13.2
  container_name: postgres-service
  ports:
    - "5432:5432"
  restart: on-failure
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U geo-worker-user"]
    interval: 30s
    timeout: 30s
    retries: 3
  env_file:
    - ./env/postgres_credentials.env
  volumes:
    - postgres-service-vol:/var/lib/postgresql/data
    - ./sql/postgres_data.sql:/docker-entrypoint-initdb.d/postgres_data.sql
    - ./healthchecks/postgres.sh:/usr/local/bin/postgres.sh
  networks:
    - geo_backend

mongo-service:
  image: mongo:4.4.3
  container_name: mongo-service
  ports:
    - 27018:27017
  restart: on-failure
  healthcheck:
    test:
      - CMD
      - mongo
      - --eval
      - "db.adminCommand('ping')"
    interval: 10s
    timeout: 10s
    retries: 5
  networks:
    - geo_backend
  volumes:
    - mongo-vol:/data/db
    - ./healthchecks/mongo.sh:/usr/local/bin/mongo.sh
```

Figura 32. *Código en Docker Compose para PostgreSQL y MongoDB*

- **Configuración de los servicios backend**

Para que los servicios backend se puedan instanciar en el proyecto de Docker Compose, previamente fue necesario habilitar un endpoint que ofrezca una respuesta del estado del servicio, es decir, este endpoint funcionará como un healthcheck y permitirá saber si un servicio está funcionando correctamente o tiene algún problema que no lo deja desplegarse o funcionar correctamente. El código para los servicios backend se puede visualizar en la **Figura 33**.

```yaml
naming-service:
  image: registry.gitlab.com/geoworker/naming-service:dev
  container_name: naming-service
  ports:
    - 6005:6005
  restart: on-failure
  env_file:
    - ./env/naming_service.env
  healthcheck:
    test: ["CMD-SHELL", "curl -f http://localhost:6005/naming-service/health"]
    interval: 30s
    timeout: 5s
    retries: 3
  networks:
    - geo_backend
  depends_on:
    rabbit-service:
      condition: service_healthy

api-gateway:
  image: registry.gitlab.com/geoworker/api-gateway:dev
  container_name: api-gateway
  ports:
    - 6010:6010
  env_file:
    - ./env/api_gateway.env
  healthcheck:
    test: ["CMD-SHELL", "curl -f http://localhost:6010/api-gateway/health"]
    interval: 30s
    timeout: 5s
    retries: 3
  depends_on:
    naming-service:
      condition: service_healthy
    rabbit-service:
      condition: service_healthy
  networks:
    - geo_backend
```

***Figura 33**. Configuración para servicios backend*

- **Configuración de RabbitMQ (Broker Message)**

En la **Figura 34**, se presenta el código para configurar el despliegue del servicio de RabbitMQ, esta configuración contempla, imagen, puertos, verificación de estado (healthcheck), la red privada dentro de Docker Compose y el volumen para la persistencia de datos.

```yaml
rabbit-service:
  image: rabbitmq:3.8.11-management
  container_name: rabbit-service
  ports:
    - 15672:15672
    - 5672:5672
  restart: on-failure
  healthcheck:
    test: ["CMD-SHELL", "/usr/local/bin/rabbit.sh"]
    interval: 30s
    start_period: 60s
  networks:
    - geo_backend
  volumes:
    - ./healthchecks/rabbitmq.sh:/usr/local/bin/rabbit.sh
```

Figura 34. *Configuración para RabbitMQ*

- **Configuración de servicios de Monitoreo**

Para el Monitoreo de los servicios backend, se utilizó Prometheus, Grafana y se complementó con NodeExporter y Cadvisor, cada servicio tiene su propio puerto y todos estos complementos sirven para mostrar datos del servidor a través de la interfaz de Grafana.

```yaml
node-exporter:
  image: prom/node-exporter
  container_name: node-exporter
  ports:
    - '9100:9100'
  networks:
    - geo_backend

cadvisor:
  image: gcr.io/cadvisor/cadvisor:latest
  container_name: cadvisor
  privileged: true      You, a day ago • add monitoring
  volumes:
    - /:/rootfs:ro
    - /var/run:/var/run:rw
    - /sys:/sys:ro
    - /var/lib/docker/:/var/lib/docker:ro
  ports:
    - '8080:8080'
  networks:
    - geo_backend

prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  ports:
    - "9090:9090"
  volumes:
    - ./prometheus:/etc/prometheus
  command:
    - "--config.file=/etc/prometheus/prometheus.yml"
  networks:
    - geo_backend

grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3000:3000"
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=geoWG24
    - GF_INSTALL_PLUGINS=grafana-clock-panel
  volumes:
    - grafana_data:/var/lib/grafana
  restart: unless-stopped
  networks:
    - geo_backend
```

Figura 35*. Servicios de Monitoreo*

6.1.7. Desplegar los servicios backend en un dominio y/o IP pública para garantizar la accesibilidad.

En esta fase se desplegó los servicios backend en Amazon Web Services (AWS), lo que permitió asegurar que los servicios desarrollados sean accesibles públicamente y se pueda crear un escenario para pruebas de carga y estrés. A continuación, se detalla el proceso implementado para cumplir con esta actividad:

1) Se creó una instancia de EC2 en AWS, esto permitió contratar un servicio de computación en la nube con un Sistema Operativo (SO) que pueda ejecutar contenedores Docker. El SO seleccionado fue Ubuntu, ya que este sistema operativo ejecuta contenedores Docker de manera nativa. La **Figura 36**, muestra el tipo de instancia contratada, la imagen del sistema operativo utilizada para la instancia y otros detalles como la IP pública, etcétera.

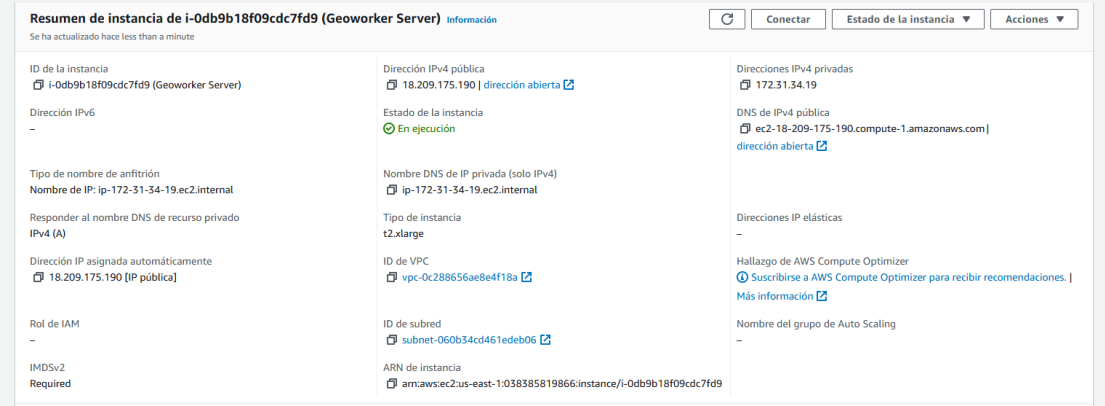


Figura 36. *Detalles de la instancia EC2 en AWS*

2) Una vez creada la instancia de EC2, se procedió a clonar el repositorio de Docker Compose dentro del servidor virtual contratado, luego, se ejecutó el comando *sudo docker-compose up -d*, que dio inicio a la descarga de las imágenes de los contenedores de la plataforma como se muestra en la **Figura 37**.

```
ubuntu@ip-172-31-34-19:/home/compose-proyect$ sudo docker-compose up -d
Starting postgres-service ...
Starting postgres-service ... done
Starting node-exporter    ... done
Starting rabbit-service   ... done
Starting mongo-service    ... done
Starting prometheus       ... done
Starting cadvisor         ... done
naming-service is up-to-date
Starting audit-service    ... done
Starting api-gateway      ... done
Starting company-service  ... done
Starting time-tracker     ... done
Starting roles-service    ... done
Starting auth-service     ... done
```

Figura 37. *Inicialización del proyecto de Docker Compose*

3) Finalmente, ya que el comando mencionado anteriormente se haya terminado de ejecutar, los servicios se pueden listar con el comando *sudo docker ps*, esto mostrará la información general de todos los contenedores que se implementaron en este TT, tal como se muestra en la **Figura 38**.

```
ubuntu@ip-172-31-34-19:/home/compose-proyect$ sudo docker ps
CONTAINER ID   IMAGE                                             COMMAND                  CREATED       STATUS                    PORTS
                                                                       NAMES
1143bc1279d8   registry.gitlab.com/geoworker/naming-service:dev   "sh -c 'java $JAVA_O…"   2 weeks ago   Up 20 minutes (healthy)   0.0.0.0:6005->6005/tcp, :::6005->6005/tcp
                                                                       naming-service
b23fdb46a123   registry.gitlab.com/geoworker/audit-service:dev    "docker-entrypoint.s…"   2 weeks ago   Up 5 minutes (healthy)    0.0.0.0:6035->6035/tcp, :::6035->6035/tcp
                                                                       audit-service
aba23a4db357   registry.gitlab.com/geoworker/api-gateway:dev      "sh -c 'java $JAVA_O…"   2 weeks ago   Up 5 minutes (healthy)    0.0.0.0:6010->6010/tcp, :::6010->6010/tcp
                                                                       api-gateway
ec5184e8c80d   registry.gitlab.com/geoworker/roles-service:dev    "docker-entrypoint.s…"   2 weeks ago   Up 5 minutes (healthy)    0.0.0.0:6030->6030/tcp, :::6030->6030/tcp
                                                                       roles-service
995c9dfec029   registry.gitlab.com/geoworker/time-tracker:dev     "docker-entrypoint.s…"   3 weeks ago   Up 5 minutes (healthy)    0.0.0.0:6025->6025/tcp, :::6025->6025/tcp
                                                                       time-tracker
61e4da165821   registry.gitlab.com/geoworker/company-service:dev  "sh -c 'java $JAVA_O…"   3 weeks ago   Up 5 minutes (healthy)    0.0.0.0:6020->6020/tcp, :::6020->6020/tcp
                                                                       company-service
d7dcc0442a3d   registry.gitlab.com/geoworker/auth-service:dev     "sh -c 'java $JAVA_O…"   3 weeks ago   Up 5 minutes (healthy)    0.0.0.0:6015->6015/tcp, :::6015->6015/tcp
                                                                       auth-service
823bb6a58427   grafana/grafana:latest                             "/run.sh"                3 weeks ago   Up 20 minutes             0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
                                                                       grafana
1ecfd04951cd   rabbitmq:3.8.11-management                         "docker-entrypoint.s…"   3 weeks ago   Up 5 minutes (healthy)    4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, ::
91-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672/tcp   rabbit-service
4151671fd854   prom/prometheus:latest                             "/bin/prometheus --c…"   3 weeks ago   Up 5 minutes              0.0.0.0:9090->9090/tcp, :::9090->9090/tcp
                                                                       prometheus
e76318a838d5   gcr.io/cadvisor/cadvisor:latest                    "/usr/bin/cadvisor -…"   3 weeks ago   Up 5 minutes (healthy)    0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
                                                                       cadvisor
59ce6943b308   prom/node-exporter                                 "/bin/node_exporter"     3 weeks ago   Up 5 minutes              0.0.0.0:9100->9100/tcp, :::9100->9100/tcp
                                                                       node-exporter
7124a3c648ea   mongo:4.4.3                                        "docker-entrypoint.s…"   3 weeks ago   Up 5 minutes (healthy)    0.0.0.0:27018->27017/tcp, :::27018->27017/tcp
                                                                       mongo-service
d3e86cc2cfcf   postgres:13.2                                      "docker-entrypoint.s…"   3 weeks ago   Up 5 minutes (healthy)    0.0.0.0:5432->5432/tcp, :::5432->5432/tcp
                                                                       postgres-service
```

Figura 38. *Lista de contenedores en ejecución en AWS*

6.1.8. Documentar las funciones, herramientas y procesos utilizados en el desarrollo de los servicios backend.

En el desarrollo del TT, se implementaron diferentes procesos, técnicas y estándares, los cuales se los detalló y documento en la sección final de anexos. Además, se generó la documentación de todas las funcionalidades desarrolladas en cada microservicio, esto con el fin de proveer una guía para futuras implementaciones o modificaciones que se requieran en la plataforma "GeoWorker".

6.2. Evaluar los servicios backend en un escenario experimental

La evaluación de los servicios backend se realizó considerando dos aspectos importantes, el primero, escribiendo las pruebas unitarias que permitieron testear el código desarrollado, de esta forma se verificó que los endpoints están procesando y retornando los datos correctamente. El segundo aspecto fue testear la capacidad de respuesta de los servicios, frente a una alta demanda de peticiones con la ayuda de JMeter.

6.2.1. Implementar pruebas unitarias en las funciones principales de los servicios backend de la plataforma GeoWorker.

Las pruebas unitarias consintieron en verificar las funcionalidades principales de los controladores y servicios de la plataforma GeoWorker. Cada prueba unitaria valida que los escenarios propuestos en cada uno de los métodos se cumplan o retornen los datos y excepciones correspondientes. Como los servicios backend están desarrollados bajo dos frameworks y lenguajes de programación diferentes, a continuación, en la **Figura 39**, se muestra un ejemplo de las pruebas unitarias escritas para los servicios construidos en Spring Boot (Java) y en la **Figura 40**, se muestra como es el resultado de la ejecución de dichas pruebas.

```java
@Test
public void testCreate() throws Exception {
    SaveProjectDto data = new SaveProjectDto();
    data.setName(name:"Project 1");
    data.setUuidCompany(uuidCompany:"uuidCompany");
    data.setDescription(description:"Description");
    data.setStartDate(new Date());
    data.setEndDate(new Date(new Date().getTime() + 1000000));

    Country country = new Country();
    country.setIdCountry(idCountry:1L);
    country.setName(name:"Country 1");

    Company company = new Company();
    company.setIdCompany(idCompany:1L);
    company.setName(name:"Company 1");
    company.setAddress(address:"Address 1");
    company.setCity(city:"City 1");
    company.setCountry(country);
    company.setStatus(status:true);

    when(companyService.findByUuid(any())).thenReturn(Optional.of(company));
    when(projectService.findByName(any())).thenReturn(Optional.empty());
    try (MockedStatic<GeneralHelper> generalHelper = Mockito.mockStatic(classToMock:GeneralHelper.class)) {
        generalHelper.when(() -> GeneralHelper.isDateGreaterThan(any(), any()))
                .thenReturn(value:false);
    }
    when(projectService.saveProject(any(), any(), any())).thenReturn(ProjectMockData.getProjectInfo().get());

    mockMvc.perform(post(PROJECT_SERVICE_CREATE_PATH).requestAttr(name:"token", TokenMock.mockToken())
            .contentType(MediaType.APPLICATION_JSON).content(mapper.writeValueAsString(data)))
            .andExpect(status().isCreated()).andDo(result -> {
                System.out.println(result.getResponse());
            });
}
```

Figura 39. *Prueba Unitaria en la función de crear proyecto*

```
2024-06-27T20:43:57.978-05:00  INFO 51393 --- [           main] o.s.b.t.m.w.SpringBootMockServletContext : Initializing Spring Test
2024-06-27T20:43:57.978-05:00  INFO 51393 --- [           main] o.s.t.web.servlet.TestDispatcherServlet  : Initializing Servlet ''
2024-06-27T20:43:57.979-05:00  INFO 51393 --- [           main] o.s.t.web.servlet.TestDispatcherServlet  : Completed initialization
2024-06-27T20:43:57.988-05:00  INFO 51393 --- [           main] o.s.b.t.m.w.SpringBootMockServletContext : Initializing Spring Test
2024-06-27T20:43:57.988-05:00  INFO 51393 --- [           main] o.s.t.web.servlet.TestDispatcherServlet  : Initializing Servlet ''
2024-06-27T20:43:57.989-05:00  INFO 51393 --- [           main] o.s.t.web.servlet.TestDispatcherServlet  : Completed initialization
2024-06-27T20:43:58.005-05:00  INFO 51393 --- [           main] o.s.b.t.m.w.SpringBootMockServletContext : Initializing Spring Test
2024-06-27T20:43:58.005-05:00  INFO 51393 --- [           main] o.s.t.web.servlet.TestDispatcherServlet  : Initializing Servlet ''
2024-06-27T20:43:58.006-05:00  INFO 51393 --- [           main] o.s.t.web.servlet.TestDispatcherServlet  : Completed initialization
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.475 s - in com.geoworker.companyservice.user.controller.Us
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 16, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  6.572 s
[INFO] Finished at: 2024-06-27T20:43:58-05:00
[INFO] ------------------------------------------------------------------------
alejandro@alejandro:~/geoworker/company-service$
```

Figura 40. *Resultados de las pruebas unitarias ejecutadas en un servicio Java*

En el mismo contexto, en la **Figura 41**, se puede visualizar un ejemplo del código utilizado para construir las pruebas unitarias en un servicio en NestJS, y en la **Figura 42**, se muestra el resultado de la ejecución de estas pruebas.

```
});

it('should save the new clock record if there is no last register', async () => {
  Click to collapse the range.
  const mockData: SaveRegistryDto = {
    uuidUser: 'uuid',
    type: TypeRegistry.IN,
    nameUser: 'name',
    date: new Date(),
  };
  jest.spyOn(service, 'findLastTypeRegister').mockResolvedValue(null);
  const saveMock = jest.fn().mockResolvedValue(mockData);
  jest.spyOn(service, 'save').mockImplementation(saveMock);
  await service.save(mockData);
  expect(saveMock).toHaveBeenCalled();
});      You, a week ago • add tests

it('should save the clock record if the last register has a different type', async () => {
  const mockData: SaveRegistryDto = {
    uuidUser: 'uuid',
    type: TypeRegistry.IN,
    nameUser: 'name',
    date: new Date(),
  };

  const lastRegister: any = {
    _id: '1234567890abcdef12345678',
    type: TypeRegistry.IN,
    uuidUser: 'uuid',
    nameUser: 'name',
    uuidCompany: 'uuid',
    date: new Date(),
  };

  jest.spyOn(service, 'findLastTypeRegister').mockResolvedValue(lastRegister);
  const saveMock = jest.fn().mockResolvedValue(mockData);
  jest.spyOn(service, 'save').mockImplementation(saveMock);
  await service.save(mockData);
  expect(saveMock).toHaveBeenCalled();
});
```

Figura 41. *Pruebas unitarias para un servicio de NestJS*

```
alejandro@alejandro:~/geoworker/time-tracker$ npm run test

> time-tracker@0.0.1 test /home/alejandro/geoworker/time-tracker
> jest
    file:///home/alejandro/geoworker/time-tracker/src/entity/configuration/service/configuration.service.spec.ts (ctrl + click)

 PASS  src/entity/configuration/service/configuration.service.spec.ts
 PASS  src/health/health.controller.spec.ts
 PASS  src/entity/clock/service/clock.service.spec.ts
 PASS  src/entity/geofence/service/geofence.service.spec.ts
 PASS  src/entity/clock/controller/clock.controller.spec.ts
 PASS  src/entity/geofence/controller/geofence.controller.spec.ts

