UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

CARRERA DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

PROYECTO DE INVESTIGACIÓN

"EVALUACIÓN DE MÉTRICAS DE CALIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS MEDIANTE EL DESARROLLO DE UN PROTOTIPO EN AMBIENTE WEB"

Proyecto de investigación presentado previo a la obtención del título de Ingeniero en Sistemas de Información

AUTORES:

Simba Taipe Danny Roberto

Sopalo Iza Eddy Geovanny

TUTOR:

Ing. Mg. Luis René Quisaguano Collaguazo

CO-TUTORA:

Ing. Mg. Verónica del Consuelo Tapia Cerda

LATACUNGA, FEBRERO 2025

DECLARACIÓN DE AUTORÍA

Nosotros, **SIMBA TAIPE DANNY ROBERTO** con C.I.: **0550644520** y **SOPALO IZA EDDY GEOVANNY** con C.I.: **0550299770**, declaramos ser autores del presente proyecto de Investigación: **"EVALUACIÓN DE MÉTRICAS DE CALIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS MEDIANTE EL DESARROLLO DE UN PROTOTIPO EN AMBIENTE WEB"**, siendo el **Ing. Mg. LUIS RENÉ QUISAGUANO COLLAGUAZO**, tutor del presente trabajo, eximamos expresamente a la Universidad Técnica de Cotopaxi y a sus representantes legales de posibles reclamos o acciones legales.

Además, certificamos que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de nuestra exclusiva responsabilidad.

Atentamente,

Simba Taipe Danny Roberto
CI: 0550644520

Eddy Geovanny Sopalo Iza
CI: 0550299770

AVAL DEL TUTOR DE PROYECTO DE INVESTIGACIÓN

En calidad de Tutor del Trabajo de Investigación con el título: **"EVALUACIÓN DE MÉTRICAS DE CALIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS MEDIANTE EL DESARROLLO DE UN PROTOTIPO EN AMBIENTE WEB"**, de los estudiantes: **DANNY ROBERTO SIMBA TAIPE** con C.I. **0550644520** y **EDDY GEOVANNY SOPALO IZA** con **C.I. 0550299770** de la Carrera de Ingeniería en Sistemas de la Información, considero que dicho Informe Investigativo cumple con los requerimientos metodológicos y aportes científico-técnicos suficientes para ser sometidos a la evaluación del Tribunal de Validación de Proyecto que el Honorable Consejo Académico de la Facultad de Ciencias de la Ingeniería y Aplicadas de la Universidad Técnica de Cotopaxi designe, para su correspondiente estudio y calificación.

Latacunga, febrero 2025

.....

Ing. MSc. Luis René Quisaguano Collaguazo

CC: 1721895181

APROBACIÓN DEL TRIBUNAL DE TITULACIÓN

En calidad de Tribunal de Lectores, aprueban el presente Informe de Investigación de acuerdo a las disposiciones reglamentarias emitidas por la Universidad Técnica de Cotopaxi, y por la Facultad de **CIENCIAS DE LA INGENIERÍA Y APLICADAS**; por cuanto, los postulantes: **Danny Roberto Simba Taipe** con C.I. **055064452-0** y **Eddy Geovanny Sopalo Iza** con C.I. **055029977-0**, con el título del proyecto de investigación: "**EVALUACIÓN DE MÉTRICAS DE CALIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS MEDIANTE EL DESARROLLO DE UN PROTOTIPO EN AMBIENTE WEB**", han considerado las recomendaciones emitidas oportunamente y reúne los méritos suficientes para ser sometido al acto de Sustentación del Proyecto.

Por lo antes expuesto, se autoriza realizar los empastados correspondientes, según la normativa institucional.

Latacunga, Febrero 2025

Por constancia firman:

Ing. Mg. Jorge Bladimir Rubio Peñaherrera
CC: 0502222292
LECTOR 1 (PRESIDENTE)

Ing. Mg. Diego Geovanny Falconí Punguil
CC: 0550080774
LECTOR 2 (MIEMBRO)

Ing. Mg. Karla Susana Cantuña Flores
CC: 0502305113
LECTOR 3 (MIEMBRO)

12 de febrero de 2025

AVAL DE IMPLEMENTACIÓN

Mediante el presente pongo en consideración que, los señores estudiantes: **DANNY ROBERTO SIMBA TAIPE** con C.I. **055064452-0** y **EDDY GEOVANNY SOPALO IZA** con C.I. **055029977-0**, realizaron el proyecto de titulación: **"EVALUACIÓN DE MÉTRICAS DE CALIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS MEDIANTE EL DESARROLLO DE UN PROTOTIPO EN AMBIENTE WEB"**, contribuyendo de esta manera con el Proyecto de Investigación Formativa: **"ESTUDIO DE MÉTRICAS DE DEPENDENCIA Y COMPLEJIDAD PARA EL DISEÑO DE ARQUITECTURA DE MICROSERVICIOS"**, trabajo que fue presentado y aprobado de manera satisfactoria.

.....

Ing. Mg. Tapia Cerda Verónica del Consuelo

C.C: 0502053697

AGRADECIMIENTO

En el trascurso de todo este gran viaje he tenido altas y bajas, he conseguido metas que no esperaba lograr alcanzar, esto fue gracias a mi capacidad y no solo eso, los que me acompañaron durante largo camino.

Este agradecimiento es a mis padres por brindarme la libertad de emprender mi propia vida y guiarme constantemente, a pesar de mi bajo nivel académico ellos no se rindieron conmigo, me dieron una mano logrando llegar hasta donde estoy ahora.

Antes deseaba conocer gran cantidad de personas, y hoy conozco a muchas personas las cuales agradezco conocer, en esta ruta que tome he conocido a mis amigos de la universidad los cuales no fueron solo pocos años los cuales nos conocimos, gracias a ellos las debilidades que solía tener se han desvanecido.

Para finalizar este agradecimiento, agradecería a los ingenieros que nos enseñaron por tantos años, comencé sin saber nada, solo cosas básicas, pero gracias a ellos ahora se gran cantidad de cosas que me serán muy útiles en el ámbito laboral y humanístico, sus guías fueron las más útiles para madurar de un simple estudiante a un ingeniero.

Danny Simba

AGRADECIMIENTO

En primer lugar, tomaría este espacio y agradecer a mi madre Rosa Elvira Iza Guaña y a mi padre Patricio Sopalo Tipan, quienes me han brindado apoyo en primera fila, tanto en el ámbito económico, pero más importante en el ámbito emocional, quienes siempre me han aconsejado y colmado de bendiciones para cumplir con mis objetivos.

En todo el momento que he llevado estudiando, he logrado establecer amistades que me han guiado y acompañado en todo el transcurso de la carrera, en especial a mis amigos Leandro, Luis, Daniela y Danny por su apoyo, así que les agradezco profundamente.

Además, agradezco a la universidad técnica de Cotopaxi, así como a todos los docentes por encaminar y compartir sus conocimientos a lo largo de toda mi formación profesional.

Eddy Sopalo

DEDICATORIA

Dedico este logro a mi madre María Taipe y a mi padre José Simba, por apoyarme todos estos años las cuales no fueron fáciles, pero aun así nos criaron a mí y mis hermanos, nunca fue una tarea fácil, pero sin ellos yo no sería nada, ustedes me demostraron que todo es posible aun que las cosas sean difíciles.

Este no es solo un título es algo más grande, quienes no confiaron en mí, los cuales se rieron y me miraban desde arriba, este título es dedicado a todas esas personas que confiaron en mi principalmente a mis mejores amigos el comandante Jesús Chasiluisa y el Ing. Christopher Vega, ellos demostraron que es amistad.

A mi compañero de tesis que realizamos una gran cantidad de cosas para llevar a cabo nuestra titulación, sacrificios como grandes logros y mucho esfuerzo para alcanzar este gran logro.

A las personas que me enseñaron grandes cantidades de cosas, ingenieros de la carrera de Sistemas de Información gracias por enseñarme, instruirme en este proceso tan largo y tenerme paciencia han dejado una huella imborrable académicamente y personal.