Test Suites: 6 passed, 6 total
Tests:       26 passed, 26 total
Snapshots:   0 total
Time:        5.387 s
Ran all test suites.
```

Figura 42. *Resultado de la ejecución de las pruebas unitarias en NestJS*

6.2.2. Ejecución de pruebas de carga y estrés a los servicios backend de la plataforma GeoWorker.

Las pruebas de carga y estrés se las aplicó al servicio de autenticación (*auth-service*) de la plataforma, para esta prueba se configuró una media de 100 usuarios por segundo realizando

la petición de inicio de sesión en JMeter. Para el primer escenario se configuró una sola instancia del servicio. Los resultados se muestran en la **Figura 43**.

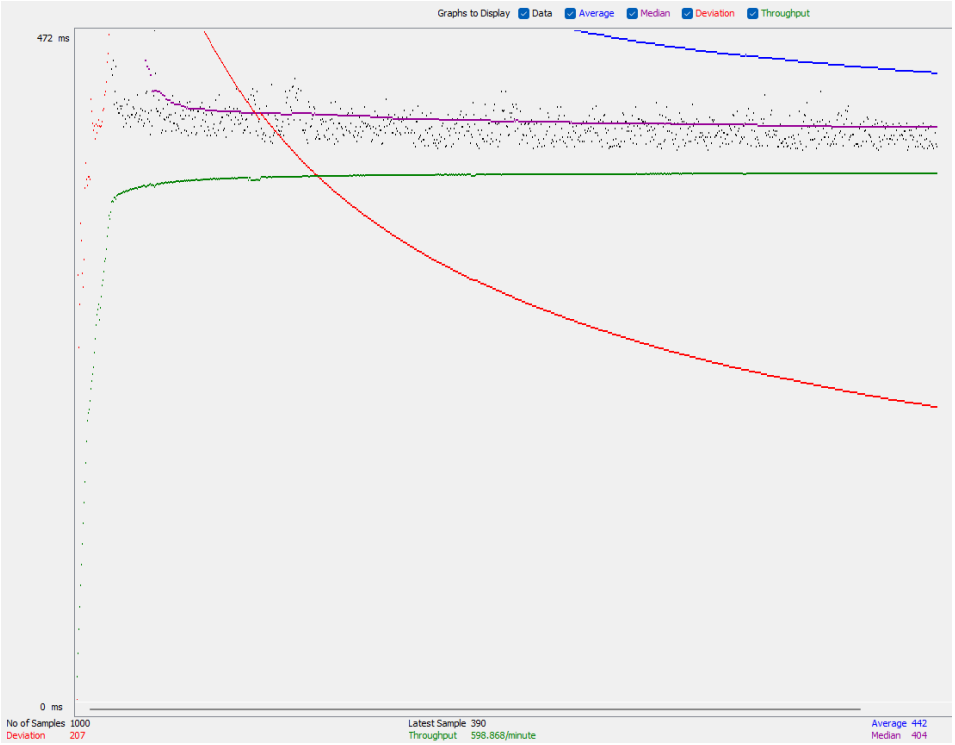


Figura 43. *Inicio de Sesión con 100 usrs/seg con una instancia*

En los resultados obtenidos en este escenario se debe considerar los siguientes puntos:

- La línea verde que corresponde a ***Throughput***, representa el número de solicitudes que el sistema puede manejar por minuto, en esta prueba, esta variable alcanzó las 598.868 solicitudes por minuto.
- El ***Average***, representado con la línea azul, simboliza el tiempo promedio que el sistema responde las solicitudes, es decir, el sistema tiene un tiempo promedio de 442ms de respuesta.
- La dispersión que se observa en la línea de color negro se puede interpretar como una variación en los tiempos de respuesta que el servidor tuvo al inicio de la prueba, pero que se estabilizó hasta el final de la misma.

Para la siguiente prueba, se planteó las mismas variables en JMeter, es decir, 100 usuarios por segundo realizando la petición de inicio de sesión, sin embargo, se realizó un cambio en el proyecto Docker Compose con el fin de proporcionar dos instancias del servicio de

autenticación. Es importante resaltar que los recursos de la instancia en EC2 no se modificó para esta prueba.

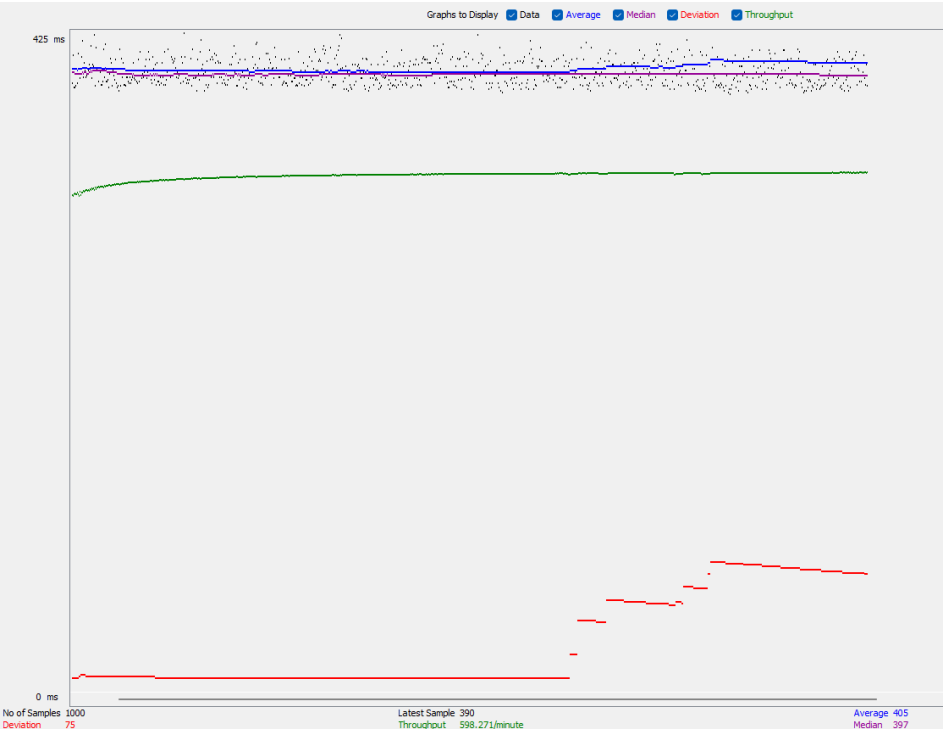


Figura 44. *Inicio de Sesión con 100 usrs/seg. con dos instancias de auth-service*

Con los resultados obtenidos en este escenario se debe considerar los siguientes puntos:

- El ***Throughput***, se mantiene en el rango, pero variando poco, llegando a 598.271 solicitudes por minuto.
- El ***Average***, es más estable y ligeramente menor con un tiempo de 405ms frente a los 442ms de la prueba anterior.
- El valor en la variable ***Median*** ha disminuido, lo que representa que el sistema en esta prueba mejoró en los tiempos de respuesta.
- Se puede visualizar que, en la línea de color negro, los datos se agrupan, lo que significa que los tiempos de respuesta individual han sido más estables en esta ocasión.

7. Discusión

En este apartado, se discute los resultados obtenidos en el Trabajo de Titulación.

Para validar los resultados obtenidos, se realizó una encuesta (Ver **Anexo 5.** Encuesta de satisfacción realizada al Gerente de la Empresa) al Gerente de la empresa, esta encuesta busca conocer el nivel de satisfacción sobre los requerimientos desarrollados y dar contestación a la pregunta de investigación, ¿Puede la integración de la arquitectura de microservicios combinado con técnicas de DevOps cumplir con los requisitos de disponibilidad y eficiencia de la plataforma GeoWorker?

Previo a la realización de la encuesta, se socializó la arquitectura implementada verificando que todos los requerimientos estén cubiertos, además, se aplicó pruebas de carga y estrés con el propósito de dar al encuestado contexto que le permita comparar y contestar las preguntas de la encuesta. En la **Tabla 12**, se presentan los aspectos abordados en la encuesta y el número de preguntas que se relacionan con dichos aspectos. La selección de las preguntas, no solo se basó en la pregunta de investigación, sino que también, se consideraron aspectos que generalmente se usan para evaluar sistemas informáticos.

Tabla 12. *Aspectos abordados en la encuesta*

Aspecto	Número Preguntas
Disponibilidad	2
Tiempo de despliegue y actualización	1
Monitoreo	1
Eficiencia operativa	1
Escalamiento	1
Mantenimiento	1
Colaboración y comunicación	1
Rendimiento	1

Para comenzar con el análisis de la encuesta realizada, se describe cada uno de los aspectos planteados y el resultado obtenido.

Con respecto a la disponibilidad, el encuestado afirmó que la plataforma tiene una mejora considerable en comparación con otros proyectos desarrollados por la empresa. Con este aspecto

se valida que la plataforma pueda mantenerse operativa frente a posibles errores o actualizaciones sin que se interrumpa el servicio al usuario final. Seguidamente, se le consultó al encuestado si ha observado una mejora en los tiempos de actualización y despliegue de los servicios backend, obteniendo una respuesta positiva en este punto, ya que, con la solución implementada, se mejoró y automatizó estos procesos que generalmente las empresas lo realizan de manera manual. Sobre el monitoreo implementado a los servicios backend de la plataforma, se obtuvo que por parte de la empresa se encuentran satisfechos y a esta característica se la calificó como muy eficiente. Al abordar la eficiencia operativa, se buscó definir si el TT, ha afectado el punto de vista operativo, aquí se concluyó, que el sistema ha mejorado significativamente con respecto a otros proyectos de la empresa. Con referencia a la escalabilidad, se le indicó al encuestado que realice una comparativa entre la arquitectura de microservicios presentada y otra arquitectura monolítica desarrollada por la empresa, concluyendo que la arquitectura de microservicios ofrece una forma más viable cuando se requiere escalar los recursos. Con respecto al mantenimiento, la encuesta arrojó que los servicios backend desarrollados en el TT, ofrecen un camino más fácil al momento de dar mantenimiento a la infraestructura propuesta. La colaboración y comunicación fue un aspecto donde se consideró la opinión del equipo de desarrollo de la empresa, aquí se reflejó que el equipo ha visto una mejora en estos parámetros, y en el mismo contexto, se obtuvo que se encuentran satisfechos con el nivel de rendimiento demostrado por la plataforma en las pruebas de carga y estrés aplicadas. Por último, en la pregunta final, el encuestado manifestó que, basándose en los resultados y características demostradas por el TT, este recomienda el uso de técnicas de DevOps y la arquitectura de microservicios en proyectos de desarrollo de software.

Es preciso añadir que en esta evaluación se esperaba aplicar las mismas pruebas de carga y estrés a otros sistemas de la empresa PuntoPymes, con el fin de recolectar datos y ofrecer una mejor comparativa, sin embargo, por motivos de protección de datos y privacidad, esta prueba no fue posible realizarla. No obstante, para comparar el rendimiento de la plataforma, se realizó una búsqueda de trabajos relacionados con arquitecturas tradicionales, con el fin de cotejar los datos obtenidos de las pruebas realizadas al TT y las pruebas implementadas en otros trabajos de titulación.

Para complementar la encuesta aplicada al gerente de la empresa, se investigó otros trabajos relacionados con arquitecturas monolíticas, seguidamente, se realizó una comparación entre las pruebas de carga y estrés del TT, y las pruebas realizadas en los trabajos seleccionados. En la **Tabla**

13, se muestra los resultados de la comparativa realizada con el programa JMeter, en esta tabla se consideró varios aspectos como el tiempo de respuesta, la capacidad de procesamiento de solicitudes y la estabilidad del servidor en escenarios de carga y estrés. Como se puede evidenciar en la última columna de la tabla, en todos los aspectos comparados con las arquitecturas monolíticas, la solución implementada en TT demuestra una mejora sustancial, ofreciendo mejores características en cuanto a rapidez y estabilidad.

Tabla 13. *Comparación con otras arquitecturas*

Sistema Monolítico	**Parámetro**	**Arquitectura de microservicios (TT)**	**Sistema Monolítico**	**Diferencia (%)**
[39]	Tiempo de respuesta (ms)	162ms	630ms	74.60%
	Capacidad de procesamiento	9293.47/min	643.307/min	1344.76%
[40]	Capacidad de procesamiento	22976.202/min	198.497/min	11479.32%
	Estabilidad Servidor	4912	8452	41.88%
[41]	Capacidad de procesamiento	100.412/min	612.12/min	509.4%
	Tiempo de respuesta (ms)	349	8743	2405.2%

8. Conclusiones

Una vez completado el Trabajo de Titulación, se llegaron a las siguientes conclusiones:

- La implementación de una arquitectura de microservicio mejora la eficiencia y tiempos de respuesta cuando la carga de trabajo aumenta. Al dividir la plataforma en servicios más pequeños, los procesos se distribuyen entre instancias disponibles, eliminando cuellos de botella dentro de la plataforma y mejorando el rendimiento de la misma.
- La implementación del patrón Domain-Driven Design (DDD), para el diseño de la arquitectura de la plataforma, permitió alienar los requisitos del proyecto de software con el modelo de dominio. Además, con el diagrama de Contextos Delimitados, obtenido en este proceso, se ofreció una manera organizada, precisa y sistemática de descubrir los microservicios y de agrupar las funcionalidades a desarrollar.
- Spring Cloud como framework para implementar arquitecturas de microservicios, ofrece una amplia gama de herramientas como api gateway, tolerancia a fallos y servicio de descubrimiento. Además, brinda algunas ventajas cuando se requiere integrar a su entorno servicios backend desarrollados con diferentes tecnologías, ya que no requiere que exista una compatibilidad entre estas.
- El uso de RabbitMQ como servicio de mensajería, cumplió con las expectativas sobre la comunicación rápida y eficiente, demostrando ser una solución confiable y robusta para la comunicación asíncrona para la arquitectura de microservicios. Su capacidad de gestionar los mensajes y su balanceador de carga hacen que este sistema se pueda aplicar a servicios donde los requerimientos son demandantes. Por último, la flexibilidad y adaptabilidad que ofrece es muy importante, ya que permite trabajar con varios tipos de lenguajes de programación y frameworks.
- Las técnicas de DevOps implementadas demostraron ser beneficiosas para el proyecto, mejorando los tiempos de desarrollo y despliegue de los servicios backend. Con la integración y despliegue continuo (CI/CD) se automatizó procesos como construcción, pruebas y despliegue, ya que generalmente, estos procesos se los realiza de manera manual. Igualmente, los servicios de monitoreo de la plataforma brindaron una perspectiva general del consumo de recursos como Memoria y CPU de los servicios backend, además de estos servicios, se pudo monitorear otros contenedores importantes como bases de datos y el

servicio de mensajería, es decir, se mantuvo un control completo de todos los componentes desarrollados para la plataforma.

- Las bases de datos relacionales son muy eficientes y confiables en escenarios donde se requiera establecer relaciones fuertes o consultas con un alto nivel de complejidad, sin embargo, pierden eficiencia cuando las lecturas y escrituras crece de manera exponencial, en este escenario donde se espera lecturas y escrituras masivas, las bases de datos no relacionales como MongoDB son la mejor opción.

9. Recomendaciones

Luego de haber culminado el desarrollo del TT, se recomienda:

- Utilizar estándares establecidos como el IEEE 830 para definir los requisitos en los proyectos de software es beneficioso. Los requisitos se estructuran de manera clara y precisa, además, mejora la comunicación y el trabajo colaborativo entre las partes interesadas del proyecto, en ese mismo contexto, la planificación también se ve beneficiada, ya que se puede realizar una mejor valoración de las tareas y funcionalidades.
- Implementar arquitecturas de microservicios en proyectos grandes, o sistemas que vayan a tener una alta demanda de usuarios. Este tipo de arquitectura al estar constituidas de servicios pequeños e independientes pueden ofrecer un soporte mayor en escenarios de alta demanda en comparación con las arquitecturas monolíticas.
- El uso de patrones de diseño para arquitecturas de microservicios como Domain-Driven Design (DDD). Este patrón permite agrupar, organizar y describir los microservicios que se van a requerir en un proyecto sin perder la alineación con los requerimientos y el modelo de negocio, además, de mejorar la comunicación entre los desarrolladores y expertos en el dominio.
- El uso de base de datos en memoria como Redis para mejorar la eficiencia y rapidez en los tiempos de respuesta de servicios backend. Este tipo de bases datos proporcionan un acceso rápido y una menor latencia frente a otras bases de datos convirtiéndolas en la mejor opción para almacenar datos que se necesita obtener de manera ágil y continua.
- Para proyectos futuros es aconsejable el uso de orquestadores de contendores Docker como Docker Swarm, Kubernets, ECS de Amazon Web Services, entre otros. Este tipo de herramientas facilitan aún más el despliegue y la gestión de una arquitectura de microservicios permitiendo mejorar los tiempos e integraciones que se requieran implementar en el proyecto.

10. Bibliografía

[1] K. Leung and J. Y. Y. Kwong, “Human resource management practices in international joint ventures in mainland China: a justice analysis,” *Human Resource Management Review*, vol. 13, pp. 85–105, 2003, [Online]. Available: www.HRmanagementreview.com

[2] K.-L. Wo ng, P. T. Sin-H we an, Y.-K. Ng, and C. he -Yang Fong, “The role of HRM in enhancing organizational performance,” *Human Resource Management Research*, vol. 3, pp. 11–15, 2013, doi: 10.5923/j.hrmr.20130301.03.

[3] J. E. Artos Ati, “SISTEMA DE GESTIÓN DE RECURSOS HUMANOS (SGRH), PARA EL CONTROL DE LOS PROCESOS DE TALENTO HUMANO PARA LA EMPRESA MARSED S.A, DE SANTO DOMINGO,” Universidad Regional Autónoma de los Andes, Santo Domingo, 2015.

[4] R. S. Schuler, “Strategic human resources management: Linking the people with the strategic needs of the business,” *Organ Dyn*, vol. 21, 1992.

[5] C. Macías Gelabert and A. Aguilera Martinez, “Contribución de la gestión de recursos humanos a la gestión del conocimiento,” *Estudios gerenciales*, vol. 28, no. 123, pp. 133–148, 2012.

[6] D. P. Lepak, H. Liao, Y. Chung, and E. E. Harden, “A Conceptual Review of Human Resource Management Systems in Strategic Human Resource Management Research,” 2006. doi: 10.1016/S0742-7301(06)25006-0.

[7] A. Bacallao, “La Evaluación Institucional de los Recursos Humanos en las Universidades Cubanas desde el Sistema de Gestión de los Recursos Humanos,” 2019.

[8] J. M. R. Pérez and J. V. Victoria, “Tipos de estrategias y sistemas de gestión de recursos humanos: un análisis de la industria manufacturera española,” *Cuadernos de Econom\ ’\ia y Dirección de la Empresa*, no. 12, pp. 421–437, 2002.

[9] F. Y. H. Ahmed, K. L. T. Aik, A. S. Radzi, and M. D. Salleh, “Develop attendance management system with feedback and complaint management function,” in *2019 IEEE 7th Conference on Systems, Process and Control (ICSPC)*, 2019, pp. 248–252.

[10] S. Li *et al.*, “Understanding and addressing quality attributes of microservices architecture: A Systematic literature review,” Mar. 01, 2021, *Elsevier B.V.* doi: 10.1016/j.infsof.2020.106449.

[11] V. Velepucha and P. Flores, “A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges,” *IEEE Access*, vol. 11, pp. 88339–88358, 2023, doi: 10.1109/ACCESS.2023.3305687.

[12] J. María, G. Haro, J. Maria, and C. Riba, “Diseño e implementación de un marco de trabajo (framework) de presentación para aplicaciones JEE.”

[13] R. Manciola Meloca, R. Re, and A. Luis Schwerz, “An analysis of frameworks for microservices,” in *Proceedings - 2018 44th Latin American Computing Conference, CLEI 2018*, Institute of Electrical and Electronics Engineers Inc., Oct. 2018, pp. 542–551. doi: 10.1109/CLEI.2018.00071.

[14] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003.

[15] F. Rademacher, S. Sachweh, and A. Zündorf, “Towards a UML profile for domain-driven design of microservice architectures,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Verlag, 2018, pp. 230–245. doi: 10.1007/978-3-319-74781-1_17.

[16] J. Tarun, “Design a DDD-oriented microservice.” Accessed: May 09, 2024. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice

[17] B. Shafabakhsh, R. Lagerström, and S. Hacks, “Evaluating the Impact of Inter Process Communication in Microservice Architectures.,” in *QuASoQ@ APSEC*, 2020, pp. 55–63.

[18] A. D. Birrell and B. J. Nelson, “Implementing Remote Procedure Calls,” *1984*.

[19] P. Dobbelaere and K. S. Esmaili, “Industry paper: Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations,” in *DEBS 2017 - Proceedings of the 11th ACM International Conference on Distributed Event-Based Systems*,

Association for Computing Machinery, Inc, Jun. 2017, pp. 227–238. doi: 10.1145/3093742.3093908.

[20] R. T. Yarlagadda, “DevOps and Its Practices,” *International Journal of Creative Research Thoughts (IJCRT), ISSN*, vol. 9, pp. 2320–2882, 2021, [Online]. Available: www.ijcrt.org

[21] G. Bou Ghantous, A. Gill, and G. Bou, “Association for Information Systems AIS Electronic Library (AISeL) DevOps: Concepts, Practices, Tools, Benefits and Challenges Recommended Citation,” *PACIS2017*, p. 1, 2017, [Online]. Available: http://aisel.aisnet.org/pacis2017/96

[22] R. Majumdar, R. Jain, S. Barthwal, and C. Choudhary, “Source code management using version control system,” in *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2017, pp. 278–281. doi: 10.1109/ICRITO.2017.8342438.

[23] M. S. Arefeen and M. Schiller, “Continuous Integration Using Gitlab,” *Undergraduate Research in Natural and Clinical Sciences and Technology Journal*, vol. 3, no. 1–11, Jan. 2019, doi: 10.26685/urncst.152.

[24] C. Vassallo *et al.*, “Continuous delivery practices in a large financial organization,” in *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, Institute of Electrical and Electronics Engineers Inc., Jan. 2017, pp. 519–528. doi: 10.1109/ICSME.2016.72.

[25] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, “How do software developers use github actions to automate their workflows?,” in *Proceedings - 2021 IEEE/ACM 18th International Conference on Mining Software Repositories, MSR 2021*, Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 420–431. doi: 10.1109/MSR52588.2021.00054.

[26] J. Virtanen, “Comparing Different CI/CD Pipelines,” 2021.

[27] X. Xu *et al.*, “Research on security issues of docker and container monitoring system in edge computing system,” in *Journal of Physics: Conference Series*, IOP Publishing Ltd, Nov. 2020. doi: 10.1088/1742-6596/1673/1/012067.

[28] M. Holopainen, “Monitoring Container Environment with Prometheus and Grafana,” 2021.

[29] A. María, F. Redondo, F. De Jesús, and N. Cárdenas, “DevOps: un vistazo rápido DevOps: a quick look,” 2022. [Online]. Available: https://repository.uaeh.edu.mx/revistas/index.php/huejutla/issue/archive

[30] J. A. Scott, “A Practical Guide to Microservices and Containers,” *MapR Data Technologies*, 2018.

[31] D. Verdier and G. Rodr\’\iguez, “Implementación de Patrones de Microservicios,” 2020.

[32] A. Raj and K. Jasmine, “Building Microservices with Docker Compose,” *The International journal of analytical and experimental modal analysis*, vol. 13, pp. 1215–1219, 2021.

[33] K. Klinbua and W. Vatanawood, “Translating tosca into docker-compose yaml file using antlr,” in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2017, pp. 145–148.

[34] H. M. Abdullah and A. M. Zeki, “Frontend and backend web technologies in social networking sites: Facebook as an example,” in *Proceedings - 3rd International Conference on Advanced Computer Science Applications and Technologies, ACSAT 2014*, Institute of Electrical and Electronics Engineers Inc., Apr. 2014, pp. 85–89. doi: 10.1109/ACSAT.2014.22.

[35] M. DE Desarrollo De Software and E. Gabriel Pacienzia, “FACULTAD DE QUÍMICA E INGENIERIA ‘FRAY ROGELIO BACON’ PONTIFICIA UNIVERSIDAD CATÓLICA ARGENTINA SANTA MARIA DE LOS BUENOS AIRES Cátedra Seminario de Sistemas.”

[36] P. Letelier and M. Carmen Penadés, “Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP).” [Online]. Available: www.agileuniverse.com.

[37] E. Bautista-Villegas, “Metodologías agiles XP y Scrum, empleadas para el desarrollo de páginas web, bajo MVC, con lenguaje PHP y framework Laravel,” *Revista Amazonía Digital*, vol. 1, no. 1, p. e168, Jan. 2022, doi: 10.55873/rad.v1i1.168.

[38] L. M. Echeverry Tobón and L. E. Delgado Carmona, “CASO PRÁCTICO DE LA METODOLOGÍA ÁGIL XP AL DESARROLLO DE SOFTWARE,” UNIVERSIDAD TECNOLÓGICA DE PEREIRA, 2007.

[39] D. F. Sanmartin Montaño, “Prototipo de un Repositorio Digital para la consulta de datos científicos de especímenes del ‘Jardín Botánico Reinaldo Espinosa,’” Universidad Nacional de Loja, Loja, 2024. Accessed: Jul. 18, 2024. [Online]. Available: https://dspace.unl.edu.ec/jspui/handle/123456789/29331

[40] C. A. Salazar Valdez, “Sistema web para la gestión y monitoreo de las denuncias de acoso escolar en la Unidad Educativa Lauro Damerval Ayora,” Universidad Nacional de Loja, Loja, 2024. Accessed: Jul. 14, 2024. [Online]. Available: https://dspace.unl.edu.ec/jspui/handle/123456789/29173

[41] N. F. Salinas Minga, “Desarrollo de una aplicación web para la gestión de materia prima y control de ventas en la empresa Breldy del cantón Yantzaza,” Universidad Nacional de Loja, Loja, 2023. Accessed: Jul. 14, 2024. [Online]. Available: https://dspace.unl.edu.ec/jspui/handle/123456789/28240

[42] P. Kruchten, “Planos Arquitectónicos: El Modelo de ‘4+1’ Vistas de la Arquitectura del Software *,” *IEEE Softw*, vol. 12, pp. 42–50, 1995.

11. Anexos

Anexo 1. Requerimientos entregados por la empresa PuntoPymes CIA. LTDA.

PuntoPymes
Software development center

Azuay 1257 y Olmedo
0963319244
www.sofpymes.com

Requerimientos de Software

Plataforma GeoWorker

Descripción

La compañía PUNTOPYMES CIA LTDA., busca mejorar su proceso de control de personal y talento humano con geolocalización del área de trabajo, con el objetivo de actualizar los servicios orientados a una plataforma de servicios WEB integrado al sistema ERP de la empresa con métricas de eficiencia y productividad de tareas por el personal, integrado a una APP móvil para el personal, misma que le permite registrar los procesos de las tareas asignadas mejorando la eficiencia operativa y la toma de decisiones.

Objetivos

Mejorar la Eficiencia Operativa:

- Automatizar el registro de horas de entrada y salida de los empleados y gestionar fácilmente sus horas de trabajo para reducir errores y ahorrar tiempo en el procesamiento manual.

Transparencia y Precisión de Tareas de Trabajo:

- Gestionar de manera precisa las tareas de los empleados, garantizando claridad y transparencia de cada tarea asignada, permitiendo que cada empleado este en el lugar adecuado y momento correcto.

Automatización de permisos laborales o vacaciones:

- Automatizar los procesos de solicitudes mejorando la comunicación interna y reduciendo los tiempos de respuesta.

Cumplimiento de tareas:

- Asegurar que todas las tareas asignadas a los colaboradores se realicen de manera correcta y efectiva.

A continuación, se detallan las principales entidades que conformará la plataforma:

Entidades Principales

1. Empresa

La presente entidad representa las empresas (clientes), los cuales harán uso de la plataforma "Geoworker".

Campos

- **Nombre:** Almacena el nombre de la empresa.
- **País:** Almacena el país donde se ubica la empresa.
- **Ciudad:** Almacena la ciudad donde se ubica la empresa.
- **Dirección:** Contiene la dirección de la empresa
- **Estado:** Sirve para representar el estado de la empresa dentro del sistema. Puede tomar dos valores que son: Activo e Inactivo.

2. Usuario

En esta entidad se realiza la gestión de los empleados de cada empresa que use la plataforma.

Campos

- **Nombre:** Almacena los nombres del usuario.
- **Apellido:** Almacena los apellidos del usuario.
- **DNI:** Identificador del usuario (Cédula)
- **Correo:** Almacena el correo electrónico del usuario.
- **Cargo:** Cargo del usuario (Diseñador, Desarrollador, Contador)
- **Dirección:** Dirección del usuario
- **Contacto:** Numero de contacto del usuario
- **Contraseña:** Almacena la contraseña del usuario.
- **Imagen de perfil:** Almacena la foto de perfil del usuario.
- **Estado:** Sirve para gestionar la interacción del usuario dentro de la plataforma.

- **Rol:** Almacena el rol del usuario, el mismo que permitirá limitar las acciones de un usuario dentro de la plataforma.

3. Proyecto

Las empresas podrán crear proyectos dentro de la plataforma, de esta manera se garantiza una organización eficiente dentro de la misma.

Campos

- **Nombre:** Almacena el nombre del proyecto
- **Líder de Equipo:** Debe almacenar al usuario al que se le encargará el proyecto de la empresa. Además, podrá añadir usuarios a su proyecto.
- **Miembros:** Son los usuarios que son añadidos por el líder del equipo al proyecto que se le encargó
- **Descripción:** Almacena la información referente al proyecto.
- **Fecha de Inicio:** Fecha en la que inicia el proyecto
- **Fecha Final:** Fecha en la que finaliza el proyecto
- **Estado:** Gestiona el estado del proyecto (Activo o Inactivo, Completado).

4. Tarea

Cada proyecto de la empresa se podrá dirigir o administrar mediante tareas. Cada tarea permitirá medir el trabajo individual o colectivo de los usuarios vinculados al proyecto.

Campos

- **Título:** Almacena el título de la tarea
- **Descripción:** Almacena la descripción de la tarea, o detalles para el cumplimiento de esta.
- **Estado:** Sirve para determinar el estado de la tarea, puede adquirir 4 estados fijos (Por hacer, En progreso, En revisión, Realizado)

- **Tags:** Almacena cadenas de texto informativo que darán breves descripciones de la tarea.
- **Archivos:** En cada tarea se puede adjuntar archivos, ya sean imágenes, archivos de textos, videos o audios.
- **Fecha límite:** Almacena la fecha límite establecida para el cumplimiento de la tarea.
- **Usuario Creador:** Este campo almacena el usuario que crea la tarea.
- **Usuarios Ejecutores:** Aquí se almacena los usuarios a los que se les encarga la realización o ejecución de la tarea.

5. Rol

Entidad que permite la autenticación y autorización de los usuarios dentro de la plataforma.

Campos

- **Nombre:** Almacena el nombre del rol
- **Módulos:** Almacena los módulos que un rol tiene o puede acceder dentro de la plataforma.

6. Solicitud de Permisos o Vacaciones

Los usuarios de una empresa pueden solicitar permisos o vacaciones desde la plataforma.

Campos

- **Usuario:** Almacena el usuario que solicita el permiso.
- **Tipo de Permiso:** Almacena el tipo de permiso solicitado (Permiso, Vacación)
- **Detalle:** Almacena el detalle del permiso, es decir, el usuario puede describir la razón de por qué se solicita el permiso.
- **Fecha de solicitud:** Almacena la fecha que se solicitó el permiso
- **Fecha del permiso:** Es la fecha que el usuario requiere el permiso.

- **Fecha de reintegro:** Fecha de reintegro al trabajo del usuario
- **Archivo:** El usuario puede cargar un archivo que sirva como prueba o evidencia para que se evalúe su solicitud de permiso.
- **Respuesta:** El líder de equipo puede escribir un breve detalle en la respuesta de la solicitud.
- **Estado:** La solicitud de permisos puede tomar 3 estados fijos (Pendiente, Aprobado, Rechazado).

7. Geo-cerca

La empresa puede controlar la ubicación de sus usuarios a través de Geocercas, es decir, se toma un área de referencia misma que representa un área de trabajo para el usuario.

Campos

- **Nombre:** Almacena el nombre de la Geo-cerca.
- **Latitud:** Latitud del punto de la Geo-cerca.
- **Longitud:** Longitud del punto de la Geo-cerca.
- **Radio:** Valor en metros que permitirá establecer el área de trabajo del usuario.
- **Horario:** Hora de entrada y salida del usuario.
- **Tiempo de descanso:** tiempo que el usuario puede tomarse como descanso.

Requerimientos

#	Requerimiento	Descripción	Rol
		EMPRESA	
1	CRUD	Crear, Actualizar, Listar y Eliminar	SUPER ADMIN
		USUARIOS	
2	CRUD	Crear, Actualizar, Listar, Eliminar	ADMINISTRADOR
3	Cargar usuarios	Cargar usuarios desde un archivo CSV	ADMINISTRADOR

		PROYECTO	
4	CRUD	Crear, Actualizar, Listar, Eliminar	ADMINISTRADOR, GESTOR
5	Añadir miembro al equipo	Añadir usuarios de la empresa como miembro del equipo que estará asignado al proyecto	ADMINISTRADOR, GESTOR
6	Listar Equipo	Listar los miembros que estén asignados al proyecto	ADMINISTRADOR
		TAREA	
7	CRUD	Crear, Actualizar, Listar, Eliminar	ADMINISTRADOR, GESTOR, MIEMBRO
8	Actualizar estado	Cambiar estado de tarea (POR HACER, EN PROGRESO, EN REVISIÓN, REALIZADA)	ADMINISTRADOR, GESTOR, MIEMBRO
9	Filtrar tareas	Filtrar tareas, por usuario, por proyecto y por estado	ADMINISTRADOR, GESTOR, MIEMBRO
		GEOCERCA	
10	CRUD	Crear, Actualizar, Listar, Eliminar	ADMINISTRADOR, GESTOR,
11	Añadir usuario	Vincular una geo-cerca a uno o varios usuarios	ADMINISTRADOR, GESTOR
12	Ver actividad	Visualizar las personas que están dentro de la geo-cerca	ADMINISTRADOR, GESTOR
		REGISTRO DE HORAS	
13	CRUD	Crear, Actualizar, Listar, Eliminar	ADMINISTRADOR, GESTOR, MIEMBRO
14	Reporte de horas trabajadas	Reporte de horas trabajadas en la semana	ADMINISTRADOR, GESTOR, MIEMBRO
		SOLICITUD DE PERMISO O VACACIONES	
15	CRUD	Crear, Actualizar, Listar, Eliminar	GESTOR, MIEMBRO
16	Responder solicitud	Responder a la solicitud de permiso o vacaciones adjuntando un detalle o razón de la respuesta	ADMINISTRADOR, GESTOR

17	Definir la cantidad de permisos que se puede solicitar	Cada empresa puede definir la cantidad de permisos que puede solicitar un miembro mensualmente	ADMINISTRADOR
		AUDITORIA	
18	Guardar registros de acciones	Almacenar logs para auditorias de las acciones que se realicen en el sistema	
19	Ver registros	Visualizar los registros guardados por el sistema de auditoria	SUPER ADMIN

JONATHAN FERNANDO ZHUNAULA ANGAMARCA

Sr. Jonathan Zhunaula Angamarca
REPRESENTANTE LEGAL COMPAÑÍA PUNTOPYMES CIA LTDA
RUC: 1191732789001

Anexo 2. Especificación de Requisitos de software

Especificación de Requisitos de Software

Proyecto

Implementación de una arquitectura de microservicios y técnicas de DevOps a los servicios backend de la plataforma GeoWorker

Elaborado por:

Manuel Alejandro Aguinsaca Medina

Mayo de 2024

1. Introducción

El presente documento es una Especificación de Requisitos de Software (ERS) de la plataforma "GeoWorker", la misma que se basado en los directrices descritas en el estándar IEEE Práctica Recomendada para Especificaciones de Requisitos Software ANSI/IEEE 830, 1998.

1.1. Propósito

Este documento tiene como propósito el definir de una manera sistemática y precisa los requisitos funcionales y no funcionales de la plataforma "GeoWorker", la cual estará destinada a la gestión de los recursos de una empresa, tales como: usuarios, proyectos, tareas, horarios y permisos.

1.2. Alcance

Esta especificación de requisitos está enfocada en las empresas que necesitan una plataforma para administrar de una mejor manera sus recursos, es decir, permitir a empresas organizar todo su personal, datos y procesos desde un mismo sistema. Así mismo, permitir a los empleados organizar sus tareas, sus horarios y las solicitudes de permisos y vacaciones de una manera más eficiente.

1.3. Personal involucrado

Nombre	Manuel Aguinsaca
Rol	Analista, Diseñador y Programador
Categoría profesional	Tesista - Ingeniero en Sistema
Responsabilidad	Análisis de información, diseño y programador del sistema
Información de contacto	maaguinsacam@unl.edu.ec

1.4. Definiciones, acrónimos y abreviaturas

Nombre	Descripción
Usuario	Persona que usará la plataforma y sus funciones
ERS	Especificación de Requisitos de Software
RF	Requerimientos Funcionales
RNF	Requerimientos No Funcionales

1.5. Referencias

Referencia	Título del Documento
Requisitos	Especificación de Requerimientos
Modelo	Arquitectura de Software 4+1
IEEE	Estándar IEEE 830 - 1998

1.6. Resumen

El presente documentos se estructura de 3 secciones principales. En la primera sección se realiza una introducción y proporciona una visión general de la especificación del sistema.

La segunda sección, otorga una descripción general de la plataforma, es decir, detalla las principales funciones con las que va a contar el sistema, los datos a tratarse y restricciones que afectaran al desarrollo de la plataforma, pero sin entrar en grandes detalles.

La última sección, es la que contendrá los requisitos de la plataforma que deben ser cumplidos, estructurándolos de una forma más detallada y organizada.

2. Descripción General

2.1. Perspectiva

La plataforma “GeoWorker” será un sistema que permita a las empresas administrar recursos como: empleados, proyectos, tareas, solicitudes (permisos, vacaciones) de manera online, los procesos con los que contará el sistema serán los siguientes:

- Creación de Empresas
- Administración de Usuarios
- Gestión de Proyectos
- Registro de Tareas
- Creación de solicitudes

2.2. Funcionalidad del producto

La plataforma GeoWorker cuenta con las siguientes funcionalidades:

- **Autenticación de Usuarios:** Los usuarios podrán iniciar sesión mediante un correo y contraseña.
- **Registro de Empresas:** Las empresas podrán ser registradas en la base de datos, permitiendo editar y eliminar su información.
- **Administración de Usuarios:** Cada empresa podrá, crear, listar, editar y eliminar usuarios, de la misma manera, podrán asignar un rol que permitirá establecer los permisos que tendrán los usuarios.
- **Gestión de Proyectos:** Los administradores de las empresas crear proyectos dentro de sus empresas, asignando usuarios como encargados de estos proyectos, los cuales podrán administrar tarea y usuarios dentro del mismo proyecto.
- **Gestión de Tareas:** Los usuarios que pertenezcan a un proyecto, podrán visualizar, crear y modificar la información de las tareas, así como también el estado actual de dicha tarea.

2.3. Características de los usuarios

Tipo de Usuario	Usuario Administrador
Formación	Ninguna
Habilidades	Conocimientos básicos de manejo de computadores, plataformas o aplicaciones web.
Actividades	Administración y soporte de plataforma

Tipo de Usuario	Encargado de la Empresa
Formación	Ninguna
Habilidades	Conocimientos básicos de manejo de computadores, plataformas o aplicaciones web.
Actividades	Administración de los datos de su empresa, usuarios, horarios, roles, proyectos y tareas