Para finalizar esta dedicatoria, este título no es solo mío sino también suyo, me ayuda a entender una frase hay que vivir con orgullo y con la frente en alto, si te vence tu debilidad, calienta tu corazón, aprieta los dientes y sigue adelante.

Danny Simba

DEDICATORIA

Dedico este logro a mis padres Rosa Elvira Iza y Patricio Sopalo Tipan, por apoyarme en todos mis estudios y hacer todo lo posible para que pueda estudiar.

Esto no solo es un logro más, es un triunfo que quedara marcado para toda mi vida y lo llevare siempre conmigo.

A mi compañero de tesis por haber realizado este proyecto en conjunto, en las buenas y en las malas, el cual con mucho esfuerzo hemos logrado conseguir.

Para dar fin a este dedicatoria, menciono a mis amigos, lo cuales me han dado apoyo y acompañado durante estos años de carrera. Por su amistad muchas gracias.

Eddy Sopalo

UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

TÍTULO

"**EVALUACIÓN DE MÉTRICAS DE CALIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS MEDIANTE EL DESARROLLO DE UN PROTOTIPO EN AMBIENTE WEB**"

Autores:

Simba Taipe Danny Roberto

Sopalo Iza Eddy Geovanny

RESUMEN

El presente trabajo se desarrolló dentro de la Universidad Técnica de Cotopaxi, específicamente en la carrera de Sistemas de Información como parte del proyecto de investigación formativa titulado "Estudio de métricas de dependencia y complejidad para el diseño de arquitecturas de microservicios", mediante el cual se evidenció la dificultad de medir dichos parámetros de dependencia y complejidad en proyectos de desarrollo de software. Para abordar esta problemática, se plantea como objetivo evaluar métricas enfocadas en estos ámbitos, con el fin de utilizarlas en el desarrollo de un prototipo web que demuestre su aplicabilidad. Para lograr esto se empleará un modelo inteligente de evaluación denominado MOAM, el cual consta de dos componentes principales; el primero se encarga de generar una arquitectura con base a historias de usuario, mientras que el segundo calcula métricas de calidad sobre dicha arquitectura. Estas métricas permiten comparar los resultados frente a una arquitectura propuesta por el grupo de investigación. La justificación de este trabajo radica en la necesidad de evaluar métricas en un enfoque de microservicios que es fundamental para garantizar la calidad y eficiencia de este tipo de arquitecturas. Gracias a este proceso, se logró proponer dos arquitecturas sometidas a un proceso de comparación para de esta forma elegir la que proporcione mejores resultados de dependencia y complejidad que entre en un rango aceptable de aceptación. Como resultado se obtuvo un sistema que demuestra que la aplicación de métricas de calidad en una arquitectura de microservicios permite un modelo eficiente evaluado en ámbitos de dependencia y complejidad.

Palabras Claves: arquitectura, dependencia, complejidad, acoplamiento, comparación.

TECHNICAL UNIVERSITY OF COTOPAXI

FACULTY OF ENGINEERING SCIENCES AND APPLIED

THEME: "Evaluation of Quality Metrics for Microservices-Based Applications Through the development of a web environment prototype"

Authors:

Simba Taipe Danny Roberto

Sopalo Iza Eddy Geovanny

ABSTRACT

This research was carried out at the Technical University of Cotopaxi, specifically in the Information Systems program, as part of a research "Study of dependency and complexity metrics for designing microservices architectures." This project highlighted the difficulty of measuring these dependency and complexity parameters in software development projects. To address this issue, evaluating metrics focused on these areas was the goal, with the aim of using in a web prototype development that demonstrates their applicability. To achieve this, an intelligent evaluation model called MOAM will be used. This model has two main components: the first generates an architecture based on user stories, while the second calculates quality metrics for that. These metrics allow for a comparison of the results against an architecture proposed by the research group. The justification for this, lies in the need to evaluate metrics in a microservices approach, which is essential to ensure the architectures quality and efficiency. Through this process, two architectures were proposed and compared to choose the one that provides better results by developing a web prototype. As a result, a system was obtained that shows applying quality metrics to a microservices architecture allows for an efficient model evaluated in terms of dependency and complexity.