Tipo de Usuario	Usuario del Sistema
Formación	Ninguna
Habilidades	Conocimientos básicos de manejo de computadores, plataformas o aplicaciones web.
Actividades	Gestión de tareas, permisos, y horarios

2.4. Restricciones

- Las contraseñas deben estar encriptadas en la base de datos
- Los procesos son dependientes de una conexión a internet.
- Los servicios deben ser independientes entre sí, es decir, la falla de un servicio no debe afectar al resto.
- Los servicios deben permitir conexiones y consultas concurrentes
- Las acciones de los usuarios se registrarán en un servicio independiente, como logs de auditoría.

2.5. Suposiciones y dependencias

- Se asume que los requisitos son estables.
- El servidor donde se vaya a desplegar los servicios backend, deben cumplir con los requisitos antes mencionados, de esta manera de garantizará el correcto funcionamiento de la plataforma.

3. Requisitos Específicos

3.1. Requerimientos Funcionales

Requerimientos Funcionales	
Código	**Requisito**
RF001	El sistema permitirá registrar empresas
RF002	El sistema permitirá modificar la información de las empresas
RF003	El sistema permitirá listar las empresas
RF004	El sistema permitirá eliminar empresas
RF005	El sistema permitirá al encargado de la empresa registrar usuarios
RF006	El sistema permitirá al encargado de la empresa modificar usuarios
RF007	El sistema permitirá al encargado de la empresa listar sus usuarios
RF008	El sistema permitirá al encargado de la empresa eliminar sus usuarios
RF009	El sistema permitirá al encargado de la empresa cargar usuarios desde un archivo .csv

RF010	El sistema permitirá al encargado de la empresa crear proyectos
RF011	El sistema permitirá al encargado de la empresa listar proyectos
RF012	El sistema permitirá al encargado de la empresa modificar proyectos
RF013	El sistema permitirá al encargado de la empresa eliminar proyectos
RF014	El sistema permitirá al encargado de la empresa asignar un usuario como líder del proyecto
RF015	El sistema permitirá al líder del proyecto listar los usuarios que pertenecen al proyecto
RF016	El sistema permitirá al líder del proyecto crear tareas a los usuarios asignados al proyecto
RF017	El sistema permitirá al líder del proyecto eliminar tareas
RF018	El sistema permitirá a los usuarios modificar los datos de una tarea
RF019	El sistema permitirá a los usuarios visualizar los datos de una tarea
RF020	El sistema permitirá al líder del proyecto listar las tareas que pertenecen al proyecto
RF021	El sistema permitirá al encargado de una empresa crear geocercas
RF022	El sistema permitirá al encargado de una empresa modificar geocercas
RF023	El sistema permitirá al encargado de una empresa listar las geocercas
RF024	El sistema permitirá al encargado de una empresa asignar una geocerca a un usuario
RF025	El sistema permitirá visualizar los usuarios que estén dentro de una geocerca
RF026	El sistema permitirá a un usuario solicitar permiso o vacaciones
RF027	El sistema permitirá responder solicitudes de permisos o vacaciones
RF028	El sistema permitirá al encargado de la empresa definir cuantas solicitudes puede crear un usuario mensualmente
RF029	El sistema permitirá registrar la entrada o salida de la jornada laboral
RF030	El sistema permitirá modificar el registro de entradas y salidas de un usuario
RF031	El sistema permitirá visualizar el reporte de horas trabajadas de un usuario
RF032	El sistema almacenará las acciones que realicen los usuarios
RF033	El sistema permitirá al usuario iniciar sesión mediante correo y contraseña

Número del Requisito	RF001, RF002, RF003, RF004
Nombre del Requerimiento	Administrar Empresas
Descripción del Requisito	El usuario administrador puede: Crear empresas Listar Empresas Modificar Empresas Eliminar Empresas
Prioridad del requisito	Alta

Número del Requisito	RF005, RF006, RF007, RF008
Nombre del Requerimiento	Administrar Usuarios
Descripción del Requisito	El encargado de la empresa puede: Crear Usuarios Listar Usuarios Modificar Usuarios Eliminar Usuarios
Prioridad del requisito	Alta

Número del Requisito	RF009
Nombre del Requerimiento	Cargar Usuarios
Descripción del Requisito	El encargado de la empresa puede: Cargar usuarios desde un archivo CSV delimitado por comas, con los siguientes campos, nombres, correo, cedula, contraseña

Prioridad del requisito	Alta

Número del Requisito	RF010, RF011, RF012, RF013
Nombre del Requerimiento	Administrar Proyectos
Descripción del Requisito	El encargado de la empresa puede: Crear Proyectos Editar Proyectos Listar Proyectos Eliminar Proyectos
Prioridad del requisito	Alta

Número del Requisito	RF014
Nombre del Requerimiento	Asignar Líder de Proyecto
Descripción del Requisito	El encargado de la empresa puede asignar a un usuario como líder del proyecto, de esta forma distribuir las obligaciones de los proyectos
Prioridad del requisito	Alta

Número del Requisito	RF015
Nombre del Requerimiento	Listar Usuarios por Proyecto
Descripción del Requisito	El líder de un proyecto puede listar los usuarios asignados a su proyecto.
Prioridad del requisito	Alta

Número del Requisito	RF016, RF017, RF018, RF019, RF020
Nombre del Requerimiento	Administrar Tareas
Descripción del Requisito	El usuario puede: Crear Tareas Modificar Tareas Listar Tareas Eliminar Tareas
Prioridad del requisito	Alta

Número del Requisito	RF021, RF022, RF023, RF024
Nombre del Requerimiento	Administrar Geocercas
Descripción del Requisito	El usuario puede: Crear Geocercas Editar Geocercas Eliminar Geocercas Asignar Geocercas
Prioridad del requisito	Alta

Número del Requisito	RF025
Nombre del Requerimiento	Visualizar Usuarios de una Geocerca
Descripción del Requisito	El encargado de la empresa puede visualizar los usuarios activos de una geocerca
Prioridad del requisito	Alta

Número del Requisito	RF026
Nombre del Requerimiento	Solicitar Permiso
Descripción del Requisito	El usuario puede solicitar permiso o vacaciones
Prioridad del requisito	Alta

Número del Requisito	RF027
Nombre del Requerimiento	Responder Permiso
Descripción del Requisito	El encargado de la empresa puede responder una solicitud de un usuario
Prioridad del requisito	Alta

Número del Requisito	RF028
Nombre del Requerimiento	Configurar Permisos
Descripción del Requisito	El encargado de la empresa puede configurar el número de permisos o vacaciones que un usuario puede solicitar al mes
Prioridad del requisito	Alta

Número del Requisito	RF029, RF030, RF031
Nombre del Requerimiento	Administrar Horario
Descripción del Requisito	El usuario puede: Registrar su entrada o salida Modificar su registro de entrada o salida Ver reporte de horas trabajadas por fechas
Prioridad del requisito	Alta

Número del Requisito	RF032
Nombre del Requerimiento	Auditoria de Sistema
Descripción del Requisito	El sistema almacenará logs de las acciones realizadas por los usuarios detallando, fecha y hora y la acción realizada
Prioridad del requisito	Alta

Número del Requisito	RF033
Nombre del Requerimiento	Autenticar Usuario
Descripción del Requisito	Los usuarios deberán identificarse con la plataforma mediante correo y contraseña
Prioridad del requisito	Alta

3.2. Requerimientos no Funcionales

3.2.1. Seguridad

Los servicios de la plataforma GeoWorker, serán capaces de controlar el acceso a las funcionalidades por medio de permisos y roles.

3.2.2. Disponibilidad

Los servicios de la plataforma deben funcionar las 24 horas del día y los 7 días de la semana, a menos que se encuentren en mantenimiento o actualización.

3.2.3. Usabilidad

Los servicios deben retornar los respectivos mensajes de error e información, de esta forma, ayudar al usuario a guiar en los procesos que este ejecutando.

Historias de Usuarios

ID: H001	**AUTENTICAR USUARIO**		
Usuario/Rol	Usuario	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Usuario, **QUIERO** iniciar sesión usando usuario/correo y contraseña, **PARA** poder acceder a las funciones del sistema		
Criterios de aceptación			
• Para iniciar sesión debe estar previamente registrado en el sistema. • Para iniciar sesión puedo enviar el correo o usuario junto con la contraseña. • La respuesta debe contener un JWT donde conste la información del usuario y su ROL. • Si las credenciales enviadas son incorrectas, se presenta el mensaje "Credenciales inválidas"			

<table>
<tr><th>ID: H002</th><th colspan="3">CREAR EMPRESA</th></tr>
<tr><td rowspan="2">Usuario/Rol</td><td rowspan="2">SUPER ADMIN</td><td>Prioridad</td><td>Alta</td></tr>
<tr><td>Puntos estimados</td><td></td></tr>
<tr><td>Descripción</td><td colspan="3">COMO Super administrador, QUIERO registrar una empresa, PARA poder listar las empresas disponibles en el sistema</td></tr>
<tr><td colspan="4">Criterios de aceptación</td></tr>
<tr><td colspan="4">

El campo nombre debe tener un máximo de 100 caracteres.
El campo nombre debe tener un mínimo de 5 caracteres.
El campo nombre debe ser obligatorio.
El campo nombre debe ser único, si existe otra empresa con el mismo nombre, se presenta el mensaje “La empresa ya existe”
El país debe ser el identificador válido de un país registrado en el sistema.
El campo ciudad puede tener máximo 100 caracteres.
El campo ciudad n puede estar vacío.
El campo dirección debe tener máximo 100 caracteres.
El campo dirección no puede estar vacío.
El estado por defecto de una empresa es ACTIVA.
A la empresa se le asignara un UUID único
Se creará tres roles para la empresa (ADMINISTRADOR, GESTOR, MIEMBRO)

</td></tr>
</table>

ID: H003	ACTUALIZAR EMPRESA		
Usuario/Rol	SUPER ADMIN	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Super administrador, **QUIERO** actualizar una empresa, **PARA** poder modificar los datos de una empresa registrada en el sistema.		
Criterios de aceptación			
• El campo nombre debe tener un máximo de 100 caracteres. • El campo nombre debe tener un mínimo de 5 caracteres. • El campo nombre debe ser obligatorio. • El campo nombre debe ser único, si existe otra empresa con el mismo nombre, se presenta el mensaje “La empresa ya existe” • El país debe ser el identificador válido de un país registrado en el sistema. • El campo ciudad puede tener máximo 100 caracteres. • El campo ciudad n puede estar vacío. • El campo dirección debe tener máximo 100 caracteres. • El campo dirección no puede estar vacío. • El campo estado, acepta valores booleanos.			

ID: H004	LISTAR EMPRESA		
Usuario/Rol	SUPER ADMIN	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Super administrador, **QUIERO** listar empresas, **PARA** poder conocer las empresas que están disponibles en la plataforma		
Criterios de aceptación			
• Puedo listar las empresas activas, inactivas o todas.			

ID: H005	ELIMINAR EMPRESA		
Usuario/Rol	SUPER ADMIN	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Super administrador, **QUIERO** eliminar una empresa, **PARA** que no se listen los registros de dicha empresa o eliminar datos creados incorrectamente.		
Criterios de aceptación			
• Se tiene que enviar el UUID de la empresa a eliminar en la url de la petición. • Si el UUID no es válido, se muestra el mensaje “Datos inválidos”.			

ID: H006	REGISTRAR USUARIO		
Usuario/Rol	ADMINISTRADOR	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Administrador, **QUIERO** registrar un usuario, **PARA** que pueda iniciar sesión en la plataforma y hacer uso de sus funcionalidades.		
Criterios de aceptación			