Keywords: architecture, dependence, complexity, coupling, comparison.

UNIVERSIDAD TÉCNICA DE COTOPAXI

CENTRO DE IDIOMAS

AVAL DE TRADUCCIÓN

En calidad de Docente del Idioma Inglés del Centro de Idiomas de la Universidad Técnica de Cotopaxi; en forma legal **CERTIFICO** que:

La traducción del resumen al idioma Inglés del proyecto de investigación cuyo título versa: "**EVALUACIÓN DE MÉTRICAS DE CALIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS MEDIANTE EL DESARROLLO DE UN PROTOTIPO EN AMBIENTE WEB**" presentado por: **Simba Taipe Danny Roberto** y **Sopalo Iza Eddy Geovanny**, egresados de la Carrera de: **Ingeniería en Sistemas de Información**, perteneciente a la **Facultad de Ciencias de la Ingeniería y Aplicadas**, lo realizaron bajo mi supervisión y cumple con una correcta estructura gramatical del Idioma.

Es todo cuanto puedo certificar en honor a la verdad y autorizo a los peticionarios hacer uso del presente aval para los fines académicos legales.

Latacunga, 20 febrero del 2025

Atentamente,

CENTRO DE IDIOMAS SECRETARIA

Mg. Lidia Rebeca Yugla Lema

DOCENTE CENTRO DE IDIOMAS-UTC

CI: 0502652340

CENTRO DE IDIOMAS

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INFORMACIÓN GENERAL

Título: Evaluación de métricas de calidad de aplicaciones basadas en microservicios mediante el desarrollo de un prototipo en ambiente web.

Tipo de Proyecto: Proyecto de investigación

Carrera: Ingeniería de Sistemas de Información.

Proyecto de investigación vinculado: Estudio de métricas de dependencia y complejidad para el diseño de arquitectura de microservicios.

Equipo de Trabajo:

Tutor de titulación:

Nombre: Ing. Mg. Quisaguano Collaguazo Luis René

Cedula: 1721895181

Correo: luis.quisaguano1@utc.edu.ec

Cotutor:

Nombre: Ing. Verónica de Consuelo Tapia Cerda. Mg

Cedula: 0502053697

Correo: veronica.tapia@utc.edu.ec

Estudiante 1:

Nombre: Simba Taipe Danny Roberto

Cedula: 0550644520

Correo electrónico: danny.simba4520@utc.edu.ec

Estudiante 2:

Nombre: Sopalo Iza Eddy Geovanny

Cedula: 0550299770

Correo electrónico: eddy.sopalo9770@utc.edu.ec

ÁREA DEL CONOCIMIENTO: 06 información y Comunicación (TIC) / 061 Información y Comunicación (TIC) / 0611 El uso del Ordenador, 0613 Software y desarrollo y análisis de aplicativo.

LÍNEA DE INVESTIGACIÓN: Tecnología de la Información y las comunicaciones, robótica, automatización y optimización de sistemas.

SUB LÍNEA DE INVESTIGACIÓN DE LA CARRERA: Ciencias informáticas para la modelación de Sistemas de información a través del desarrollo de software.

2. INTRODUCCIÓN

En la actualidad, las aplicaciones y sitios de internet que se utiliza como redes sociales y aplicaciones de productividad, surgieron gracias al desarrollo de software, esto con el fin de automatizar procesos según las necesidades de empleados o usuarios finales. Partiendo de este punto se llega al análisis de estos sistemas, así como la pregunta de qué manera fueron desarrollados y se mantienen. Aunque a día de hoy se sigue utilizando métodos tradicionales de desarrollo de software, los cuales consisten en almacenar todas las funcionalidades y componentes en un solo código fuente ya no es una opción viable debido a su defecto en el ámbito de escalabilidad, en el que, si una pequeña parte requiere actualización, es necesario hacer cambios en todo el sistema para que pueda adaptarse a dicho cambio. Esto ya que las funcionalidades dependen de un solo paquete, en el que, si se llegara a saturar ya sea de peticiones o datos resultaría en la caída de todo el sistema, como también los datos o procesos que se hayan estado llevando a cabo antes de que ocurriera el fallo.