• El campo nombre debe tener máximo 100 caracteres y mínimo 5. • El campo empresa, debe tener un UUID válido de una empresa activa. • El campo correo debe contener un correo electrónico válido. • El campo contraseña debe tener máximo 16 caracteres y mínimo 5. • El campo estado debe tener un valor booleano. • El campo rol, debe tener un ID del rol GESTOR o MIEMBRO. • El campo foto, debe tener un archivo tipo imagen que se subirá a un servidor externo. • El campo foto es opcional.			

ID: H007	REGISTRAR ADMINISTRADOR DE EMPRESA		
Usuario/Rol	SUPER ADMIN	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** SUPER ADMIN, **QUIERO** registrar un administrador de una empresa, **PARA** poder designar a un encargado de la empresa y que pueda acceder a las funciones específicas de su rol.		
Criterios de aceptación			
• El campo nombre debe tener máximo 100 caracteres y mínimo 5. • El campo empresa, debe tener un UUID válido de una empresa activa. • El campo correo debe contener un correo electrónico válido. • El campo contraseña debe tener máximo 16 caracteres y mínimo 5. • El campo estado debe tener un valor booleano. • El campo rol, debe tener un ID del rol ADMINISTRADOR. • El campo foto, debe tener un archivo tipo imagen que se subirá a un servidor externo. • El campo foto es opcional.			

ID: H008	BUSCAR USUARIOS		
Usuario/Rol	ADMINISTRADOR	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** ADMINISTRADOR, **QUIERO** poder listar los usuarios, **PARA** realizar acciones como actualizar o eliminar.		
Criterios de aceptación			
• Se debe buscar los usuarios por el criterio de búsqueda como: Estado, Empresa, Proyecto.			

ID: H009	ACTUALIZAR USUARIO		
Usuario/Rol	ADMINISTRADOR	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** ADMINISTRADOR, **QUIERO** poder actualizar un usuario, **PARA** corregir datos erróneos o actualizar datos de un usuario.		
Criterios de aceptación			
• El UUID del usuario se debe enviar en la url de la petición. • El campo nombre debe tener máximo 100 caracteres y mínimo 5. • El campo correo debe contener un correo electrónico válido. • El correo electrónico no debe estar asignando a otro usuario. • El campo estado debe tener un valor booleano.			

<table>
<tr><th>ID: H010</th><th colspan="3">OBTENER PERFIL DEL USUARIO</th></tr>
<tr><td rowspan="2">Usuario/Rol</td><td rowspan="2">Usuario</td><td>Prioridad</td><td>Alta</td></tr>
<tr><td>Puntos estimados</td><td></td></tr>
<tr><td>Descripción</td><td colspan="3">COMO Usuario, QUIERO poder obtener mi perfil, PARA poder obtener la información completa de mi perfil.</td></tr>
<tr><td colspan="4">Criterios de aceptación</td></tr>
<tr><td colspan="4">• Se debe enviar el JWT en la cabecera de autorización de la petición.</td></tr>
</table>

ID: H011	CREAR TAREA		
Usuario/Rol	Usuario	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Usuario, **QUIERO** poder crear una tarea, **PARA** poder asignar actividades a un usuario de la empresa.		
Criterios de aceptación			
• El campo título debe contener máximo 100 caracteres y mínimo 10. • El campo descripción debe tener mínimo 10n caracteres y máximo 255. • El campo estado debe contener un identificador válido de la lista de estados disponibles. • El campo fecha límite, debe contener una fecha válida. • El campo creador debe tener el UUID de un usuario válido. • El campo ejecutor, debe tener una lista de UUIDS válidos de usuarios. • El campo proyecto, debe tener un UUID válido de un proyecto del sistema. • El campo tags debe tener una lista de palabras para describir brevemente a la tarea. • El campo tags es opcional. • El campo archivos debe contener una lista de archivos que se subirán a un servidor externo. • El campo archivos es opcional.			

ID: H012	LISTAR TAREAS		
Usuario/Rol	Usuario	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Usuario, **QUIERO** poder listar tareas, **PARA** poder visualizar las tareas asignadas a los usuarios de una empresa.		
Criterios de aceptación			
• La búsqueda de tareas se debe realizar por criterio de búsqueda como: Estado y Proyecto. • Los usuarios solo pueden ver sus tareas o las tareas de la empresa a la que pertenece.			

ID: H013	ACTUALIZAR ESTADO DE TAREA		
Usuario/Rol	Usuario	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Usuario, **QUIERO** poder actualizar el estado de una tarea, **PARA** poder llevar un control de las tareas que se están realizando, revisando o ya estén terminadas.		
Criterios de aceptación			
• El UUID de la tarea debe enviarse en la url de la petición. • El campo estado debe ser enviado en el cuerpo de la petición. • El campo estado debe contener un identificador válido de la lista de estados disponibles en la plataforma.			

<table>
<tr><th>ID: H014</th><th colspan="3">CREAR PROYECTO</th></tr>
<tr><td rowspan="2">Usuario/Rol</td><td rowspan="2">ADMINISTRADOR</td><td>Prioridad</td><td>Alta</td></tr>
<tr><td>Puntos estimados</td><td></td></tr>
<tr><td>Descripción</td><td colspan="3">COMO ADMINISTRADOR, QUIERO poder crear un proyecto, PARA poder dividir las actividades o tareas que tiene que realizar mi empresa.</td></tr>
<tr><td colspan="4">Criterios de aceptación</td></tr>
<tr><td colspan="4">Al proyecto se le asignará un UUID único.
El campo nombre debe tener un mínimo de 10 caracteres y máximo 100.
El campo descripción debe contener mínimo 10 caracteres y máximo 255.
El campo fecha de inicio debe contener una fecha válida.
El campo fecha fin debe tener una fecha válida.
El campo fecha inicio debe ser menos a la fecha fin.
En el cuerpo de la petición se debe enviar el UUID de la empresa a la que va a pertenecer el proyecto.</td></tr>
</table>

ID: H015	CREAR LIDER DE PROYECTO		
Usuario/Rol	ADMINISTRADOR	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Administrador, **QUIERO** designar un líder de proyecto, **PARA** tener un encargado de los proyectos que tenga en mi empresa.		
Criterios de aceptación			
• El campo empresa debe tener el UUID de una empresa válida. • El campo líder de equipo debe tener un UUID de un usuario válido de la empresa. • El campo proyecto, debe tener un UUID válido de un proyecto y que pertenezca a la empresa. • Un usuario puede ser líder de equipo de varios proyectos de la empresa.			

ID: H016	AÑADIR MIEMBRO AL PROYECTO		
Usuario/Rol	GESTOR	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Gestor, **QUIERO** añadir un usuario al proyecto, **PARA** designar tareas al usuario y controlar sus actividades desde la plataforma.		
Criterios de aceptación			
• El UUID del proyecto debe ir en la url de la petición. • El campo usuario, debe tener el UUID de un usuario válido de le empresa. • El gestor del proyecto puede visualizar su equipo, es decir, los usuarios asignados a un proyecto.			

ID: H017	AÑADIR MIEMBRO AL PROYECTO		
Usuario/Rol	GESTOR	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Gestor, **QUIERO** poder remover un usuario del proyecto, **PARA** mantener mi proyecto organizado y con el personal necesario para su ejecución.		
Criterios de aceptación			
• El UUID del proyecto debe ir en la url de la petición. • El campo usuario, debe tener el UUID de un usuario válido de le empresa.			

ID: H018	SOLICITAR PERMISO		
Usuario/Rol	Usuario	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Usuario, **QUIERO** poder solicitar un permiso, **PARA** avisar a mi encargado o administrador de la empresa que requiero un permiso o vacaciones y que quede guardado en el sistema		
Criterios de aceptación			
• Al permiso se le asignará un UUID único en el sistema. • El campo tipo puede contener dos valores, (vacación, permiso) • El campo tipo, es requerido. • El campo detalle debe tener mínimo 10 caracteres y máximo 255. • El campo detalle es requerido. • El campo fecha debe tener una fecha válida. • El campo archivo y debe contener un archivo valido.			

ID: H019	RESPONDER PERMISO		
Usuario/Rol	GESTOR	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** GESTOR, **QUIERO** poder responder un permiso, **PARA** avisar al usuario solicitante si se aceptó o rechazo el permiso que solicito.		
Criterios de aceptación			
• EL UUID del permiso debe estar presente en la url de la petición. • El campo respuesta debe contener mínimo 10 caracteres y máximo 255. • EL campo estado debe contener un valor válido (RECHAZADO, ACEPTADO)			

<table>
<tr><th>ID: H020</th><th colspan="3">REGISTRAR ENTRADA/SALIDA</th></tr>
<tr><td rowspan="2">Usuario/Rol</td><td rowspan="2">Usuario</td><td>Prioridad</td><td>Alta</td></tr>
<tr><td>Puntos estimados</td><td></td></tr>
<tr><td>Descripción</td><td colspan="3">COMO Usuario, QUIERO poder registrar mi entrada/salida, PARA dejar registrado las horas trabajadas en el día y la semana.</td></tr>
<tr><td colspan="4">Criterios de aceptación</td></tr>
<tr><td colspan="4">• El cuerpo de la petición debe contener el UUID del usuario al que se le va a registrar la entrada o salida.
• El campo nombre de usuario debe tener el nombre del usuario al que se registrará la entrada o salida.
• El campo tipo, corresponde al tipo de registro que se va a almacenar (ENTRADA, SALIDA).
• El campo fecha, debe tener una fecha válida.</td></tr>
</table>

<table>
<tr><th>ID: H021</th><th colspan="3">CREAR GEO-CERCA</th></tr>
<tr><td rowspan="2">Usuario/Rol</td><td rowspan="2">GESTOR</td><td>Prioridad</td><td>Alta</td></tr>
<tr><td>Puntos estimados</td><td></td></tr>
<tr><td>Descripción</td><td colspan="3">COMO Gestor, QUIERO crear una geo cerca, PARA controlar y designar áreas de trabajo a los empleados de mi empresa.</td></tr>
<tr><td colspan="4">Criterios de aceptación</td></tr>
<tr><td colspan="4">El campo UUID de la empresa debe tener un UUID válido de una empresa activa.
El campo nombre, debe tener un mínimo de 10 caracteres y máximo 50.
El nombre debe ser único dentro de la empresa.
El campo coordenadas, debe ser un array numérico de 2 posiciones, las cuales representan latitud y longitud.
El campo radio, debe tener un número entero positivo y debe ser mayor a 5.</td></tr>
</table>

ID: H022	VERIFICAR GEO-CERCA		
Usuario/Rol	Usuario	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Usuario, **QUIERO** verificar si estoy en dentro de mi geo-cerca asignada, **PARA** poder marcar mis entradas y salidas.		
Criterios de aceptación			
• El campo empresa, debe tener un UUID válido de una empresa activa. • El campo geocerca, debe tener el ID de la geocerca asignada al usuario. • El campo latitud, debe tener un número. • El campo longitud debe tener un número. • El usuario debe tener una geocerca asignada.			

ID: H023	LISTAR GEOCERCA		
Usuario/Rol	ADMINISTRADOR	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Administrador, **QUIERO** listar las geocercas de mi empresa, **PARA** conocer los espacios de trabajo de mis empleados.		
Criterios de aceptación			
• El criterio de búsqueda es por la empresa. • El UUID de la empresa se envía en la url de la petición. • La empresa debe estar activa en la plataforma.			

ID: H025	GUARDAR LOGS		
Usuario/Rol	Usuario	**Prioridad**	Alta
		Puntos estimados	
Descripción	**COMO** Usuario, **QUIERO** guardar logs de las acciones que realizo, **PARA** poder saber que usuario realizo acciones indebidas o para auditorias.		
Criterios de aceptación			
• Se debe guardar en la base de datos las acciones que realicen los usuarios. • En la información guardada debe constar el UUID del usuario que realizó la acción, la acción que realizó, la fecha y la hora, el servicio donde la realizó y los datos que modifico o creo.			

Anexo 3. Diseño de la Arquitectura de Microservicios

Documento de Diseño de Arquitectura de Microservicios

Proyecto