Con todo esto dicho se ha logrado establecer una opción más viable, la cual es la adopción de arquitecturas basadas en microservicios, en la cual los desarrolladores de software podrán enfocarse en un solo servicio individual y desacoplado, en lugar de toda la aplicación. Sin embargo, su adopción no solo implica la separación de funcionalidades, sino también el diseño de una arquitectura correcta que garantice su escalabilidad y mantenibilidad. En este contexto, la arquitectura hexagonal se ha posicionado como una opción efectiva para lograr un mayor modularidad en el desarrollo de microservicios.

Uno de los aspectos importantes en la evaluación de calidad de los sistemas basados en microservicios es el análisis de métricas de dependencia y complejidad. Estas permiten identificar dependencias críticas entre servicios, por ejemplo, métricas de acoplamiento entre servicios, dependencias internas y complejidad estructural, los cuales son unos de los indicadores clave para evaluar una arquitectura.

De esta manera la evaluación se vuelve un aspecto fundamental para medir su eficiencia y sostenibilidad en estos sistemas. Esta investigación se centra en el análisis y evaluación de métricas de calidad aplicadas en una arquitectura de microservicios, con el objetivo de determinar su impacto en los ámbitos de dependencia y complejidad.

Para abordar el asunto del problema, se comienza diciendo que el desarrollo de aplicaciones web ha tenido una mejora considerable los últimos años, a medida que la demanda sube, los requisitos de calidad se han vuelto más exigentes, dejando de lado antiguas arquitecturas de software, las cuales, aunque fueron funcionales en su momento, presentan limitaciones en cuanto a adaptabilidad con las nuevas demandas del mercado.

A partir de este problema han aparecido mejores opciones, como los sistemas elaborados en base a arquitecturas de microservicios, que son una opción más viable que arquitecturas antiguas ya que permiten la adaptabilidad y mejora. Sin embargo, a pesar de sus ventajas, estas arquitecturas no están libres de desafíos. Uno de los principales problemas radica en los equipos de desarrollo, de los cuales no muchos siguen prácticas establecidas para definir una arquitectura solida en base a métricas adecuadas que beneficien la correcta dependencia y complejidad de los servicios. En lugar de priorizar una base bien estructurada que sirva de pilar en la implementación de microservicios, gran parte de su enfoque recae en resolver problemas y evaluar métricas una vez que la aplicación está en funcionamiento como el uso de métricas para determinar latencia, seguridad, tolerancia a la carga, resiliencia, reusabilidad.

La falta de evaluación en la arquitectura de microservicios no solo resulta perjuiciosa en la calidad y sostenibilidad de los sistemas basados en estos, sino que también limita una evaluación efectiva de métricas clave durante el inicio del ciclo de vida del desarrollo, reduciendo la posibilidad de que las aplicaciones cumplan con estándares de calidad requeridos y estén preparadas para evolucionar sin tener que preocuparse por el ámbito de dependencia y complejidad. Este estudio se enfocará en la evaluación de las métricas de dependencia y complejidad, ya que estos aspectos suelen recibir poca atención en comparación con otras métricas de software, así que, al analizarlas, se busca comprender mejor su impacto en la calidad del software y en la eficiencia del desarrollo.

2.1. FORMULACIÓN DEL PROBLEMA

¿De qué manera podría medirse la dependencia y complejidad en las arquitecturas de microservicios?

2.2. OBJETO Y CAMPO DE ACCIÓN

Objeto de estudio: Aplicación de microservicios y su arquitectura

Campo de acción: 1203.18 Sistemas de Información, Diseño Componentes

Caso de Estudio: Camper Plus

2.3. BENEFICIARIOS

Tabla 1. Especificación de beneficiarios del proyecto de investigación.