Implementación de una arquitectura de microservicios y técnicas de DevOps a los servicios backend de la plataforma GeoWorker

Elaborado por:

Manuel Alejandro Aguinsaca Medina

Mayo de 2024

1. Introducción

Este documento presenta una vista general del proceso que se llevó a cabo para diseñar la arquitectura de microservicios de la plataforma “GeoWorker”, a través del patrón Domain-Driven Design (DDD) se obtuvo una serie de diagramas con el objetivo de ir encapsulando la lógica de negocio y así poder ir agrupando e identificando los microservicios a ser desarrollados.

2. Propósito

El propósito de este documento de es comprender las fases o el proceso llevado a cabo para identificar los microservicios de la plataforma “GeoWorker” con el patrón de diseño Domain-Driven Design.

3. Alcance

El presente documento detalla los diagramas utilizados para la identificación de los microservicios de la plataforma “GeoWorker” por medio del patrón DDD se partió del diagrama de clases y como resultado se obtuvo el diagrama de Contextos Delimitados (Bounded Context).

4. Referencias

Referencia	Título del Documento
Patrón	Domain-Driven Design
Requisitos	Especificación de requerimientos
Lenguaje de modelado	- UML - UML 2.5

5. Desarrollo

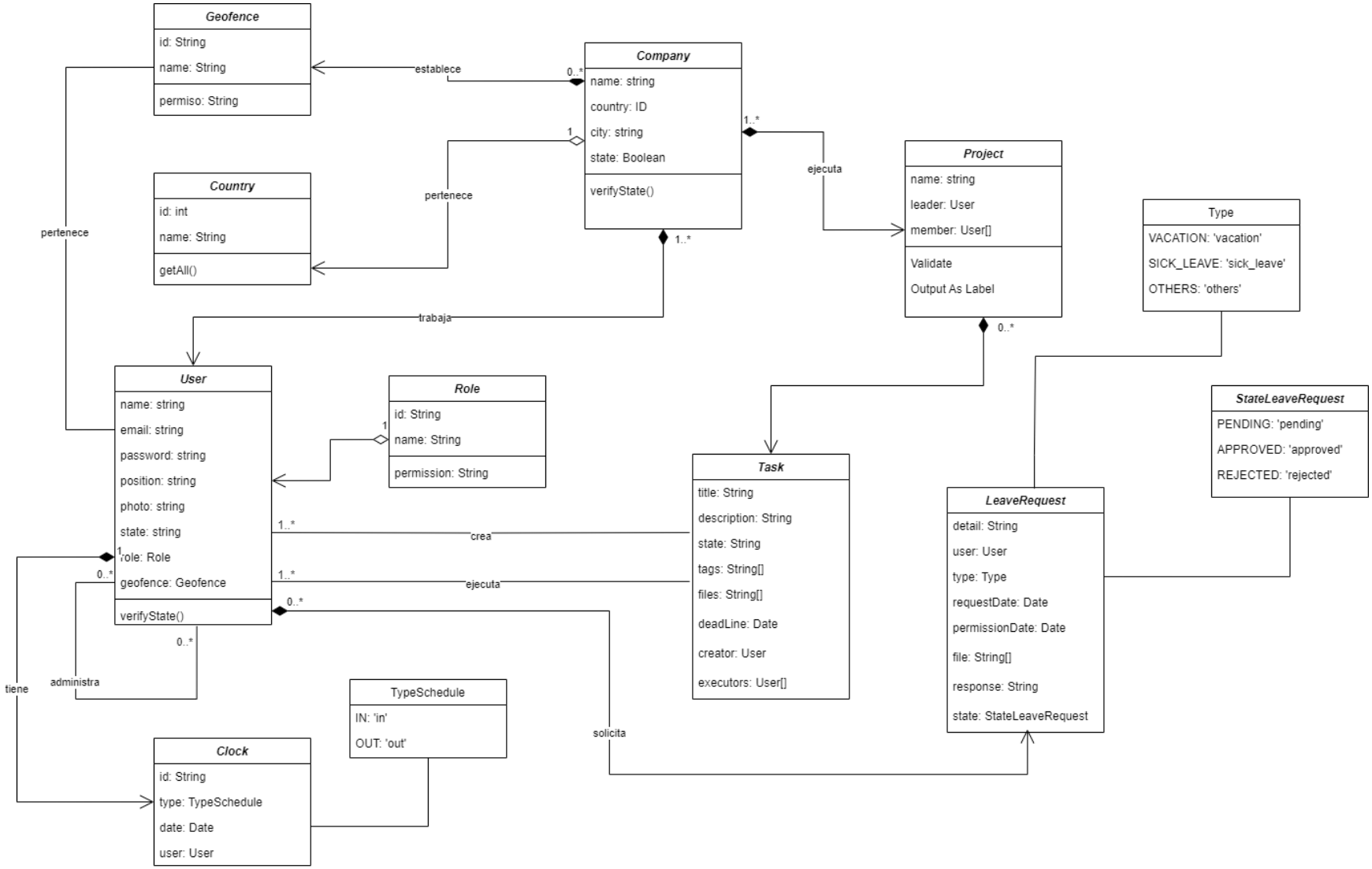
A continuación, de detallan las fases que se implementaron para la identificación de los microservicios de la plataforma.

5.1. Fase 1

En esta fase, se obtuvo el diagrama de clases de la plataforma a partir de los requisitos obtenidos previamente. Dentro de las entidades principales identificadas en los requisitos tenemos:

- Empresa
- Usuario
- Proyecto
- Tarea
- Solicitud
- Horario
- Geocerca

A partir de estas entidades, se agregó los atributos y las relaciones entre ellas, además, se agregó otras entidades como: País para el almacenamiento de la información de países disponibles, así como también Roles, que permitirá almacenar los roles y sus permisos.

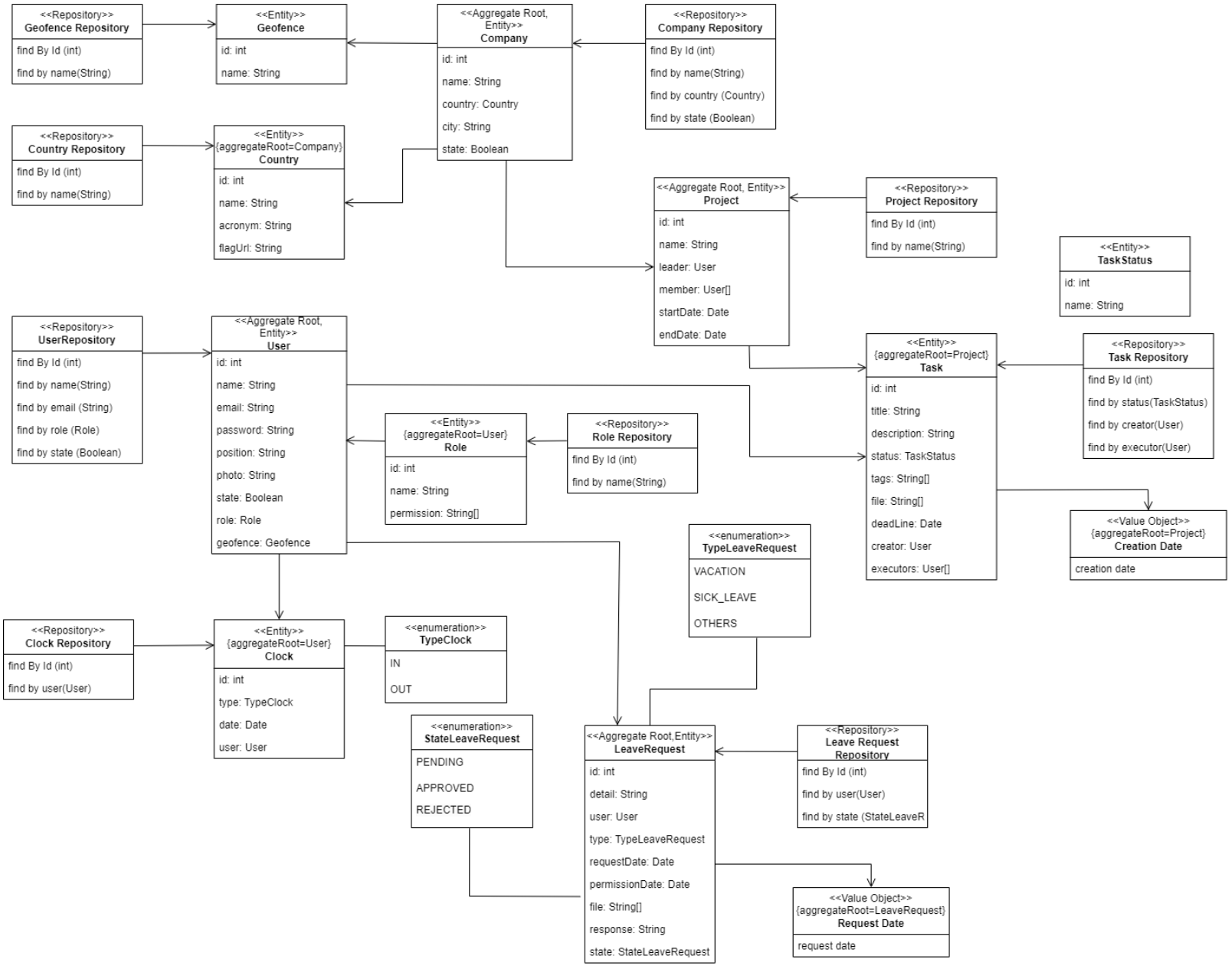

Geofence
id: String
name: String
permiso: String
establece
0..*
Company
name: string
country: ID
city: string
state: Boolean
verifyState()
1..*
ejecuta
Project
name: string
leader: User
member: User[]
Validate
Output As Label
Country
id: int
name: String
getAll()
pertenece
1
pertenece
1..*
trabaja
0..*
Type
VACATION: 'vacation'
SICK_LEAVE: 'sick_leave'
OTHERS: 'others'
User
name: string
email: string
password: string
position: string
photo: string
state: string
role: Role
geofence: Geofence
verifyState()
Role
id: String
name: String
permission: String
1
1..*
crea
Task
title: String
description: String
state: String
tags: String[]
files: String[]
deadLine: Date
creator: User
executors: User[]
StateLeaveRequest
PENDING: 'pending'
APPROVED: 'approved'
REJECTED: 'rejected'
LeaveRequest
detail: String
user: User
type: Type
requestDate: Date
permissionDate: Date
file: String[]
response: String
state: StateLeaveRequest
1
0..*
1..*
ejecuta
0..*
0..*
administra
tiene
TypeSchedule
IN: 'in'
OUT: 'out'
solicita
Clock
id: String
type: TypeSchedule
date: Date
user: User

5.2. Fase 2

La Fase 2 consiste en enriquecer el diagrama de clases con conceptos propios de DDD y ayudándose de UML 2.5.

Los conceptos aplicados en esta fase son los siguientes:

- **Aggregation Root:** Esta es una clase que encapsula a varias clases como un conjunto o un todo. Un ejemplo de este concepto es la entidad Empresa, que encapsula a la entidad País, otro ejemplo es la entidad Usuario que encapsula a la entidad Rol.
- **Repository:** Los Repositorios son clases que sirven como conexión sobre la capa de persistencia de datos, proporcionando a los desarrolladores independencia. En el diagrama propuesto, cada clase principal tiene un repositorio, con el fin de garantizar su persistencia independiente y accesos a métodos que permitan manipular los datos de una entidad.
- **Entity:** Las entidades son objetos de dominio, su estado puede variar, pero persiste con el tiempo. Por estas razones, a cada clase principal de la plataforma se le ha convertido en una entidad.
- **Value Object:** Este concepto se aplica a datos que no cambian su valor una vez son creados, y si su estado cambia, se crea otra instancia de la misma. Dentro del diagrama la fecha de creación de una tarea se la considera como un Value Object, ya que el valor de este campo no va a variar.

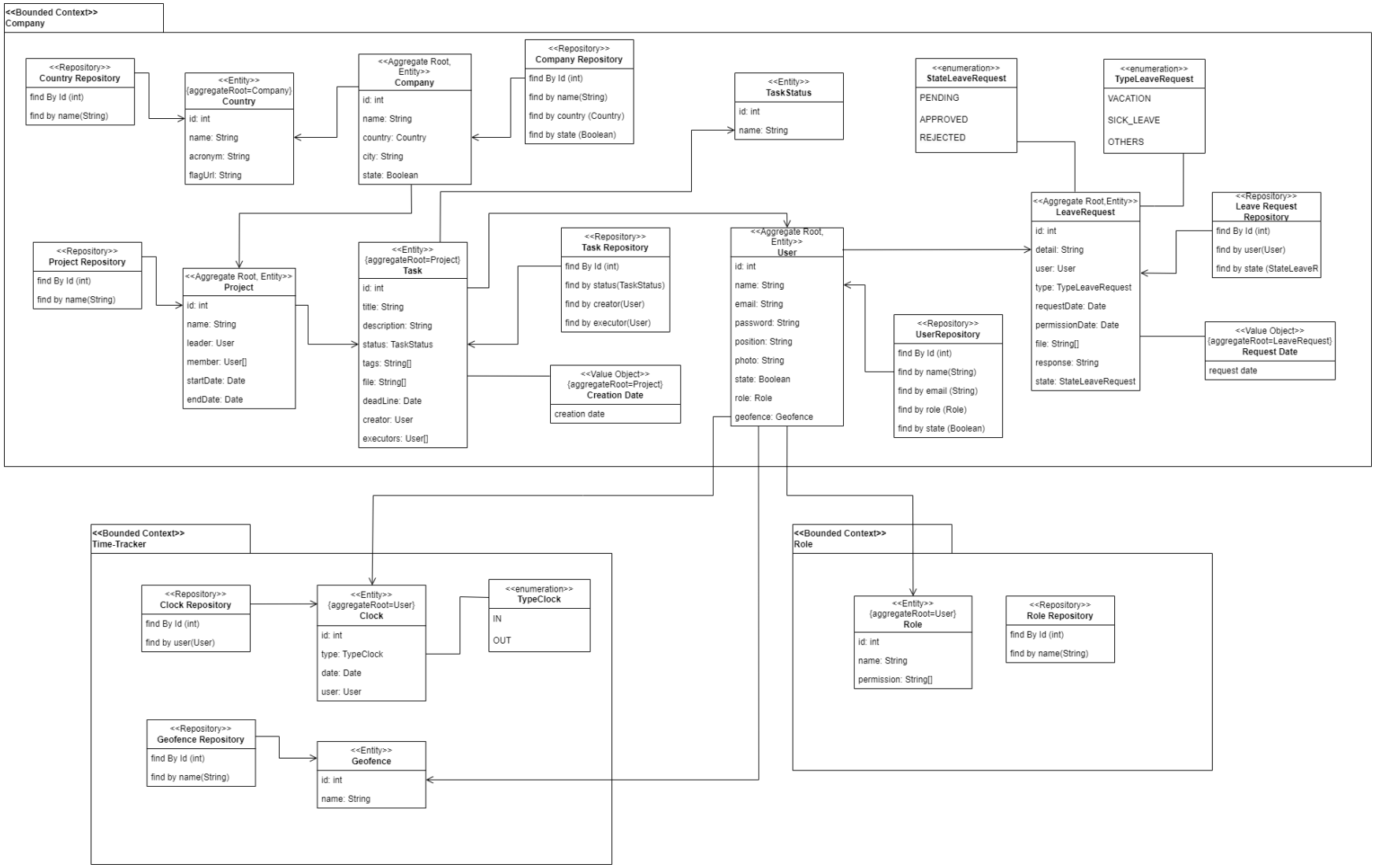

<<Repository>>
Geofence Repository
find By Id (int)
find by name(String)
<<Entity>>
Geofence
id: int
name: String
<<Aggregate Root, Entity>>
Company
id: int
name: String
country: Country
city: String
state: Boolean
<<Repository>>
Company Repository
find By Id (int)
find by name(String)
find by country (Country)
find by state (Boolean)
<<Repository>>
Country Repository
find By Id (int)
find by name(String)
<<Entity>>
{aggregateRoot=Company}
Country
id: int
name: String
acronym: String
flagUrl: String
<<Aggregate Root, Entity>>
Project
id: int
name: String
leader: User
member: User[]
startDate: Date
endDate: Date
<<Repository>>
Project Repository
find By Id (int)
find by name(String)
<<Entity>>
TaskStatus
id: int
name: String
<<Repository>>
UserRepository
find By Id (int)
find by name(String)
find by email (String)
find by role (Role)
find by state (Boolean)
<<Aggregate Root, Entity>>
User
id: int
name: String
email: String
password: String
position: String
photo: String
state: Boolean
role: Role
geofence: Geofence
<<Entity>>
{aggregateRoot=Project}
Task
id: int
title: String
description: String
status: TaskStatus
tags: String[]
file: String[]
deadLine: Date
creator: User
executors: User[]
<<Repository>>
Task Repository
find By Id (int)
find by status(TaskStatus)
find by creator(User)
find by executor(User)
<<Entity>>
{aggregateRoot=User}
Role
id: int
name: String
permission: String[]
<<Repository>>
Role Repository
find By Id (int)
find by name(String)
<<Value Object>>
{aggregateRoot=Project}
Creation Date
creation date
<<enumeration>>
TypeLeaveRequest
VACATION
SICK_LEAVE
OTHERS
<<Repository>>
Clock Repository
find By Id (int)
find by user(User)
<<Entity>>
{aggregateRoot=User}
Clock
id: int
type: TypeClock
date: Date
user: User
<<enumeration>>
TypeClock
IN
OUT
<<enumeration>>
StateLeaveRequest
PENDING
APPROVED
REJECTED
<<Aggregate Root,Entity>>
LeaveRequest
id: int
detail: String
user: User
type: TypeLeaveRequest
requestDate: Date
permissionDate: Date
file: String[]
response: String
state: StateLeaveRequest
<<Repository>>
Leave Request Repository
find By Id (int)
find by user(User)
find by state (StateLeaveR
<<Value Object>>
{aggregateRoot=LeaveRequest}
Request Date
request date

5.3. Fase 3

Esta fase es la última de todo el proceso, consiste en generar un diagrama de Contextos Delimitados (Bounded Context), es decir, se agrupan funcionalidades relacionadas en microservicios aislados e independientes.

La agrupación que se realizó en este diagrama consiste en:

- **Company Service:** Servicio encargado de administrar las entidades de Empresa, Usuarios, Proyecto, Tareas y Solicitudes. Se agruparon de esta forma, ya que las entidades antes mencionadas comparten una relación directa entre ellas, además de requerir consultas avanzadas.
- **Role Service:** Este contexto contiene la gestión de roles y permisos de la plataforma, con esta agrupación se garantiza que los permisos y sus roles puedan escalar con la plataforma.
- **Time Tracker:** En este contexto están contempladas las funcionalidades que conciernen a la administración del horario de un usuario, es decir, el control de entradas y salidas de su jornada laboral y el control de las geocercas.
- **Auth Service:** Aquí se contempla toda la funcionalidad de administrar la sesión de los usuarios.
- **Audit Service:** Este contexto encapsula la funcionalidad de registrar los logs de auditoría del sistema.


<<Bounded Context>>
Company
<<Repository>>
Country Repository
find By Id (int)
find by name(String)
<<Entity>>
{aggregateRoot=Company}
Country
id: int
name: String
acronym: String
flagUrl: String
<<Aggregate Root, Entity>>
Company
id: int
name: String
country: Country
city: String
state: Boolean
<<Repository>>
Company Repository
find By Id (int)
find by name(String)
find by country (Country)
find by state (Boolean)
<<Entity>>
TaskStatus
id: int
name: String
<<enumeration>>
StateLeaveRequest
PENDING
APPROVED
REJECTED
<<enumeration>>
TypeLeaveRequest
VACATION
SICK_LEAVE
OTHERS
<<Repository>>
Project Repository
find By Id (int)
find by name(String)
<<Aggregate Root, Entity>>
Project
id: int
name: String
leader: User
member: User[]
startDate: Date
endDate: Date
<<Entity>>
{aggregateRoot=Project}
Task
id: int
title: String
description: String
status: TaskStatus
tags: String[]
file: String[]
deadLine: Date
creator: User
executors: User[]
<<Repository>>
Task Repository
find By Id (int)
find by status(TaskStatus)
find by creator(User)
find by executor(User)
<<Value Object>>
{aggregateRoot=Project}
Creation Date
creation date
<<Aggregate Root, Entity>>
User
id: int
name: String
email: String
password: String
position: String
photo: String
state: Boolean
role: Role
geofence: Geofence
<<Repository>>
UserRepository
find By Id (int)
find by name(String)
find by email (String)
find by role (Role)
find by state (Boolean)
<<Aggregate Root,Entity>>
LeaveRequest
id: int
detail: String
user: User
type: TypeLeaveRequest
requestDate: Date
permissionDate: Date
file: String[]
response: String
state: StateLeaveRequest
<<Repository>>
Leave Request Repository
find By Id (int)
find by user(User)
find by state (StateLeaveR
<<Value Object>>
{aggregateRoot=LeaveRequest}
Request Date
request date
<<Bounded Context>>
Time-Tracker
<<Repository>>
Clock Repository
find By Id (int)
find by user(User)
<<Entity>>
{aggregateRoot=User}
Clock
id: int
type: TypeClock
date: Date
user: User
<<enumeration>>
TypeClock
IN
OUT
<<Repository>>
Geofence Repository
find By Id (int)
find by name(String)
<<Entity>>
Geofence
id: int
name: String
<<Bounded Context>>
Role
<<Entity>>
{aggregateRoot=User}
Role
id: int
name: String
permission: String[]
<<Repository>>
Role Repository
find By Id (int)
find by name(String)

Anexo 4. Documento de Arquitectura de Software

Documento de Arquitectura de Software

Proyecto

Implementación de una arquitectura de microservicios y técnicas de DevOps a los servicios backend de la plataforma GeoWorker

Elaborado por:

Manuel Alejandro Aguinsaca Medina

Mayo de 2024

1. Introducción

Este documento presenta una perspectiva general de la arquitectura de los servicios backend de la plataforma “GeoWorker”, aquí de detallan la vista lógica, vista física, vista de escenarios, de procesos y, por último, la vista de despliegue. El objetivo de este documento es el obtener una comprensión más clara de los procesos y flujos dentro de los servicios backend.

2. Propósito

El propósito de este documento de arquitectura de software es describir el diseño correspondiente según la arquitectura 4+1. El principal objetivo que las funcionalidades a desarrollar en el proyecto se adapten a los requerimientos del cliente

3. Alcance

En el presente documento contiene la arquitectura de software de los servicios backend de la plataforma “GeoWorker”, con la ayuda de las vistas del modelo 4+1, el mismo que está conformado por las vistas de despliegue, procesos, escenarios, y físicas.

4. Referencias

Referencia	Título del Documento
Requisitos	Especificación de requerimientos
Modelo	Arquitectura de Software 4+1[42]

5. Vista Global

Este documento presenta una vista organizada de la arquitectura utilizada para el desarrollo de los servicios backend de la plataforma “GeoWorker”, haciendo uso del modelo 4 + 1 que proporciona las siguientes vistas:

- **Vista Lógica:** Se centra en describir las funciones y la estructura de la plataforma “GeoWorker”
- **Vista de Despliegue:** Utiliza los diagramas de componentes o paquetes para representar como el sistema se encuentra segmentado, es decir sus componentes dependencias y gestión de software
- **Vista de Escenarios:** Se utiliza los diagramas de casos de uso.
- **Vista de Procesos:** Muestra de manera detallada la comunicación de los procesos.
- **Vista Física:** Se representan los componentes físicos del sistema y sus conexiones.

6. Representación de Arquitectura

Mediante el modelo arquitectónico 4 + 1 de Kruchten, se presenta una serie de vistas construidas con la ayuda del lenguaje de modelado unificado. A continuación, se detallan las 5 vistas utilizadas en este documento.

Vista	Elemento Modelado	Descripción
Vista de Escenarios	Casos de Uso	Muestra las interacciones de los diferentes usuarios con la plataforma
Vista Lógica	Diagrama de Clases	Representa los servicios y funciones que serán desarrolladas al usuario
Vista de Despliegue	Diagrama de Componentes	Representa los componentes de la plataforma, esto brindara una mejor comprensión al desarrollador
Vista de Procesos	Diagrama de secuencia o Actividad	Describe la comunicación entre los procesos de la plataforma
Vista Física	Diagrama de Despliegue	Presenta los componentes físicos del sistema

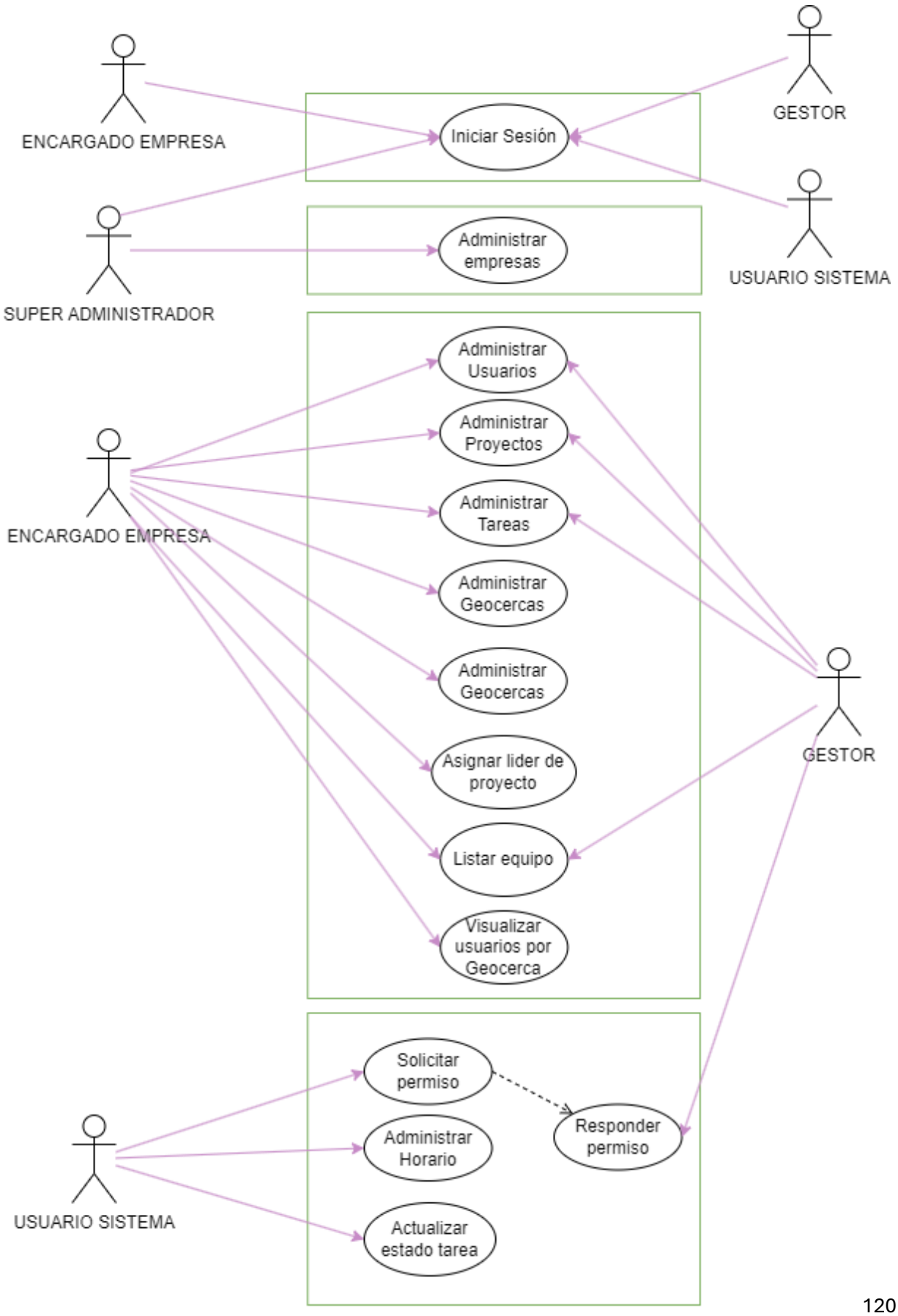
7. Objetivos de la Arquitectura

Los servicios backend de la plataforma “GeoWorker” deberán contar con lo siguiente:

- **Disponibilidad:** Lo servicios deberán estar disponibles las 24 horas del día, los 7 días de la semana, excepto los días que se esté actualizado o dando mantenimiento a los servicios.
- **Seguridad:** El acceso a los servicios backend y sus funcionalidades estarán limitadas por permisos y roles de los usuarios.
- **Rendimiento:** Los procesos dentro de los servicios y sus respuestas deberán ejecutarse de manera rápida y eficiente.
- **Usabilidad:** Los servicios deberán responder con los mensajes apropiados para guiar al usuario e informar de algún fallo ocurrido en el proceso.

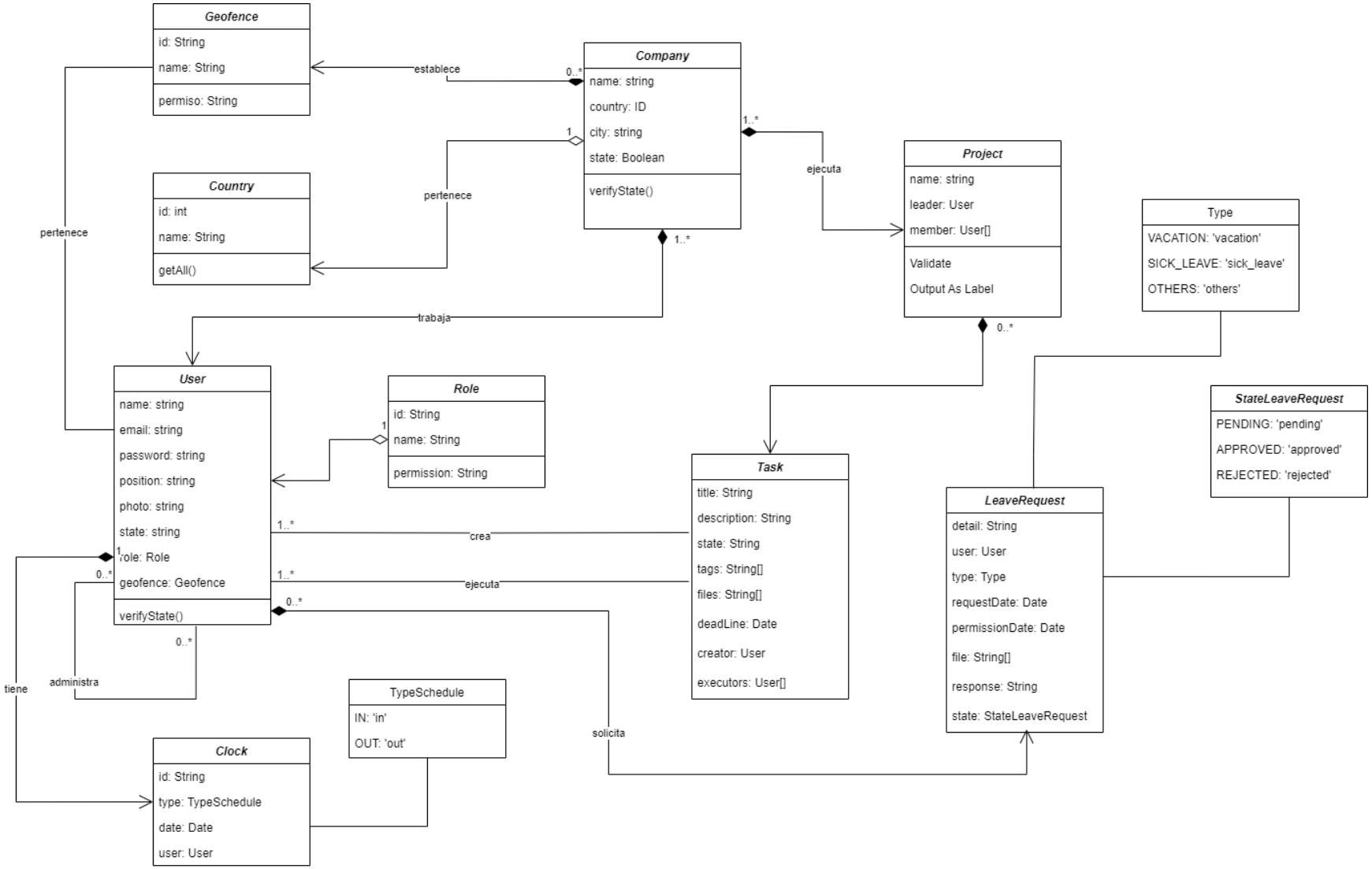
8. Vista de escenarios

Esta vista representa los escenarios con los que pueden interactuar los usuarios de la plataforma “GeoWorker”, es decir, las funcionalidades que el usuario administrador, encargado de empresa y usuario final pueden acceder.


ENCARGADO EMPRESA
GESTOR
Iniciar Sesión
SUPER ADMINISTRADOR
Administrar empresas
USUARIO SISTEMA
Administrar Usuarios
Administrar Proyectos
ENCARGADO EMPRESA
Administrar Tareas
Administrar Geocercas
Administrar Geocercas
Asignar lider de proyecto
GESTOR
Listar equipo
Visualizar usuarios por Geocerca
Solicitar permiso
Responder permiso
Administrar Horario
USUARIO SISTEMA
Actualizar estado tarea

9. Vista Lógica

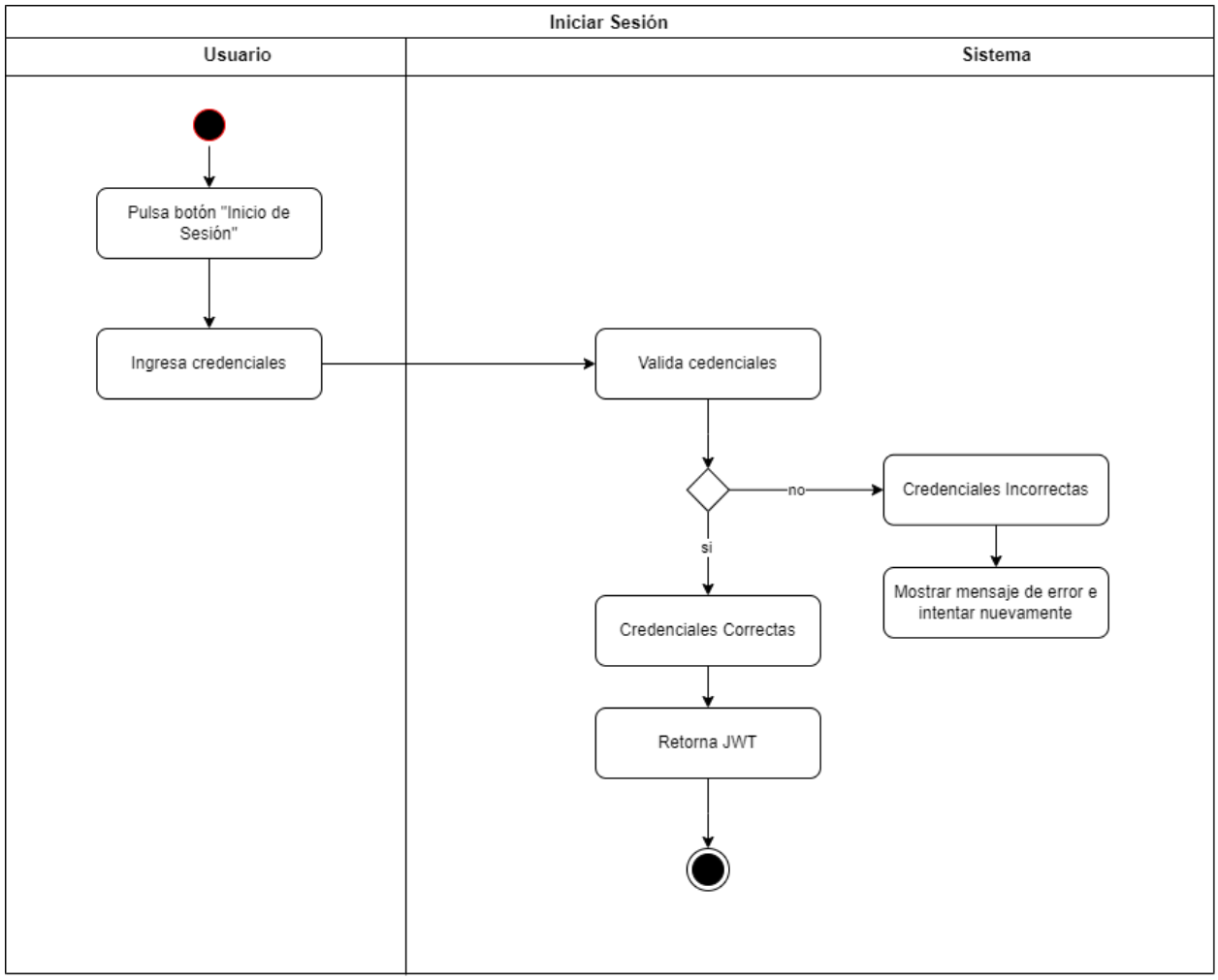
La vista lógica mediante el diagrama de clases permite representar las entidades con sus atributos, métodos, y relaciones de los servicios backend de la plataforma “GeoWorker”.


Geofence
id: String
name: String
permiso: String
establece
0..*
Company
name: string
country: ID
city: string
state: Boolean
verifyState()
1..*
1
ejecuta
Project
name: string
leader: User
member: User[]
Validate
Output As Label
Country
id: int
name: String
getAll()
pertenece
pertenece
1..*
trabaja
0..*
Type
VACATION: 'vacation'
SICK_LEAVE: 'sick_leave'
OTHERS: 'others'
User
name: string
email: string
password: string
position: string
photo: string
state: string
role: Role
geofence: Geofence
verifyState()
Role
id: String
name: String
permission: String
1
1..*
crea
Task
title: String
description: String
state: String
tags: String[]
files: String[]
deadLine: Date
creator: User
executors: User[]
StateLeaveRequest
PENDING: 'pending'
APPROVED: 'approved'
REJECTED: 'rejected'
LeaveRequest
detail: String
user: User
type: Type
requestDate: Date
permissionDate: Date
file: String[]
response: String
state: StateLeaveRequest
1..*
ejecuta
0..*
0..*
0..*
administra
tiene
TypeSchedule
IN: 'in'
OUT: 'out'
solicita
Clock
id: String
type: TypeSchedule
date: Date
user: User

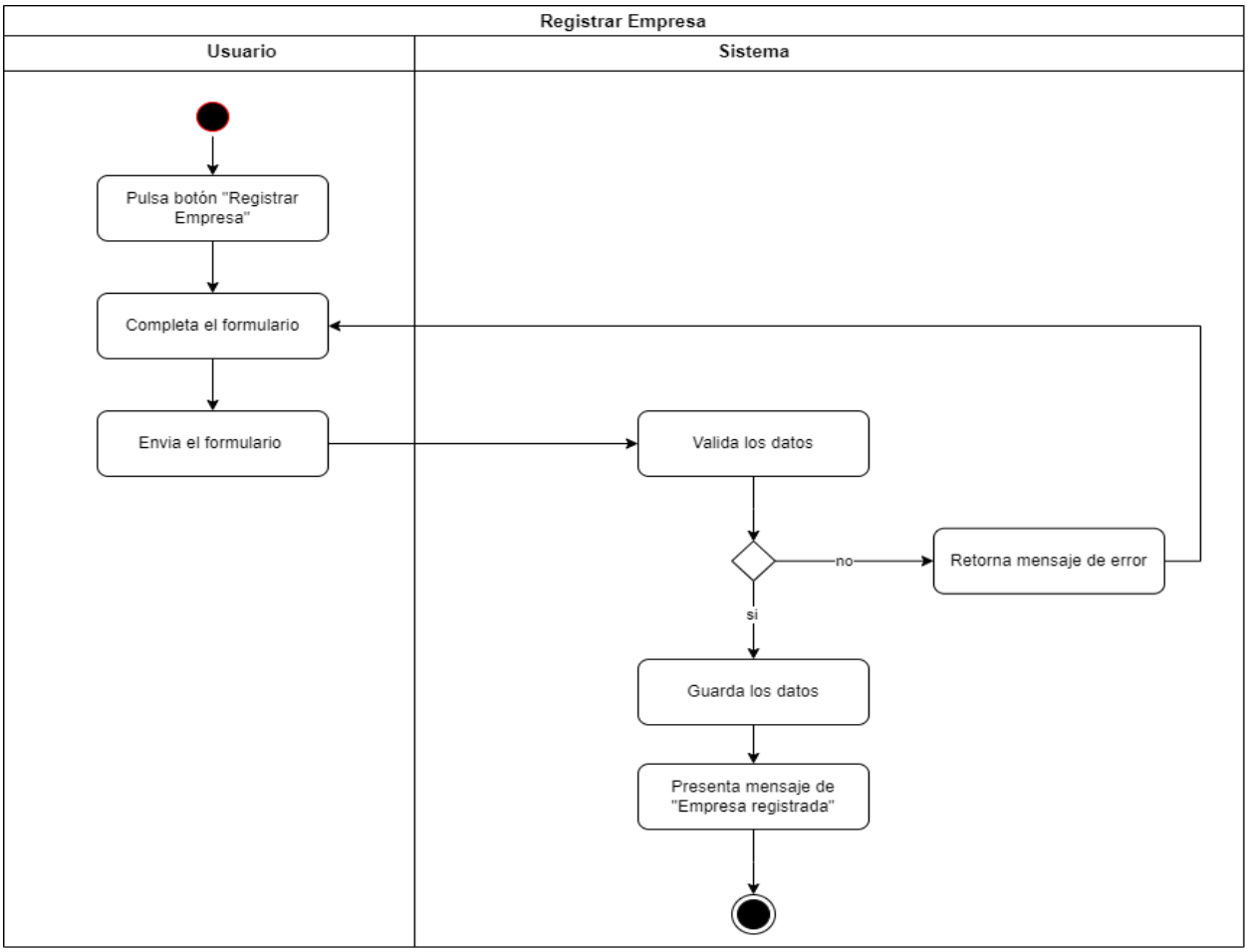
10. Vista de Procesos

En esta sección se presentan los diagramas de actividades, en los cuales de ilustran los procesos que los usuarios de la plataforma pueden realizar a través de los servicios backend.

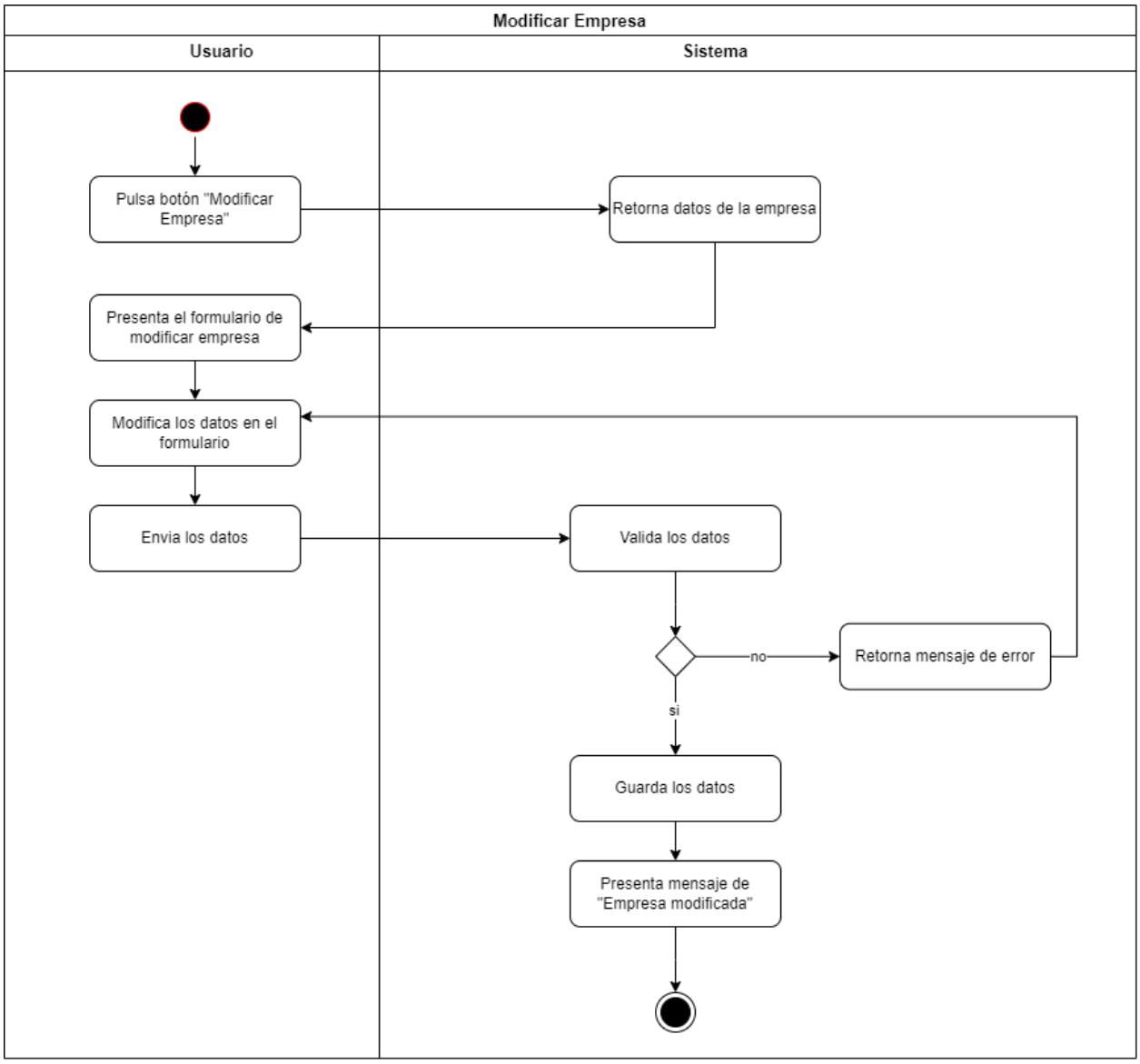
a. Diagrama de Actividades: Inicia Sesión



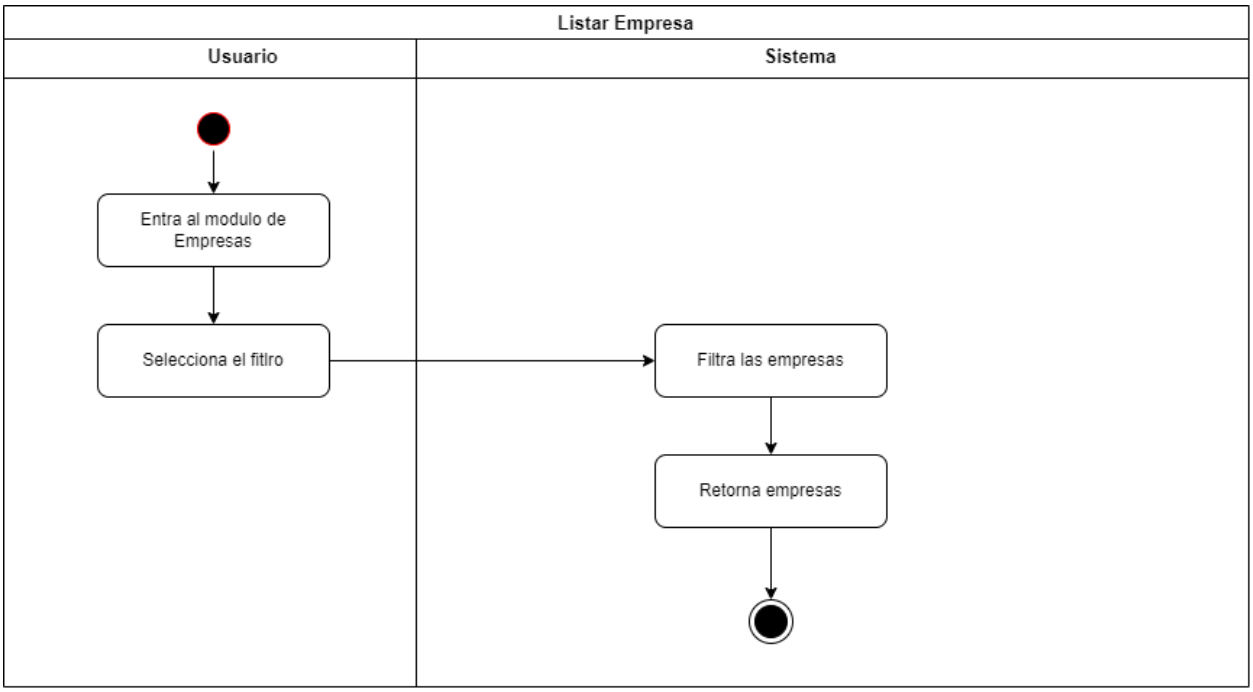
b. Diagrama de Actividades: Registrar Empresa



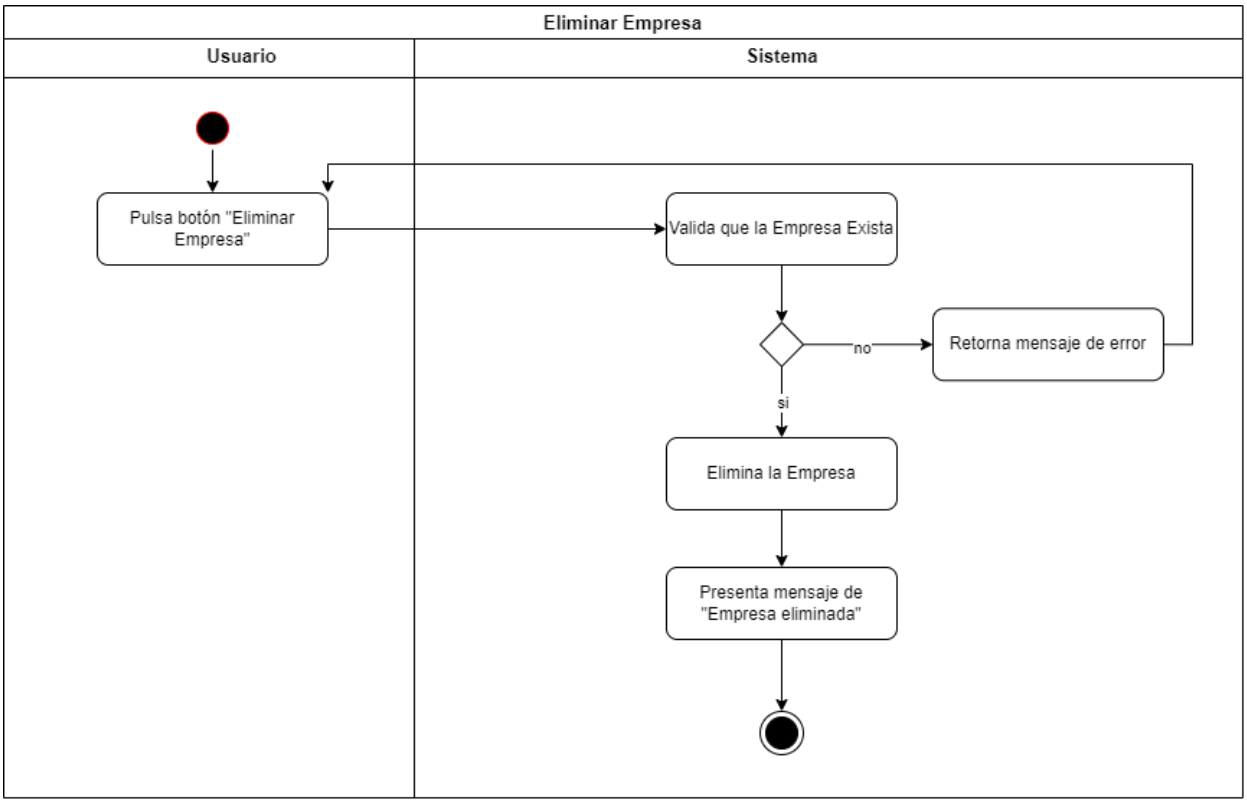
c. Diagrama de Actividades: Modificar Empresa



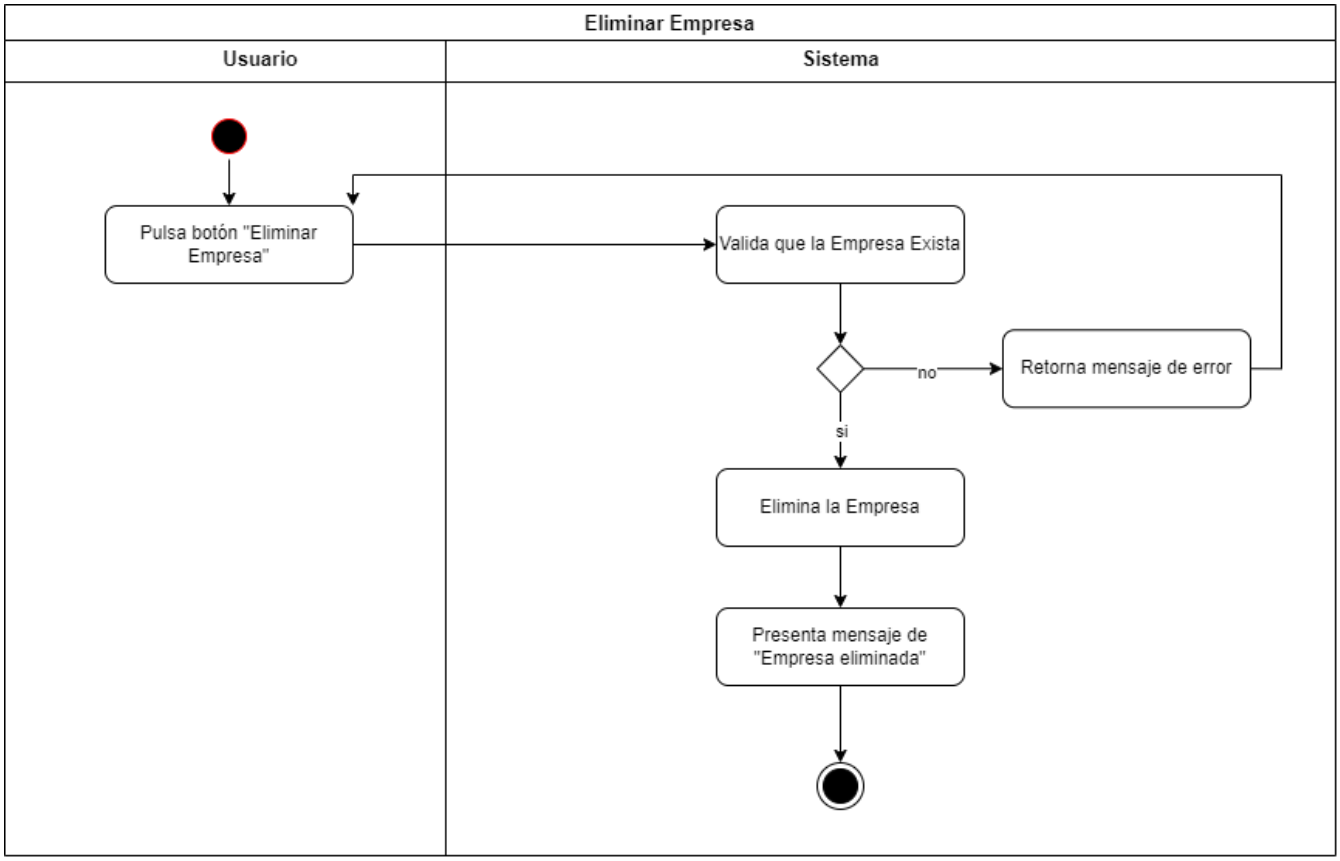
d. Diagrama de Actividades: Listar Empresa



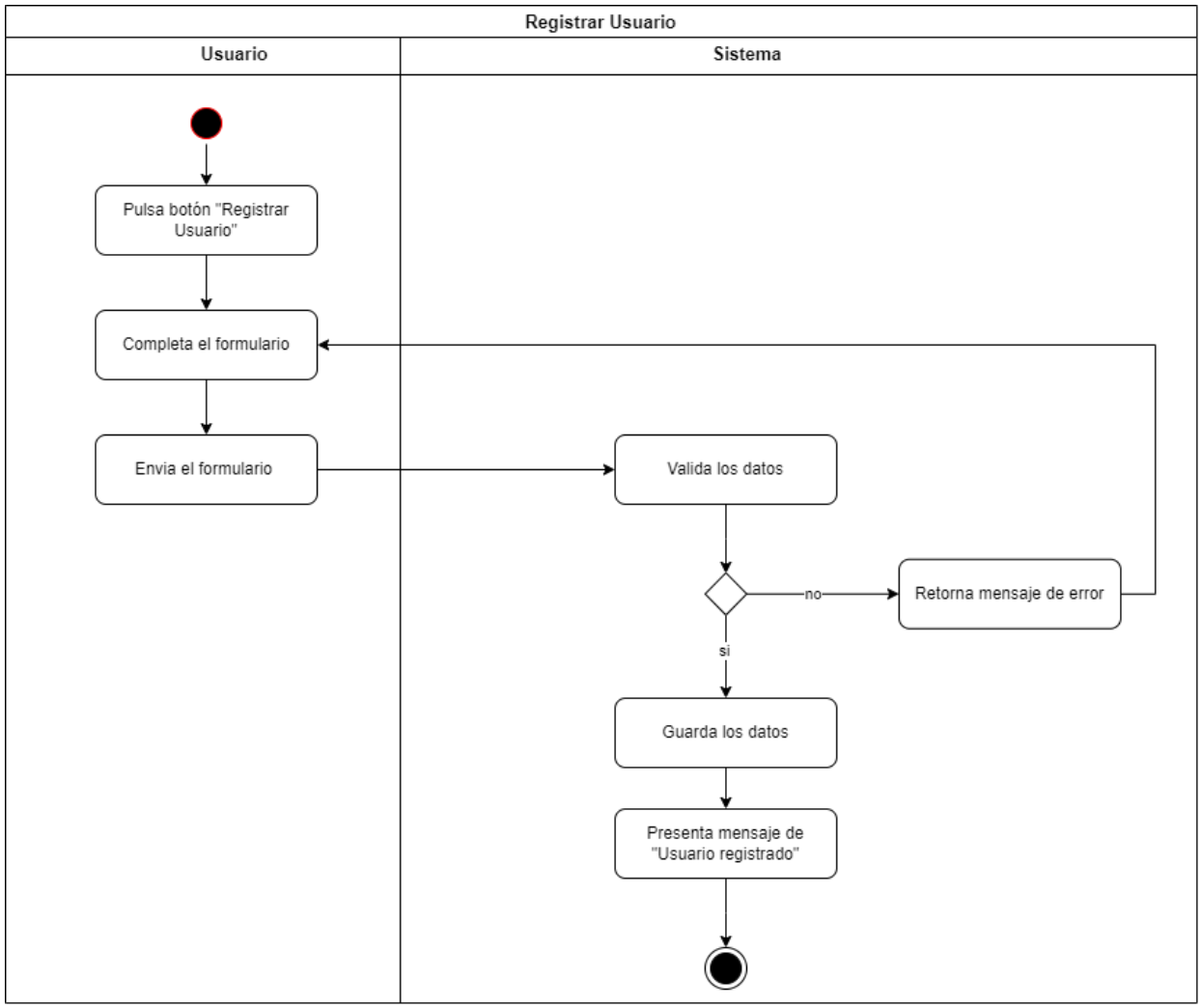
e. Diagrama de Actividades: Eliminar Empresa



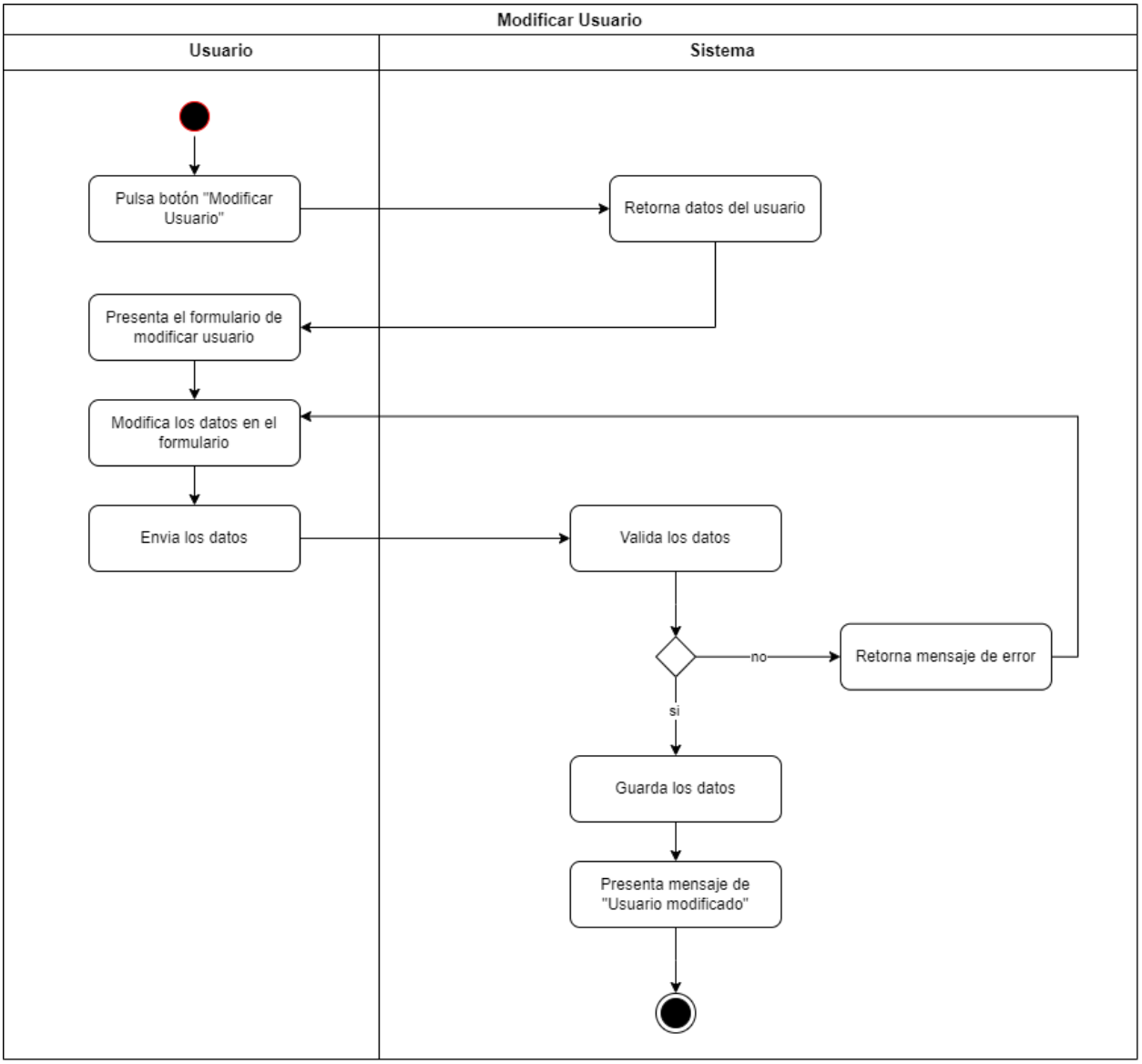
f. Diagrama de Actividades: Eliminar Empresa



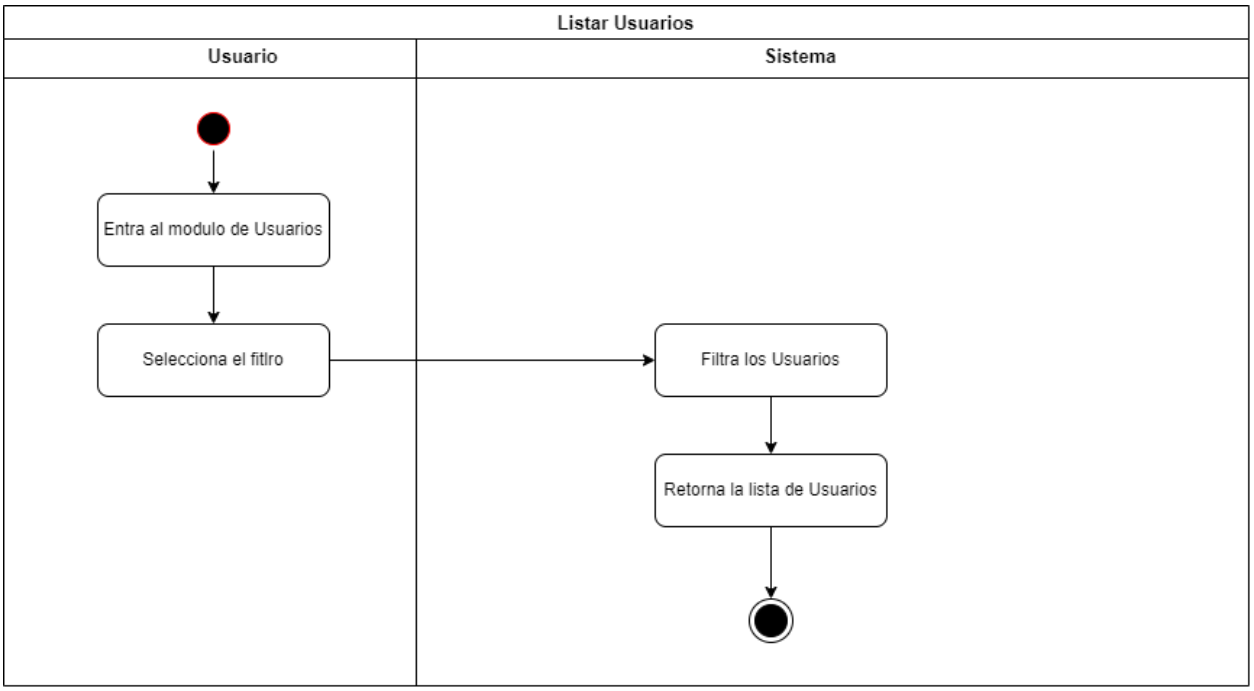
g. Diagrama de Actividades: Registrar Usuario



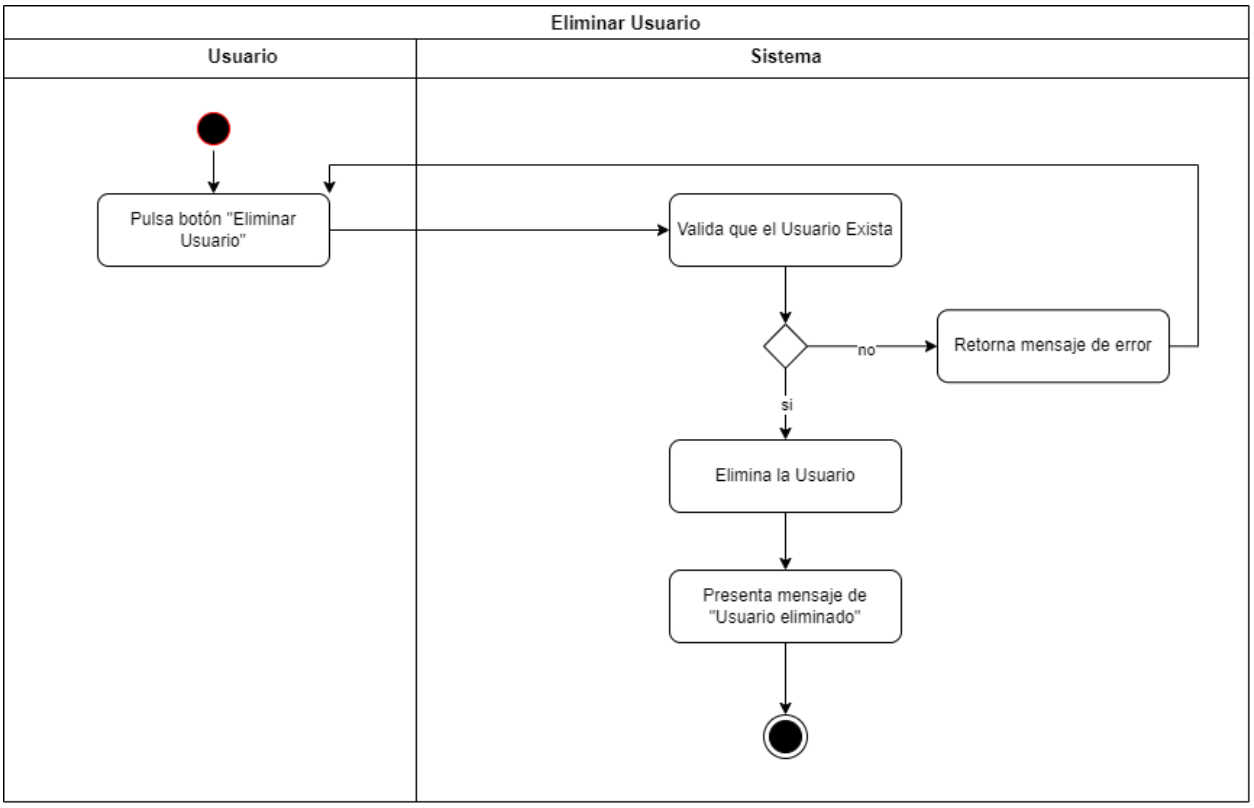
h. Diagrama de Actividades: Modificar Usuario



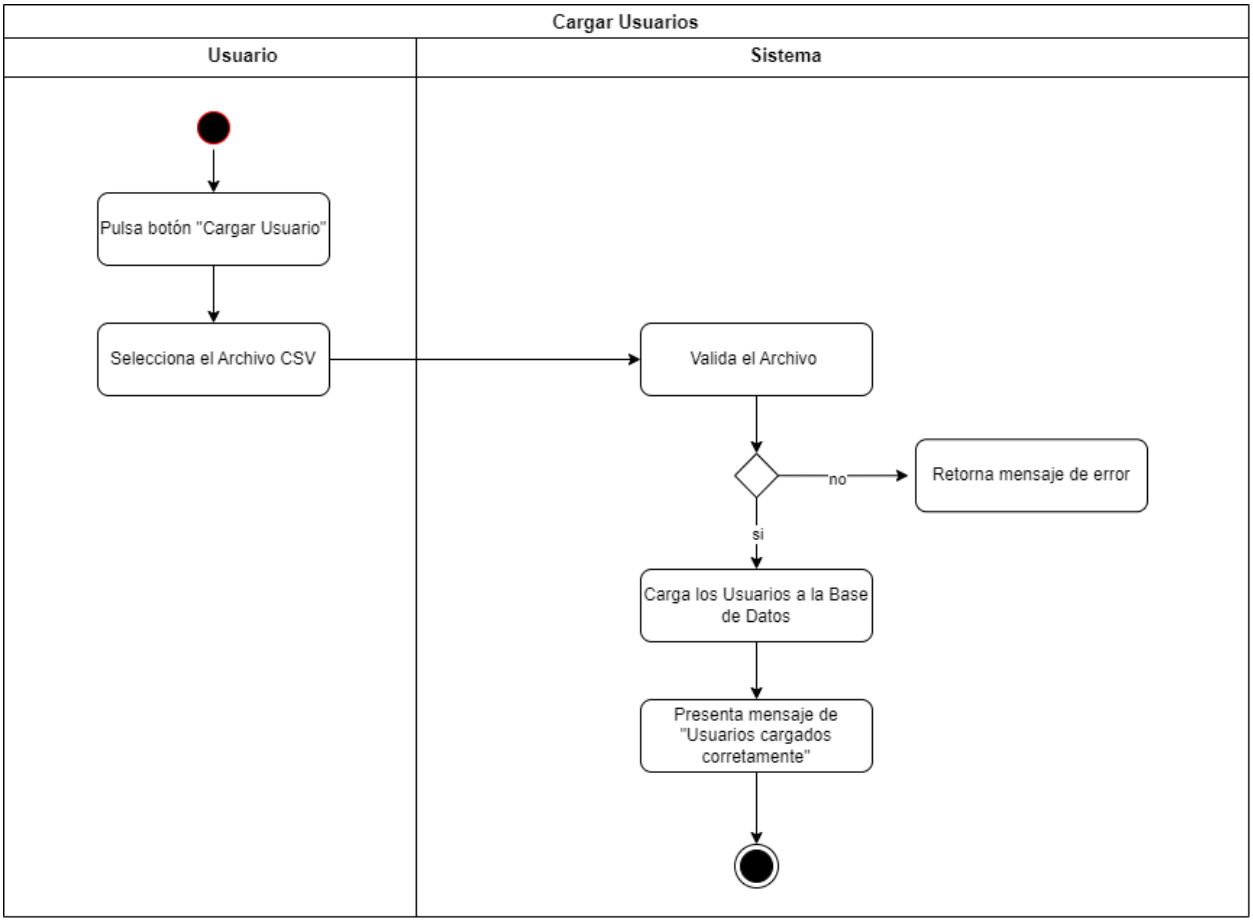
i. Diagrama de Actividades: Listar Usuarios



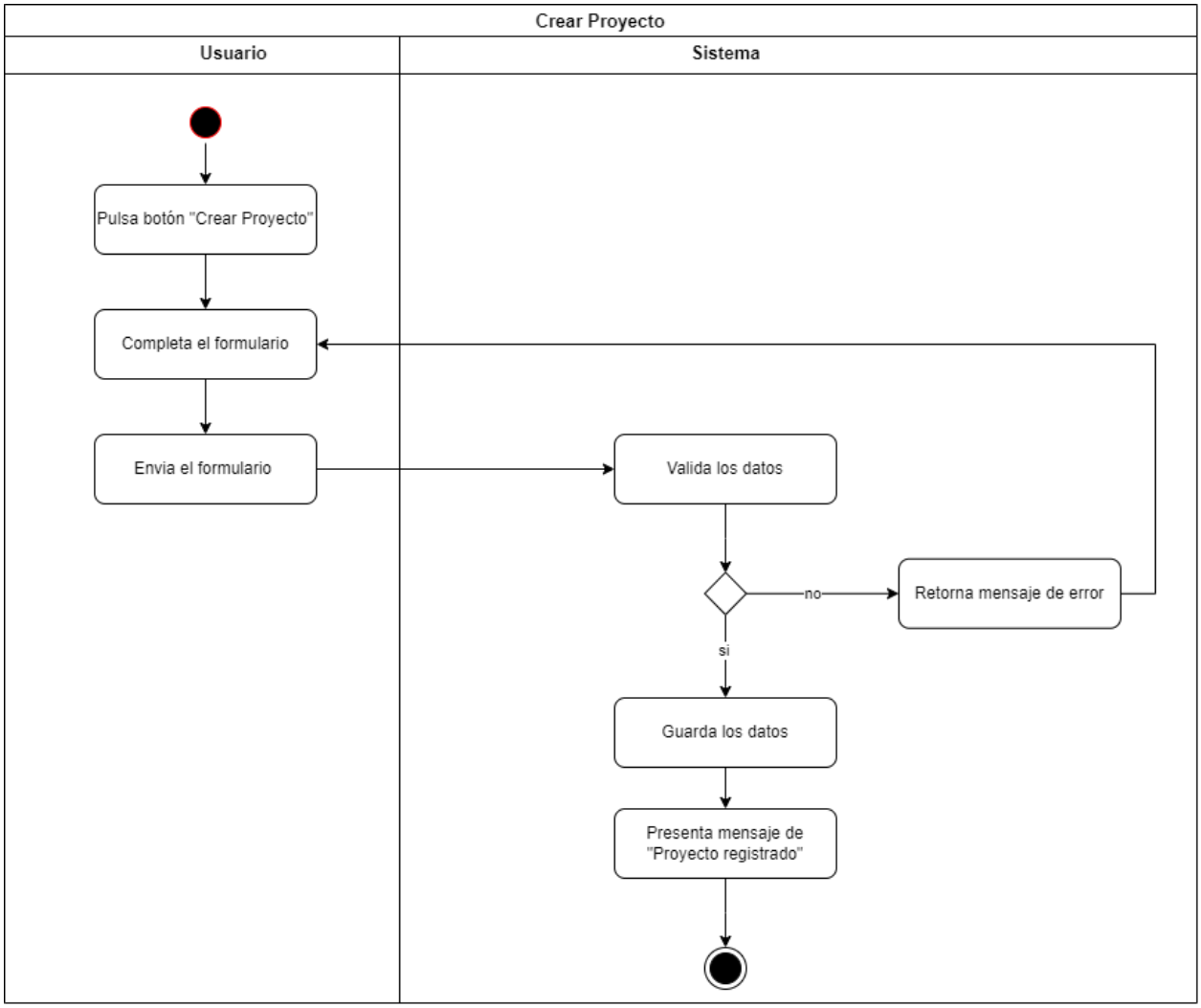
j. Diagrama de Actividades: Eliminar Usuario



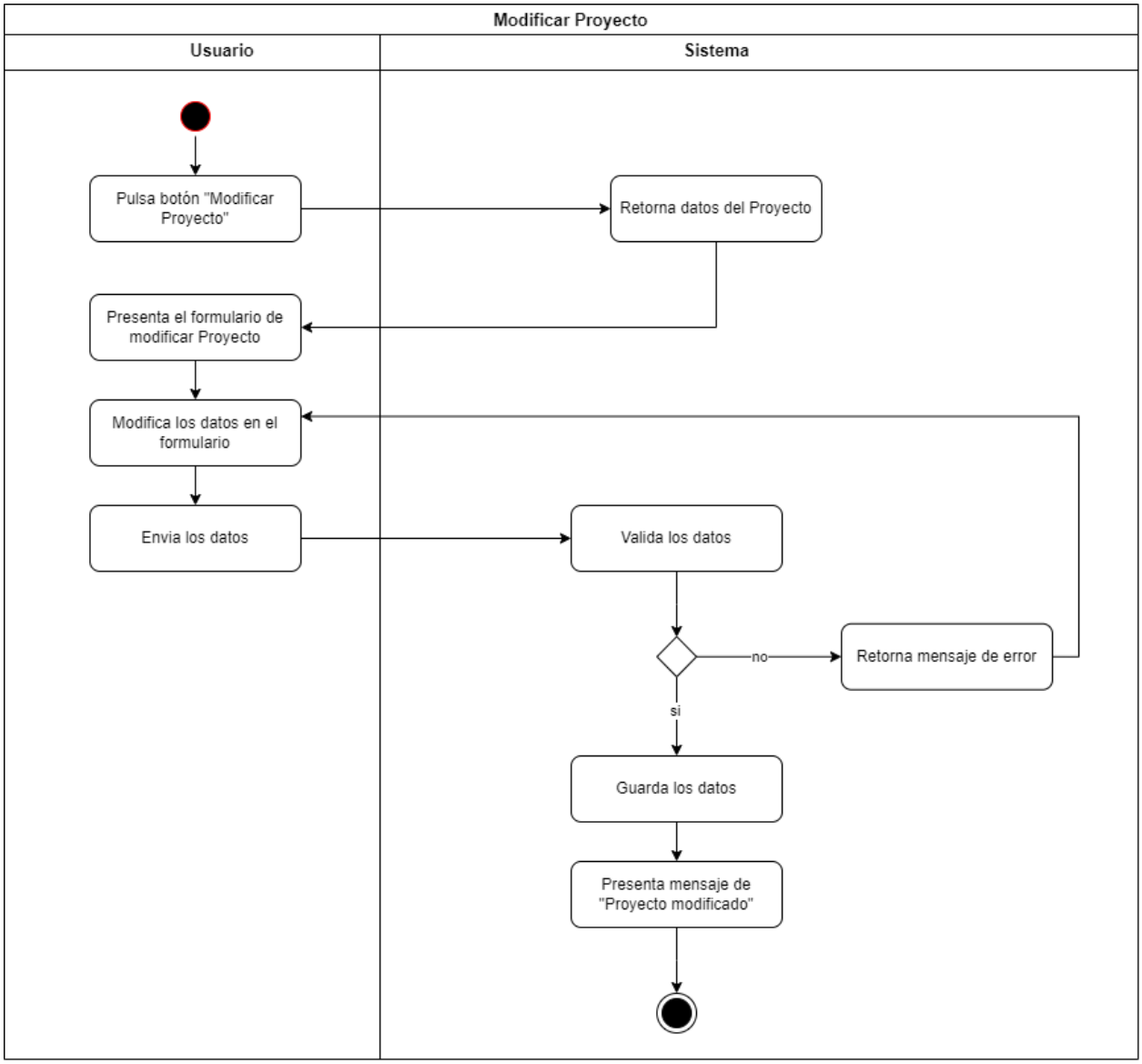
k. Diagrama de Actividades: Cargar Usuario



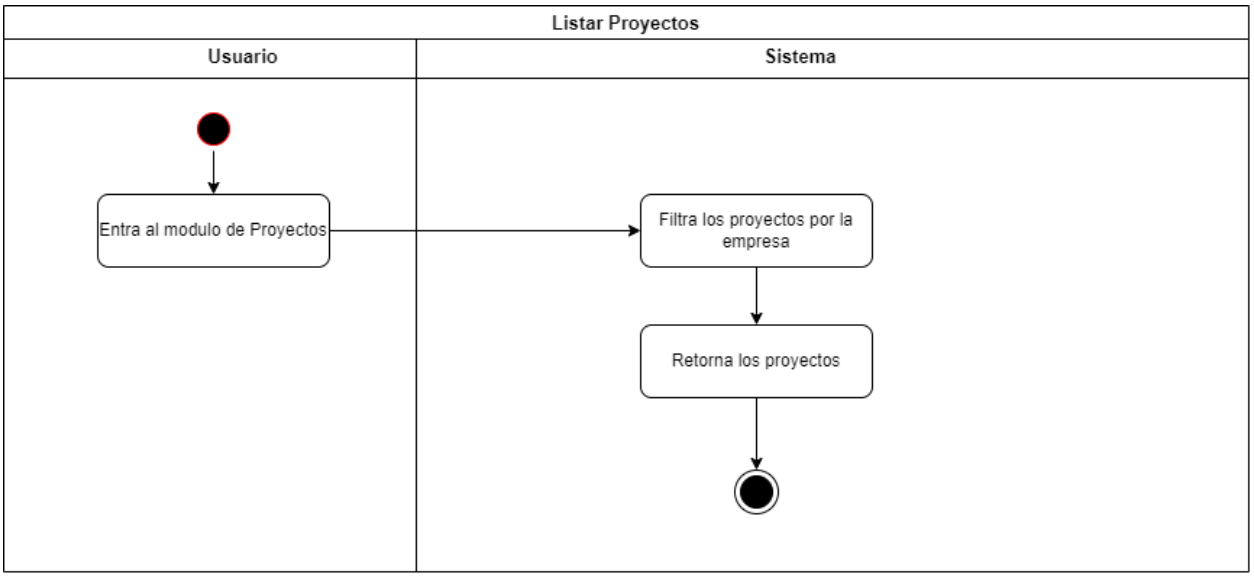
l. Diagrama de Actividades: Crear Proyecto



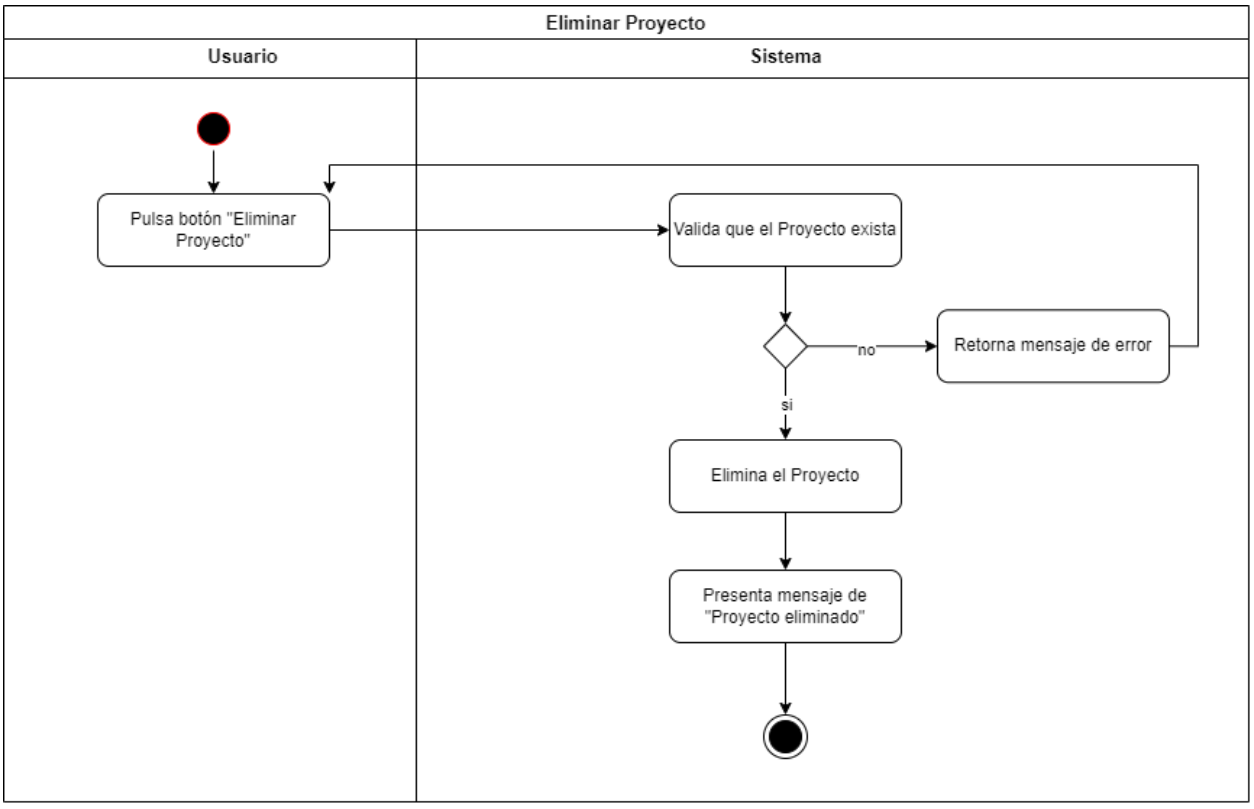
m. Diagrama de Actividades: Modificar Proyecto



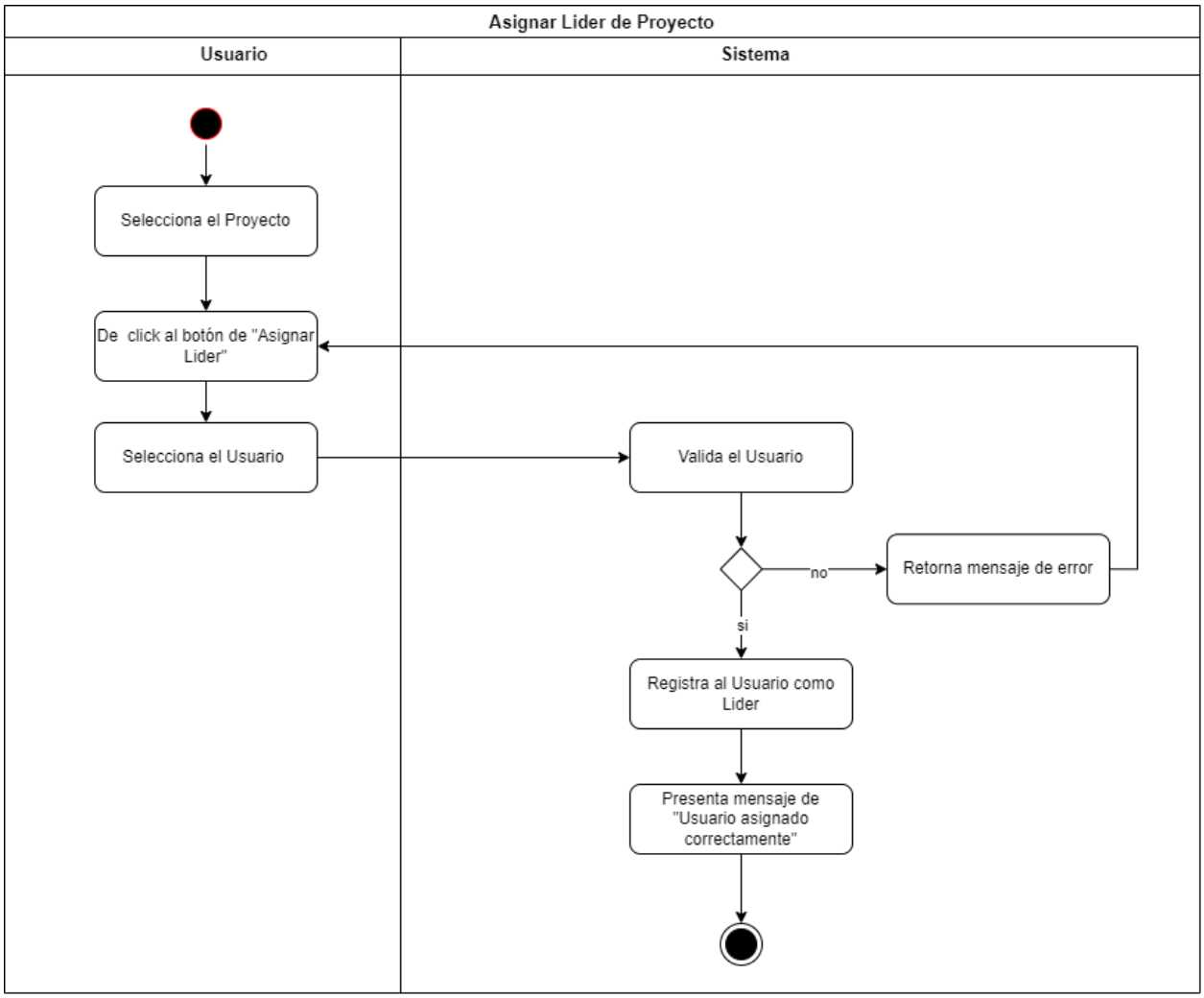
n. Diagrama de Actividades: Listar Proyecto



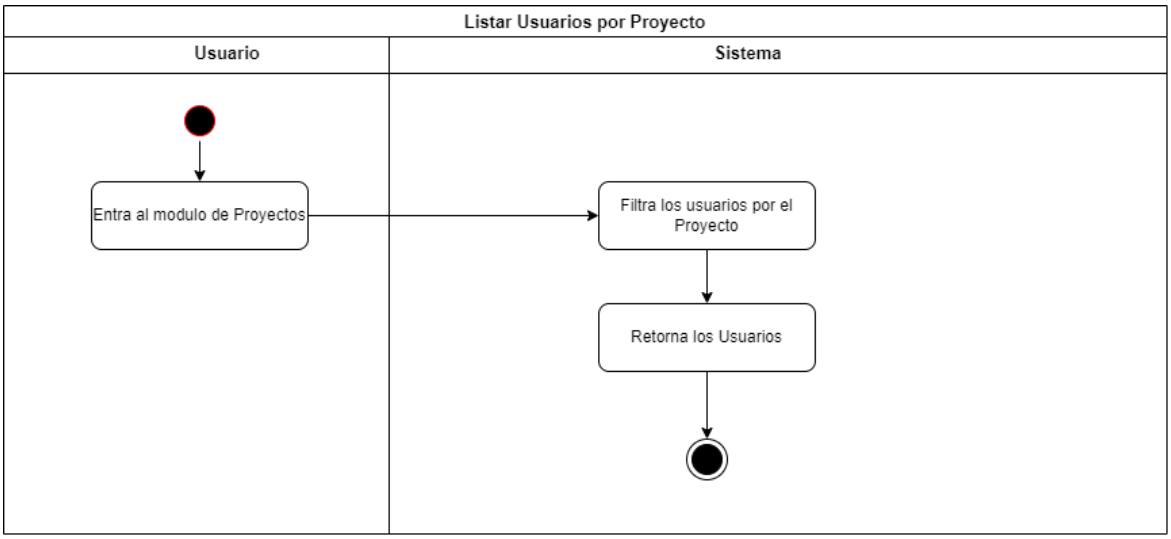
o. Diagrama de Actividades: Eliminar Proyecto



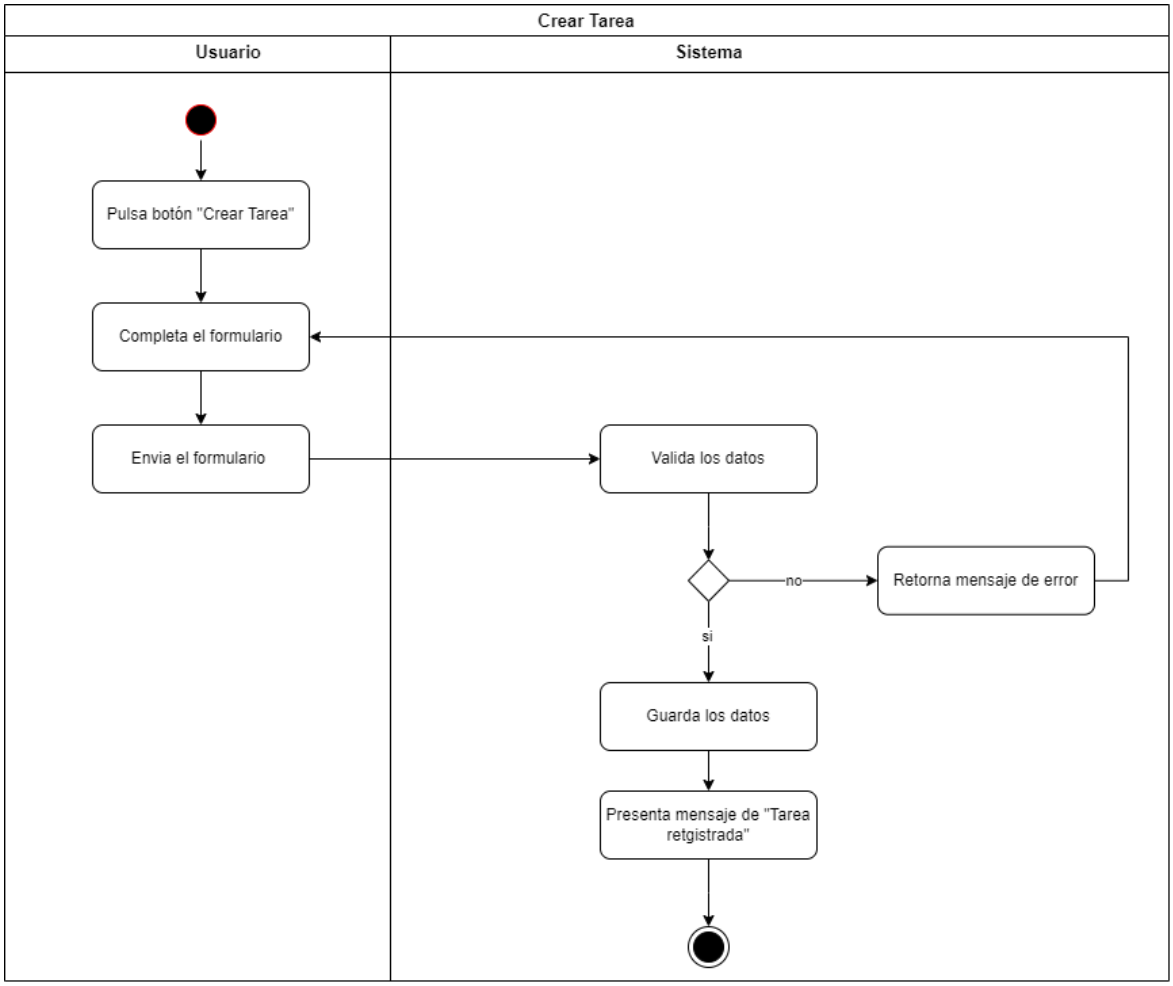
p. Diagrama de Actividades: Asignar Líder de Proyecto



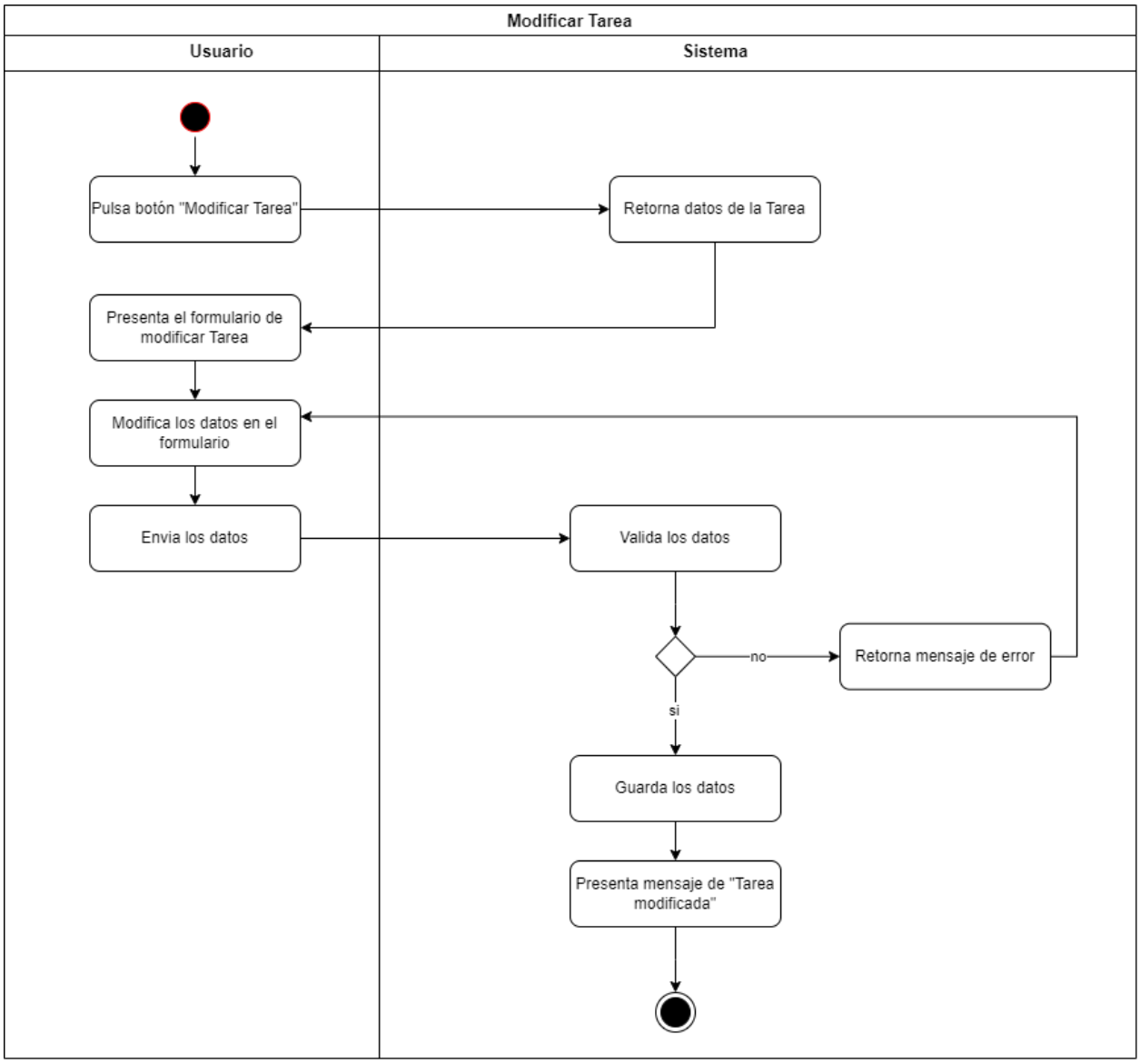
q. Diagrama de Actividades: Listar Usuarios por Proyecto



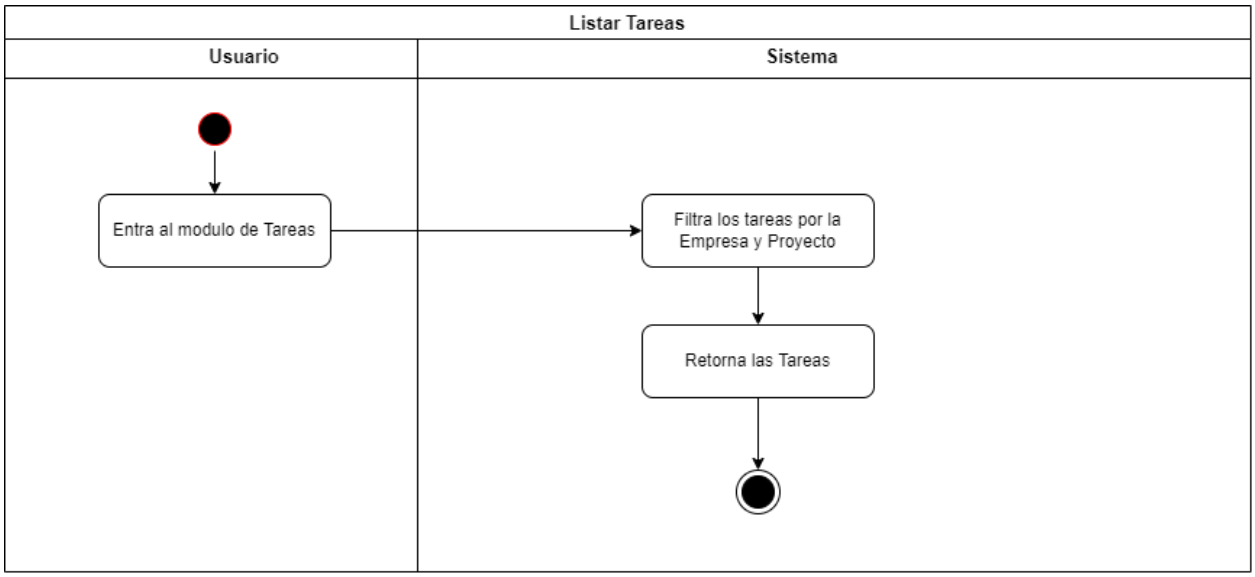
r. Diagrama de Actividades: Crear Tarea



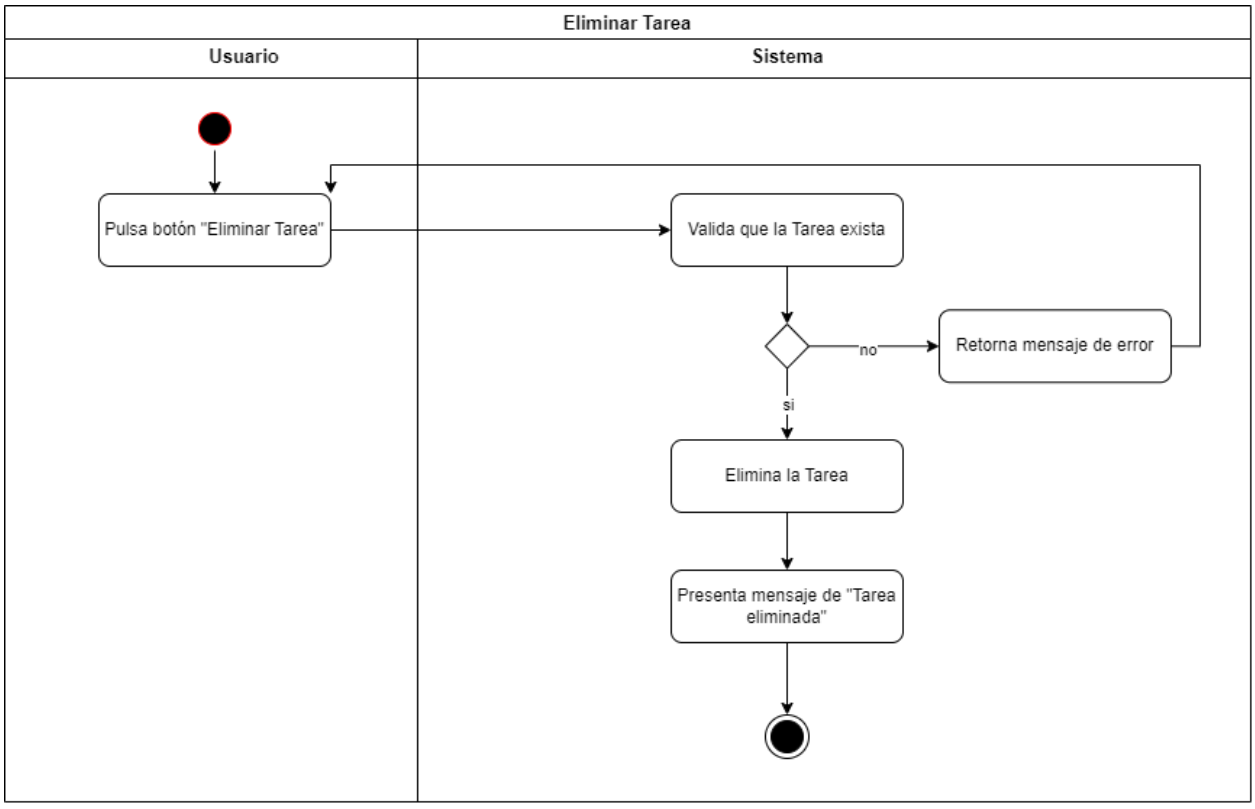
s. Diagrama de Actividades: Modificar Tarea



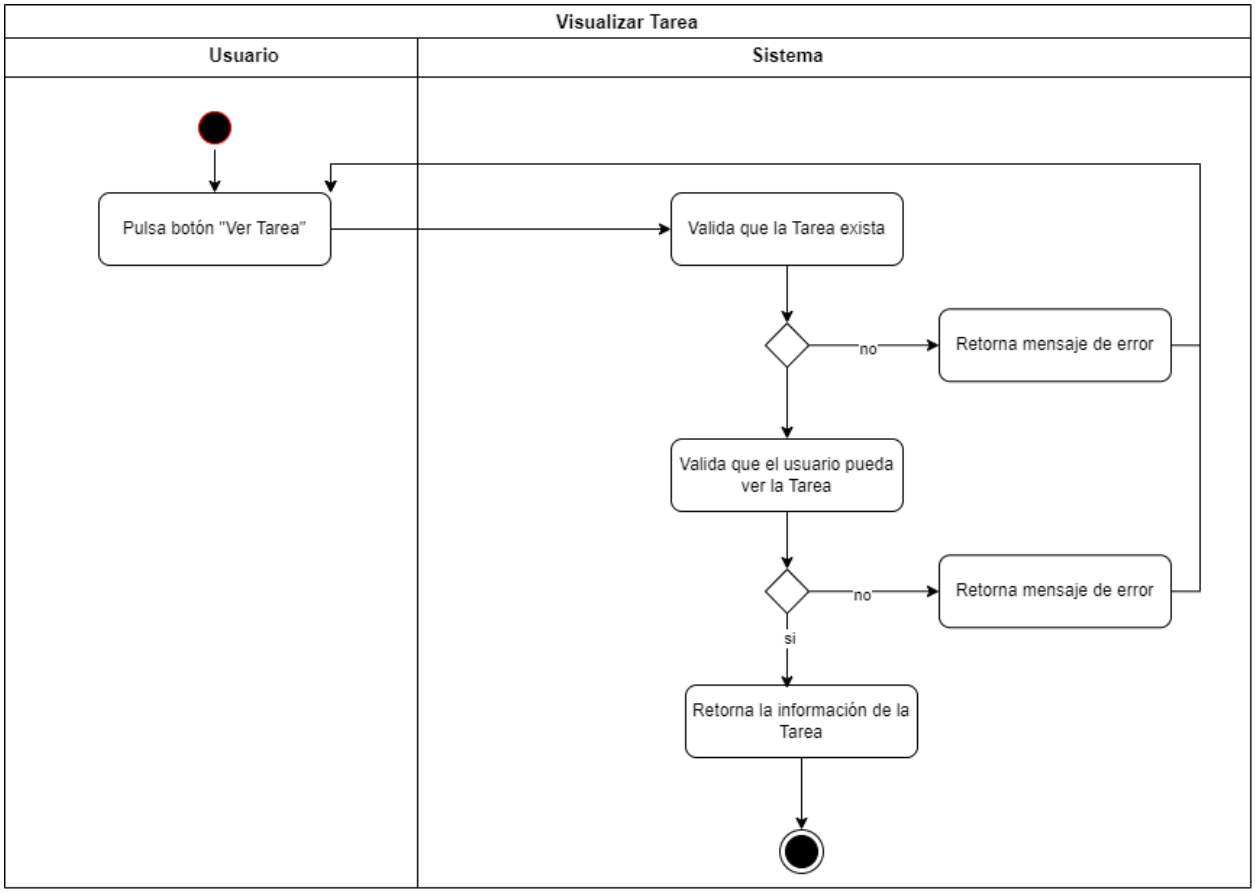
t. Diagrama de Actividades: Listar Tareas



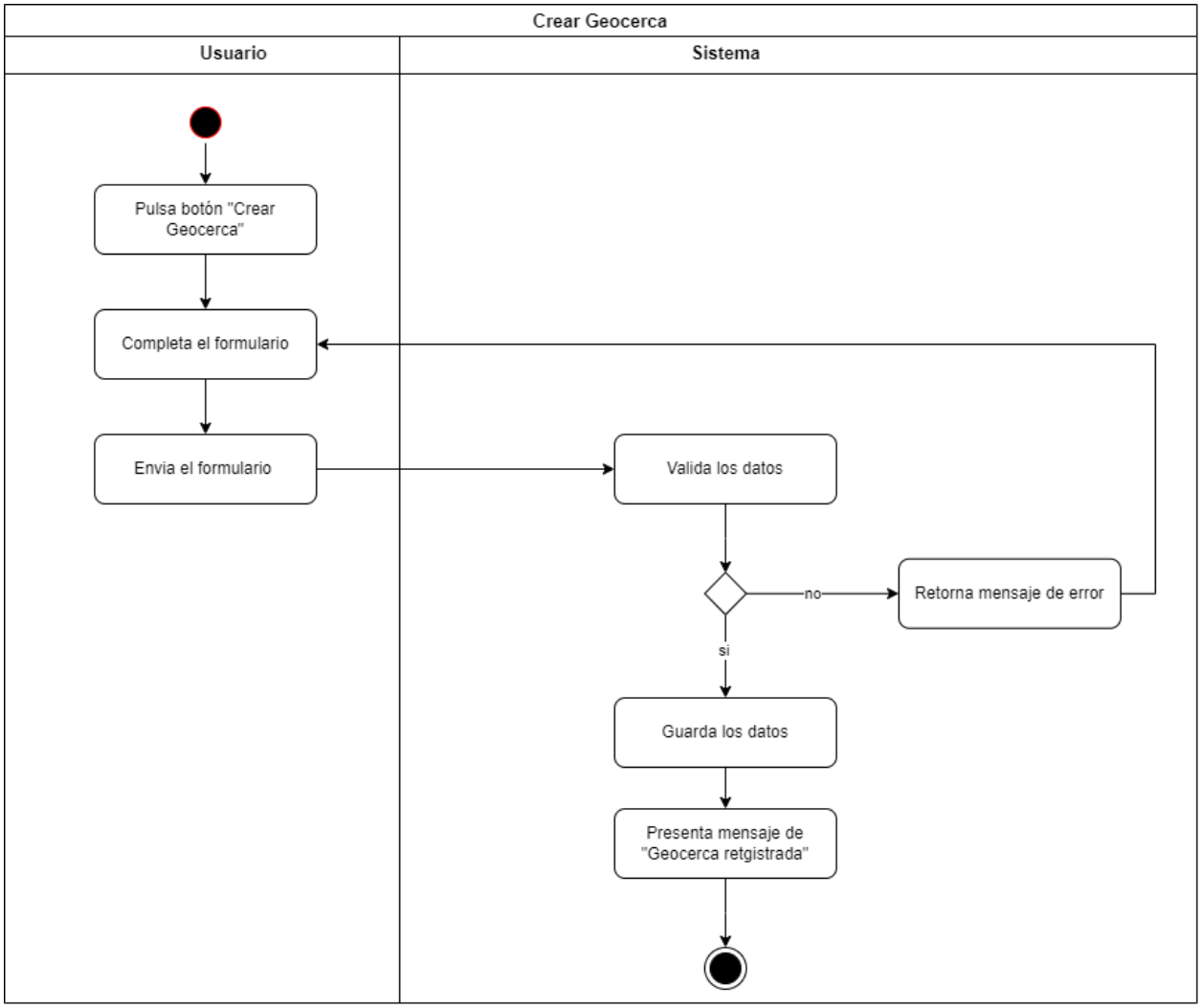
u. Diagrama de Actividades: Eliminar Tarea



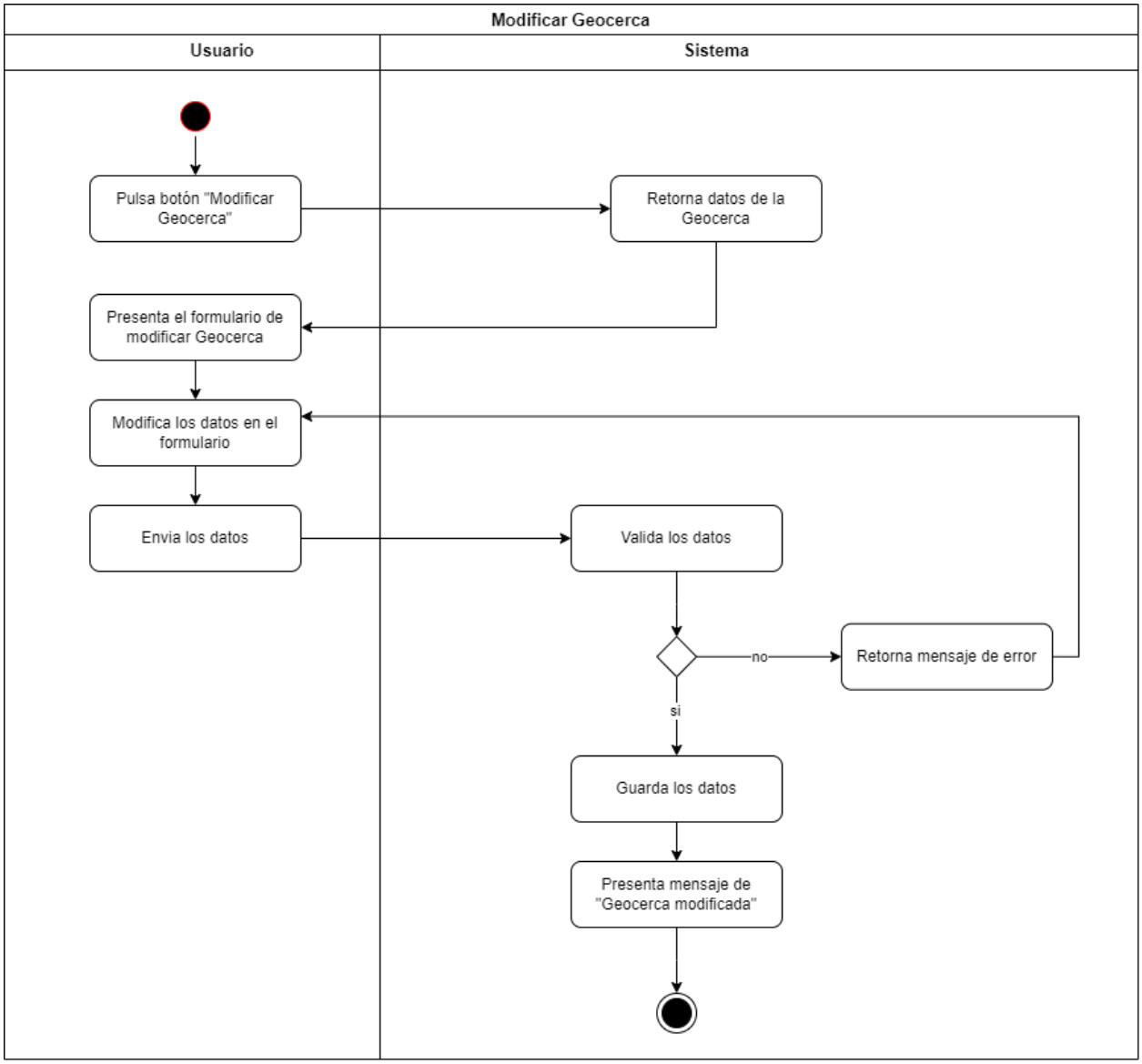
v. Diagrama de Actividades: Visualizar Tarea



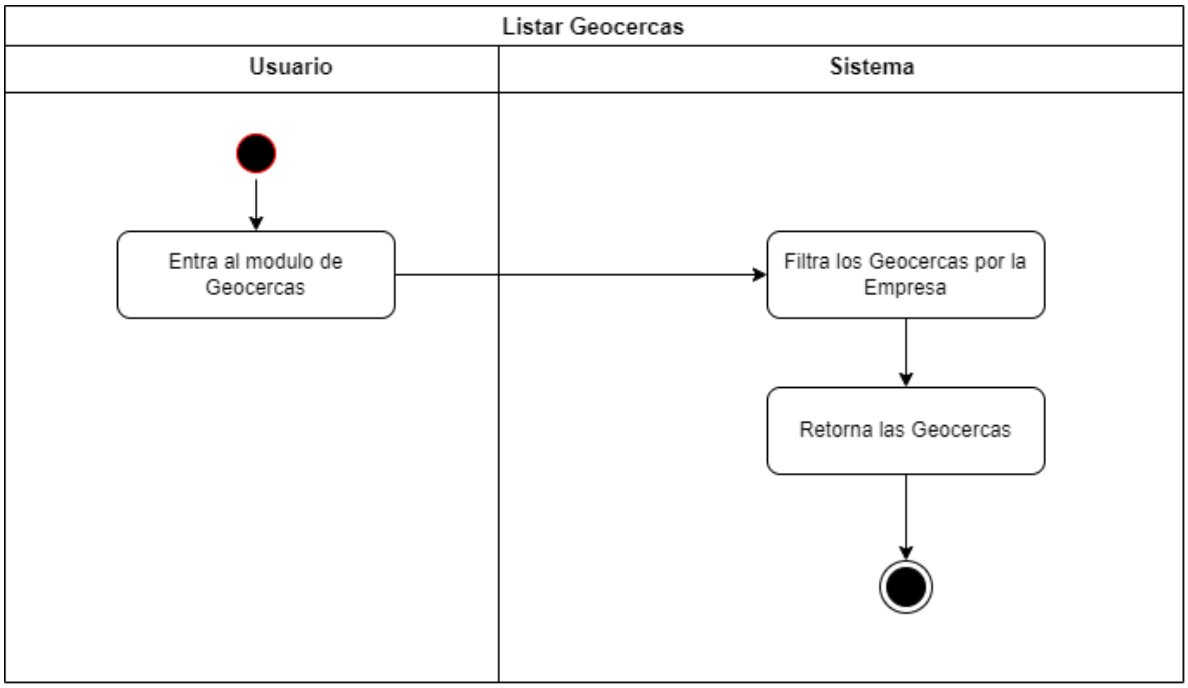
w. Diagrama de Actividades: Crear Geocerca



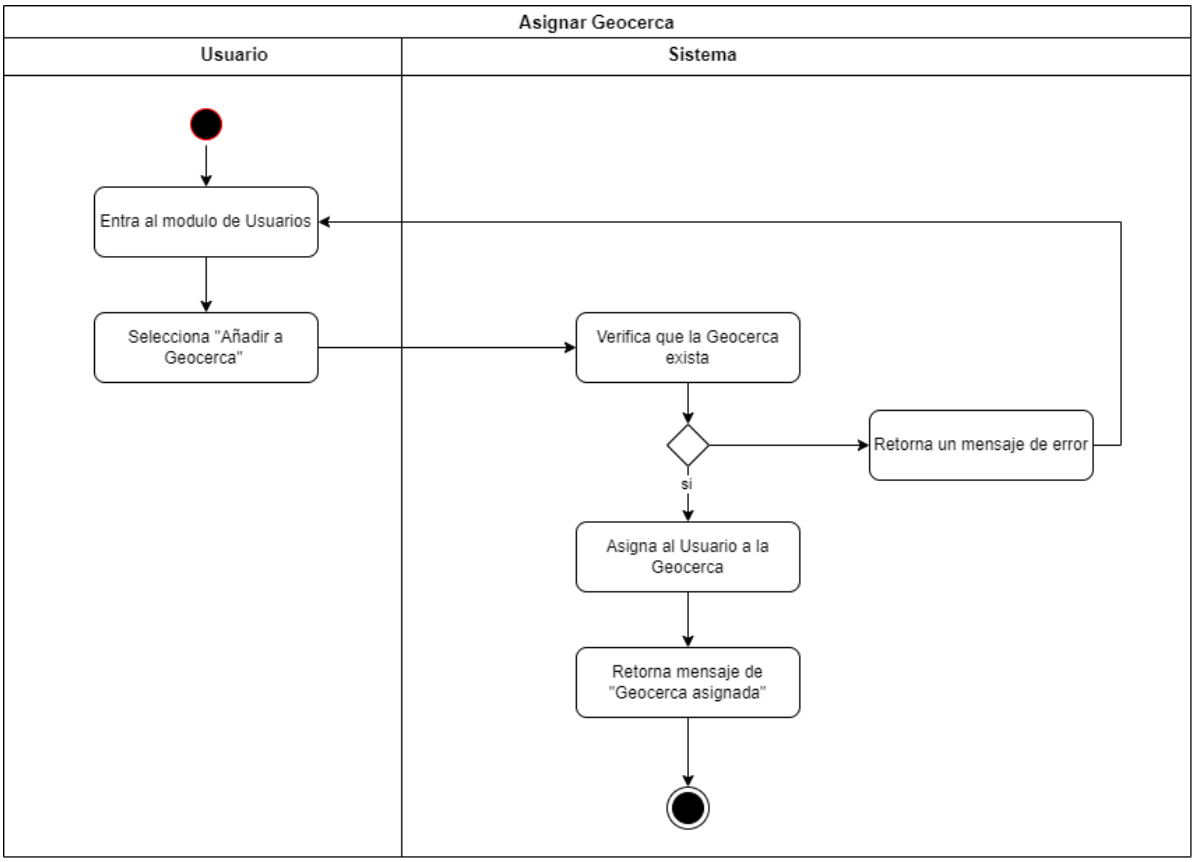
x. Diagrama de Actividades: Modificar Geocerca



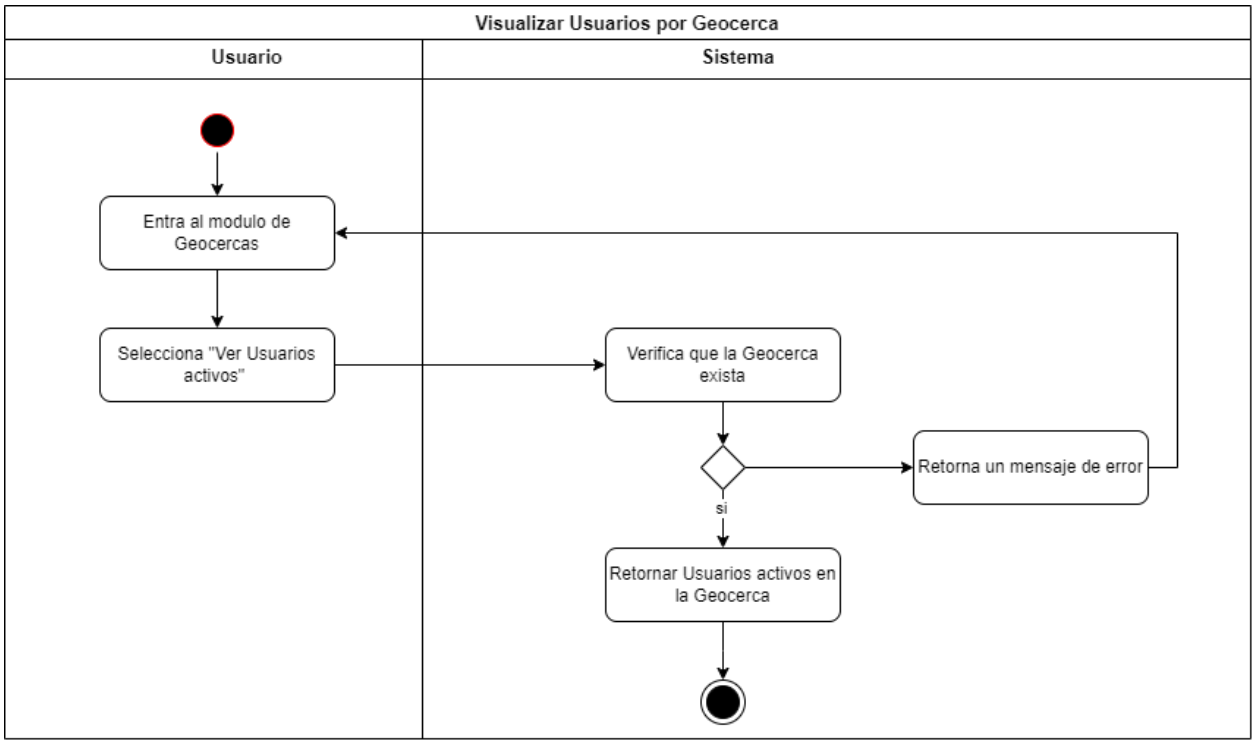
y. Diagrama de Actividades: Listar Geocercas



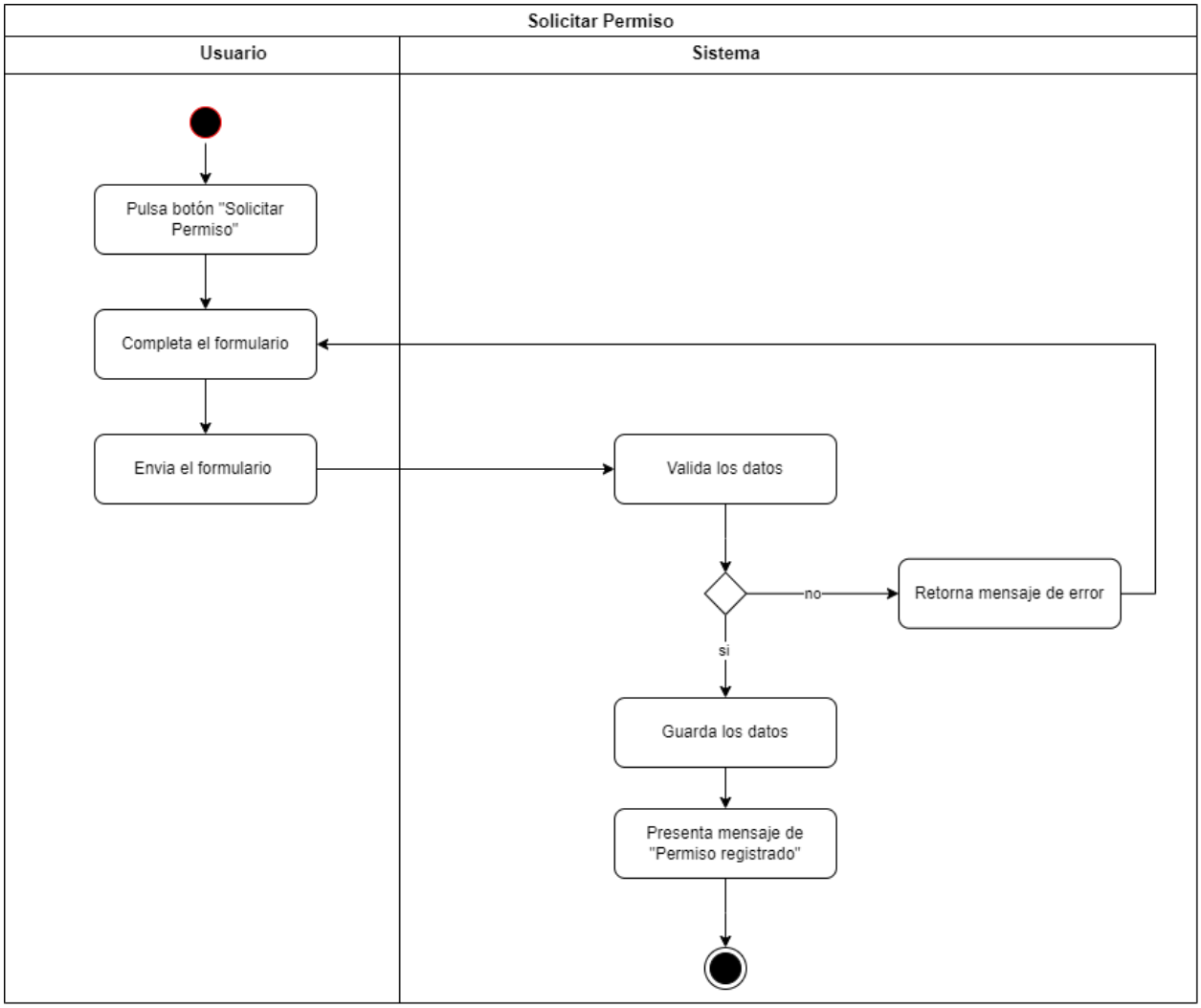
z. Diagrama de Actividades: Asignar Geocerca



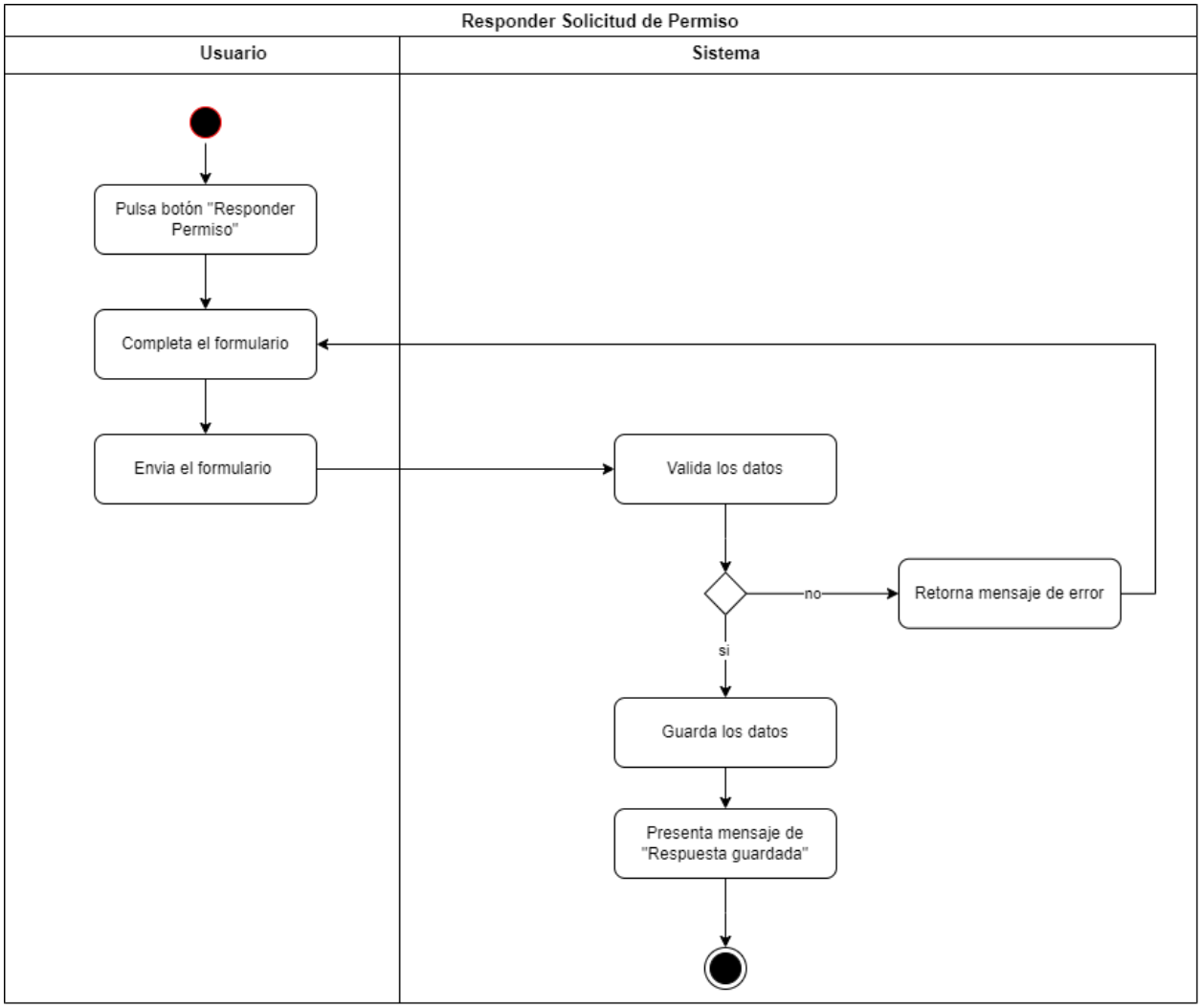
aa. Diagrama de Actividades: Visualizar Usuarios por Geocerca



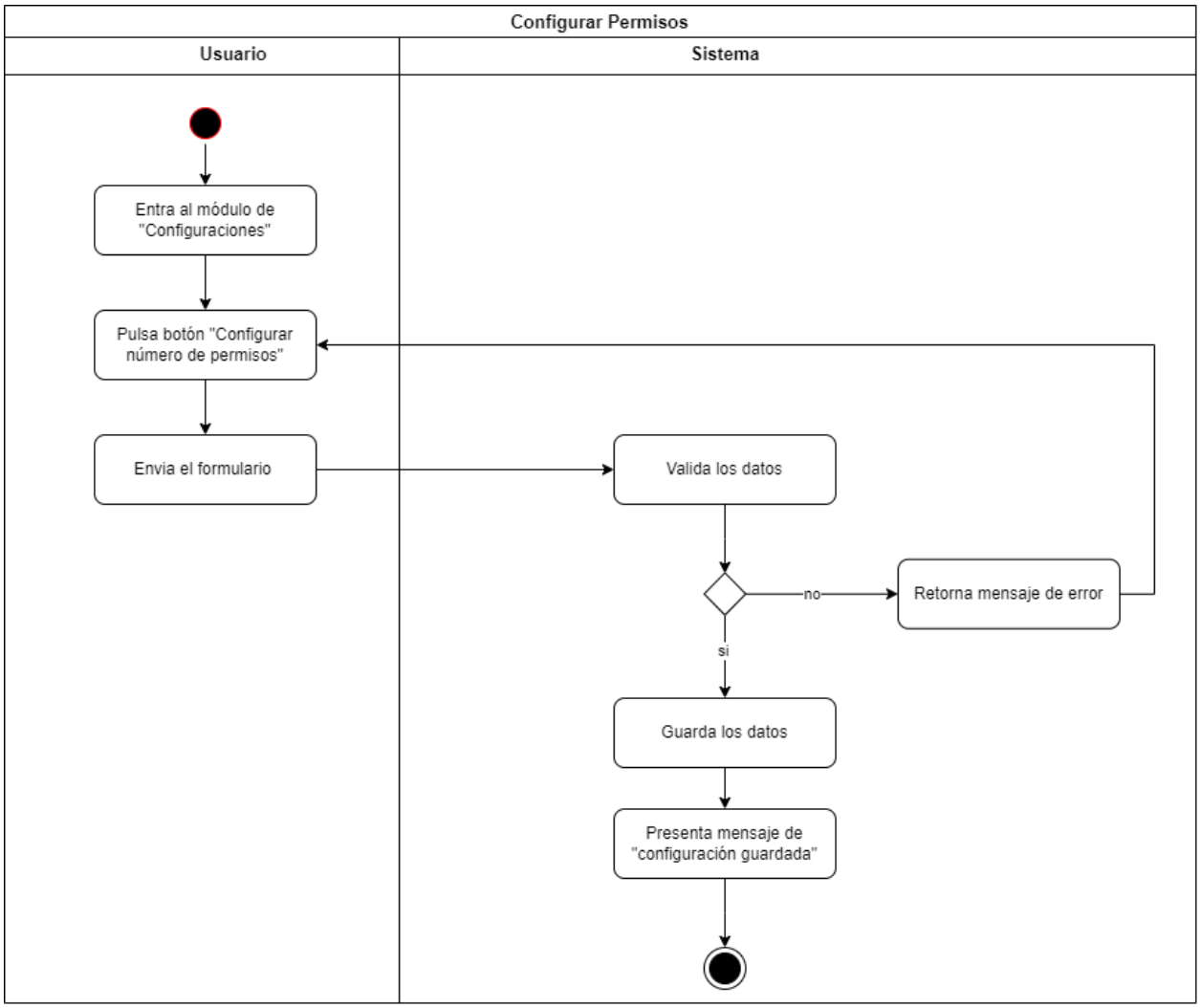
bb. Diagrama de Actividades: Solicitar Permiso



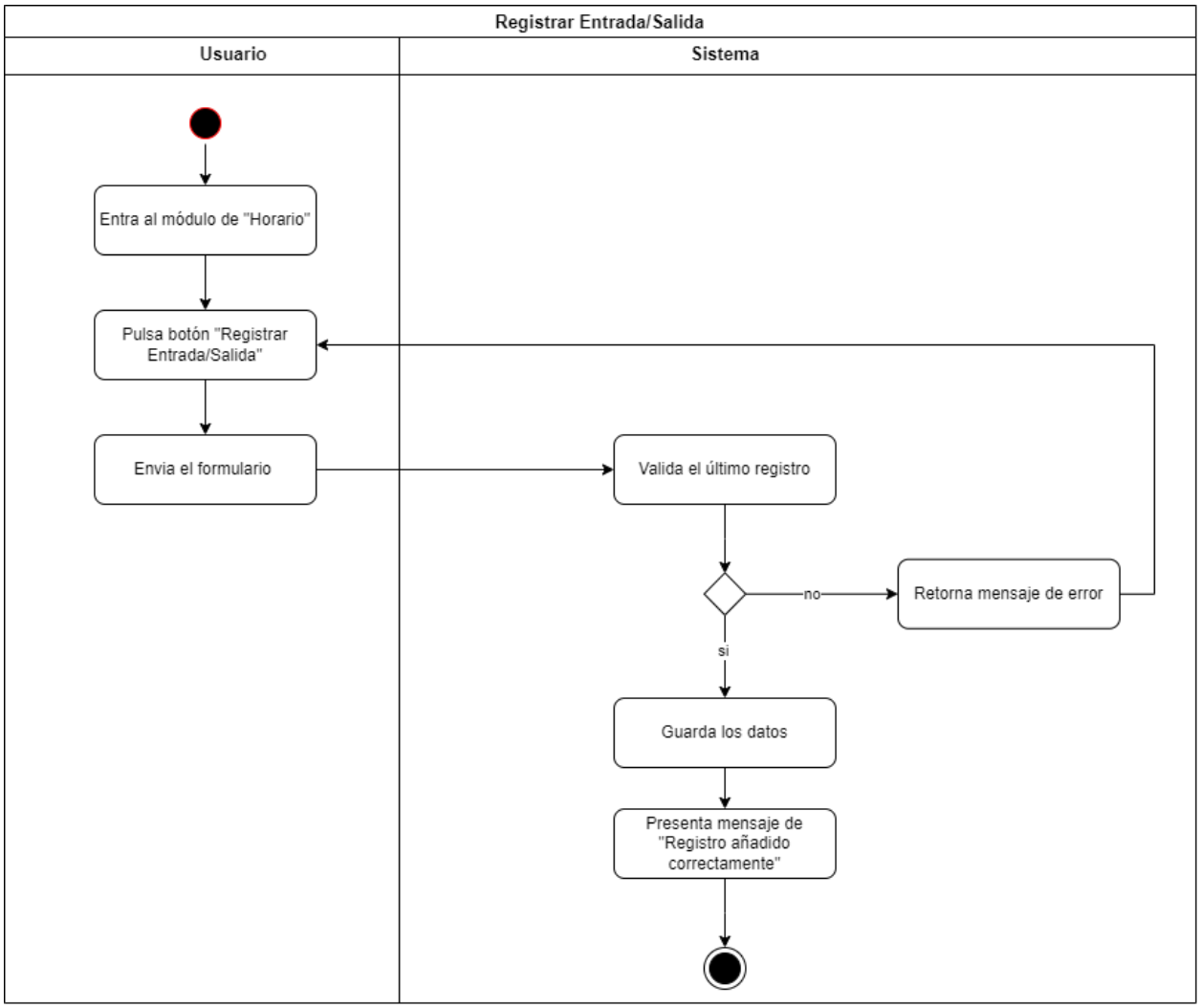
cc. Diagrama de Actividades: Responder Permiso



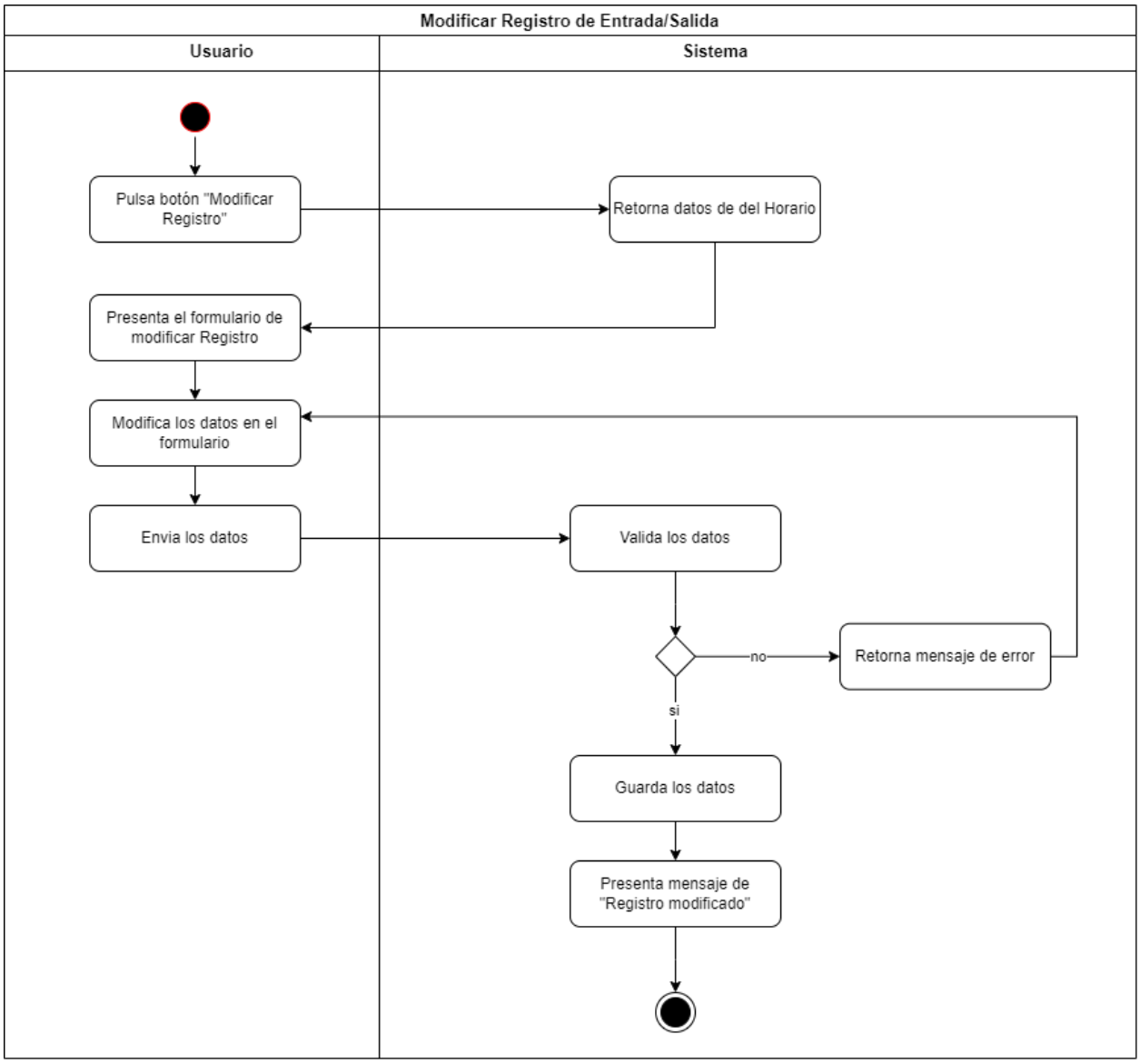
dd. Diagrama de Actividades: Configurar Permisos



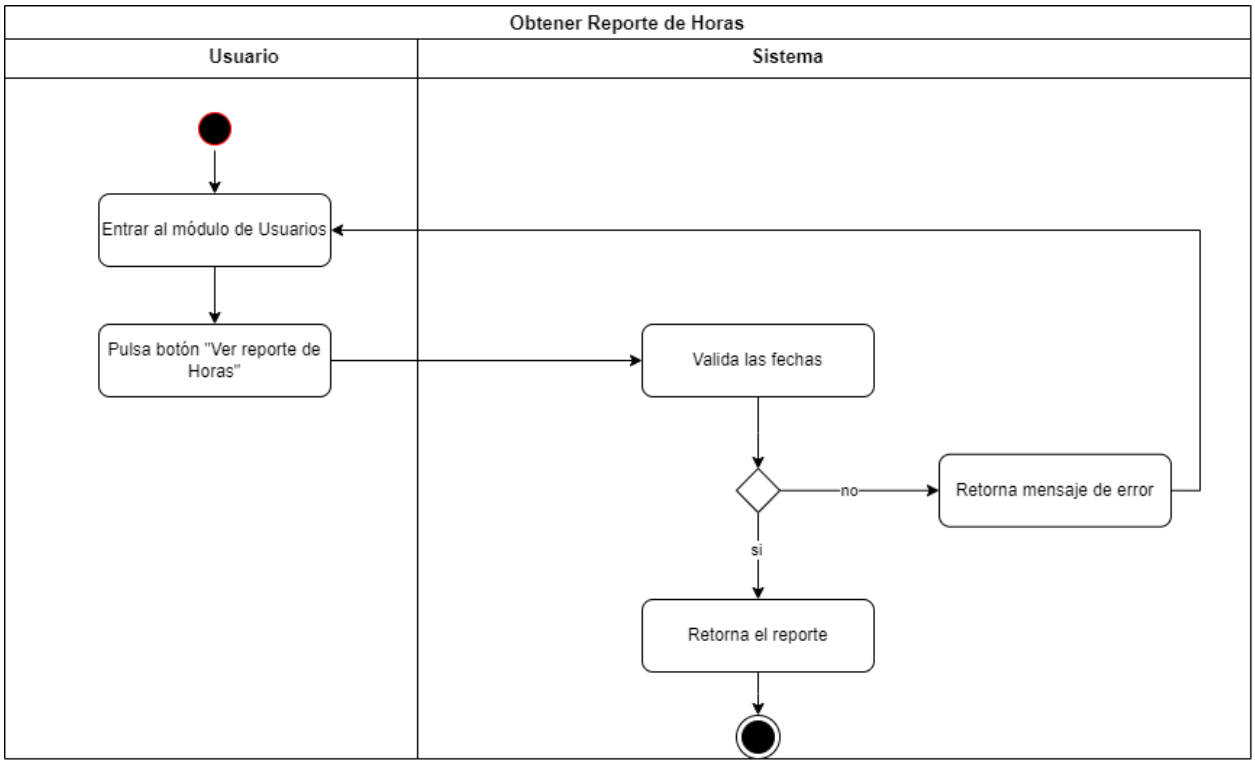
ee. Diagrama de Actividades: Registrar Entrada/Salida



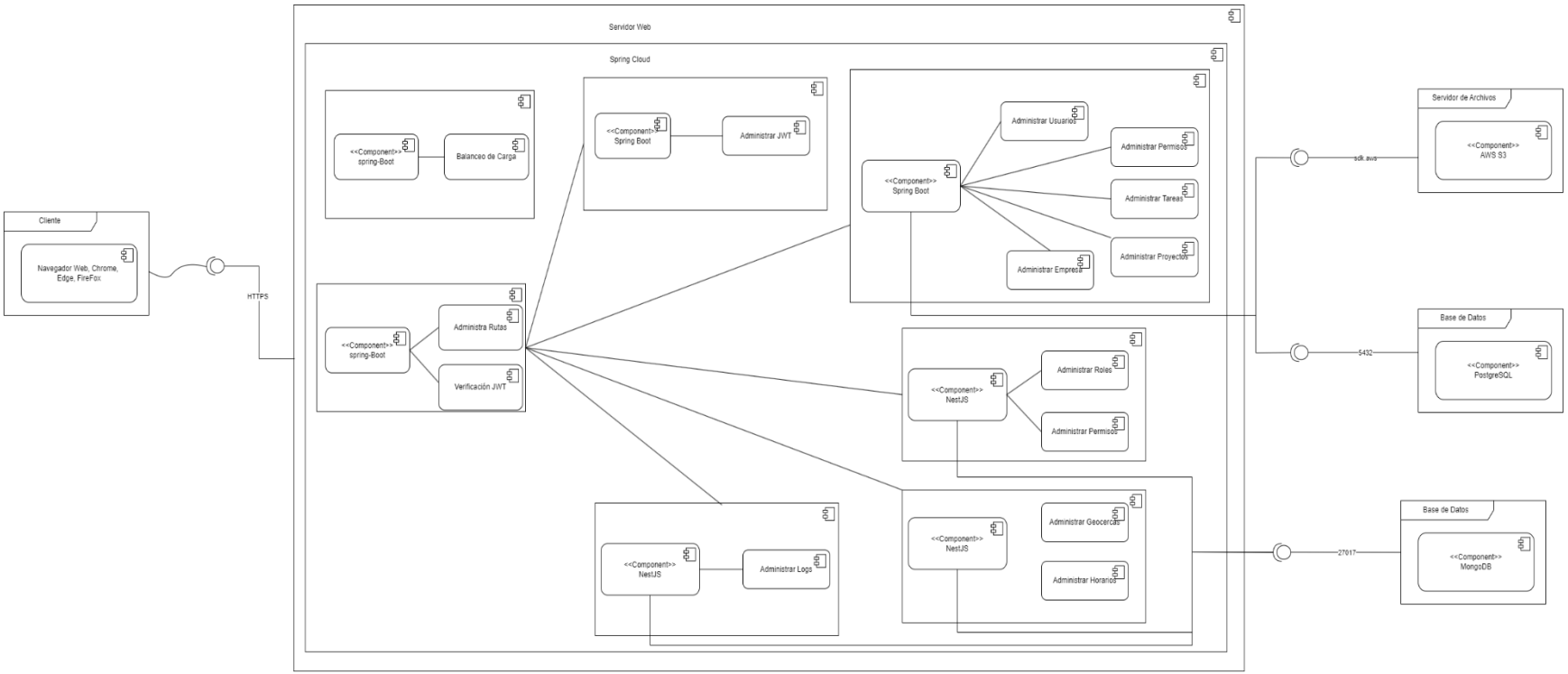
ff. Diagrama de Actividades: Modificar Registro



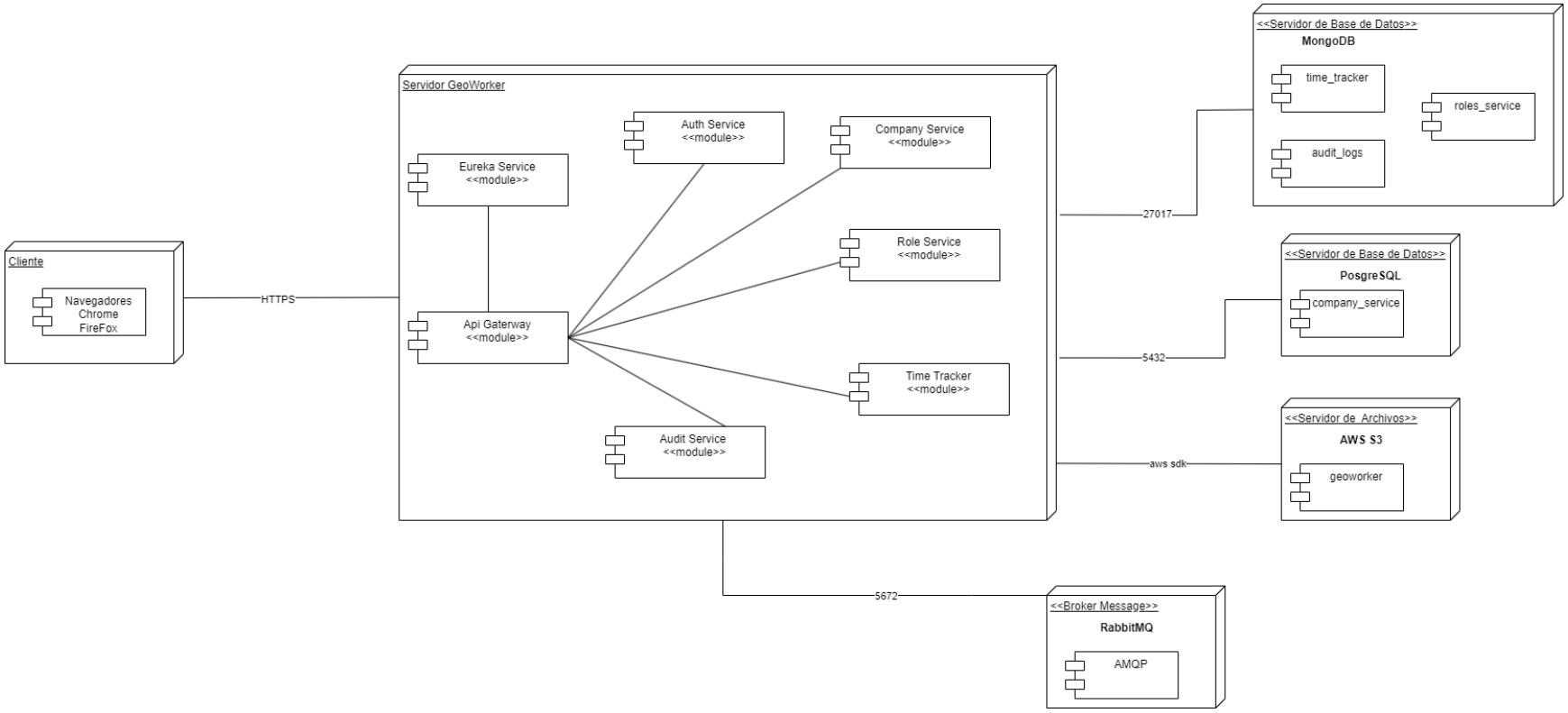
gg. Diagrama de Actividades: Obtener Reporte



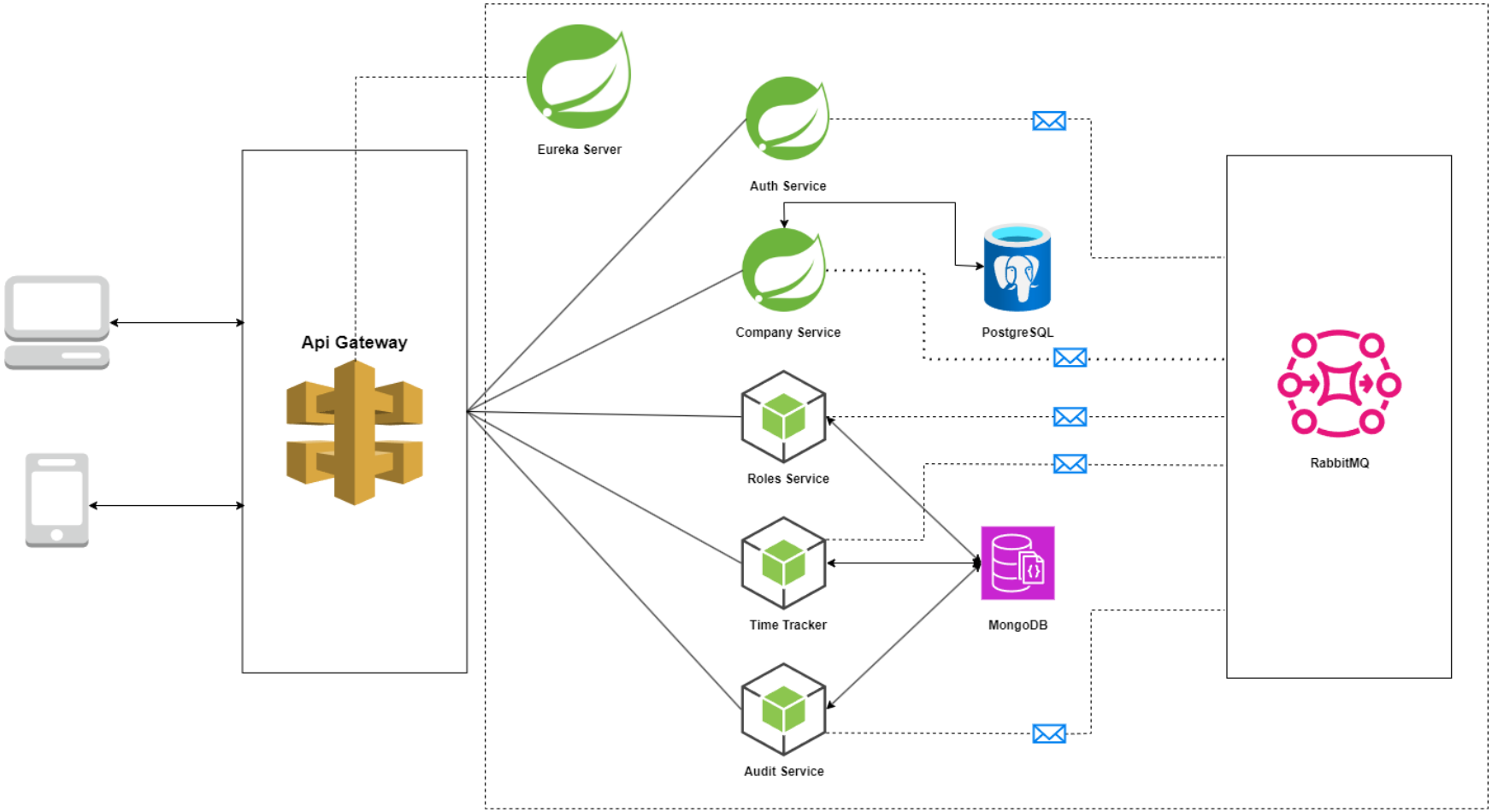
11. Vista de Despliegue



12. Vista Física



13. Arquitectura



Anexo 5. Encuesta de satisfacción realizada al Gerente de la Empresa

Encuesta de satisfacción de los servicios backend de la plataforma "GeoWorker"

El objetivo de la presente encuesta es realizar la verificación del nivel de satisfacción con la arquitectura implementada para los servicios backend de la plataforma "GeoWorker". Esta encuesta está dirigida al Sr. Jonathan Zhunaula en calidad de Gerente y Representante Legal de la empresa Consultora PuntoPymes CIA. LTDA.

1. **¿Cómo evaluaría la disponibilidad de la plataforma GeoWorker en comparación con otros proyectos monolíticos de la empresa?**
   - ☑ Mucho mejor
   - ☐ Mejor
   - ☐ Igual
   - ☐ Peor
   - ☐ Mucho peor
2. **¿Ha observado una mayor eficiencia operativa en la plataforma GeoWorker en comparación con otros proyectos?**
   - ☑ Sí, significativamente
   - ☐ Sí, moderadamente
   - ☐ No ha habido cambios
   - ☐ Ha empeorado moderadamente
   - ☐ Ha empeorado significativamente
3. **¿Se han reducido los tiempos de despliegue y actualización de la plataforma GeoWorker en comparación con otros proyectos?**
   - ☑ Sí
   - ☐ No
4. **¿Qué impacto ha tenido la implementación de microservicios en la capacidad de escalar la plataforma GeoWorker en comparación con otros proyectos monolíticos?**
   - ☑ Muy positivo
   - ☐ Positivo
   - ☐ Neutro
   - ☐ Negativo
   - ☐ Muy negativo
5. **¿Ha mejorado la disponibilidad de los servicios de GeoWorker durante escenarios de carga y estrés comparación con otros proyectos monolíticos?**
   - ☑ Sí, significativamente
   - ☐ Sí, moderadamente
   - ☐ No ha habido cambios
   - ☐ Ha disminuido moderadamente
   - ☐ Ha disminuido significativamente
6. **¿Cómo calificaría la eficiencia del monitoreo en la plataforma GeoWorker en comparación con otros proyectos monolíticos?**
   - ☑ Muy eficiente
   - ☐ Eficiente

- ☐ Neutro
- ☐ Ineficiente
- ☐ Muy ineficiente

7. **¿La adopción de microservicios y DevOps ha facilitado el mantenimiento de la plataforma GeoWorker en comparación con otros proyectos de desarrollo de software?**
   - ☑ Sí, mucho
   - ☐ Sí, algo
   - ☐ No ha habido cambios
   - ☐ Ha dificultado algo
   - ☐ Ha dificultado mucho
8. **¿Qué nivel de satisfacción tiene el equipo de desarrollo con la nueva infraestructura de microservicios y DevOps en términos de disponibilidad y rendimiento de los servicios backend desarrollados?**
   - ☑ Muy satisfecho
   - ☐ Satisfecho
   - ☐ Neutro
   - ☐ Insatisfecho
   - ☐ Muy insatisfecho
9. **¿Considera que la integración de DevOps ha mejorado la colaboración y comunicación dentro del equipo de desarrollo de GeoWorker en comparación con otros proyectos monolíticos?**
   - ☑ Sí, significativamente
   - ☐ Sí, moderadamente
   - ☐ No ha habido cambios
   - ☐ Ha empeorado moderadamente
   - ☐ Ha empeorado significativamente
10. **¿Recomendaría el uso de arquitectura de microservicios y técnicas de DevOps para el desarrollo de otros proyectos de su empresa?**
    - ☑ Sí
    - ☐ No

PuntoPymes
software development center
RUC: 1191732789001
PBX 07-3700520

Sr. Jonathan Zhunaula

CI. 1105003337

Gerente de la empresa PuntoPymes

Manuel Aguinsaca

CI. 1105947350

Estudiante

Anexo 6. Certificado de traducción de resumen

Certificado

Gloria M. Andrade C.
Licenciada en Ciencias de la Educación, Mención Ingles
CERTIFICA:

Que el Sr. Manuel Alejandro Aguinsaca Medina con Nro. de cédula 1105947350, autor del proyecto de fin de carrera: "**Implementación de una arquitectura de microservicios y técnicas de DevOps a los servicios backend de la plataforma GeoWorker**" ha cumplido con la traducción al idioma inglés del resumen del trabajo de acuerdo a la redacción y reglas gramaticales del idioma; de esta manera da cumplimiento con la sección 'Abstract' de dicho proyecto.

Es cuanto puedo certificar en honor a la verdad, facultando al interesado hacer uso del presente.

Loja, 26 de julio de 2024

Lic. Gloria M. Andrade C.
C.C. 1102182928