Beneficiarios	*Cargo*	*Descripción*	*N de personas*
Directos	*Docentes*	Posibilidad de enseñar a los estudiantes la forma de evaluar y aplicar métricas de dependencia y complejidad	15
	Grupo de investigación	Proporcionar un marco estructurado para medir métricas en una arquitectura de microservicios	10
Sub Total Beneficiarios Directos			25
Indirectos	*Estudiantes de la carrera de sistemas de la información y software*	Mejoramiento de técnicas y habilidades al momento de desarrollo de arquitecturas	292 Sistemas 31 Software
Sub Total Beneficiarios Indirectos			*323*
Total, de beneficiarios			*548*

2.4. JUSTIFICACIÓN

La investigación propuesta sobre “Evaluación de métricas de calidad de aplicaciones basadas en microservicios mediante el desarrollo de un prototipo en ambiente web” será de mucha importancia ya que ayudará en la evaluación de una arquitectura de microservicios, enfocada en el ámbito de dependencia y complejidad.

En primer lugar, el correcto planteamiento de una arquitectura de microservicios en un proyecto, así como en su aplicación es de vital importancia para asegurar su calidad y funcionamiento. Una mala implementación puede causar problemas, como perdida de datos e interrupciones en su productividad. Entonces al utilizar métricas de calidad en etapas de

desarrollo de la arquitectura, se puede asegurar que se cumpla con estándares aceptables, aumentando su eficiencia y reduciendo riesgos en el sistema.

Comúnmente las métricas se usan una vez que el sistema está en funcionamiento, limitando la capacidad de identificar y corregir problemas en etapas tempranas de desarrollo. Esta investigación propone un cambio de enfoque, al aplicar métricas de dependencia y complejidad desde la fase inicial del diseño y desarrollo.

Desde el punto de vista teórico, esta investigación demostrará el uso innovador de métricas de dependencia y complejidad para resolver problemas en fases de desarrollo de arquitectura de software, promoviendo buenas prácticas y métodos que podrían influir en la forma en que se diseñan y evalúan estas arquitecturas a futuro.

En cuanto al enfoque práctico, el desarrollo de un prototipo permitirá la validación de la implementación de esta arquitectura evaluada en un entorno real, ya que permitirá apreciar cómo su utilización mejora la forma en la que los desarrolladores agrupan microservicios, partiendo de una arquitectura enfocada en reducir su complejidad.

Por lo que concierne a la factibilidad, la evaluación de métricas será viable en el ámbito de la fase de definición de una arquitectura dentro del ciclo de desarrollo de software, además de que servirá para su validación, con el fin de determinar si está bien planteada.

Este proyecto será beneficioso para desarrolladores de software, específicamente en el campo de los microservicios, puesto que permitirá aplicar un enfoque no muy común, con el cual obtendrán la ventaja de aplicar parámetros de dependencia y complejidad en la arquitectura de este tipo de aplicaciones.

2.5. HIPÓTESIS

La aplicación de métricas de calidad en una arquitectura de microservicios permitirá evaluar los ámbitos de dependencia y complejidad, optimizando el diseño arquitectónico.

2.6. OBJETIVOS

2.6.1. Objetivo General

Evaluar métricas de dependencia y complejidad en una arquitectura de microservicios mediante el análisis de los resultados obtenidos por MOAM para la posterior aplicación en el desarrollo de un prototipo web.

2.6.2. Objetivos Específicos

- Realizar la revisión bibliográfica sobre métricas de calidad en arquitecturas de microservicios con énfasis en dependencia y complejidad, para la elaboración de la fundamentación teórica.
- Comparar los resultados obtenidos de las métricas de calidad entre la arquitectura propuesta por MOAM frente al grupo de investigación, y aplicarlo en el desarrollo de un prototipo web.
- Desarrollar un prototipo web haciendo uso de una arquitectura optimizada con el fin de validar su aplicación.

2.7. SISTEMA DE TAREAS

Tabla 2. Sistemas de tareas por objetivo a desarrollar.

Objetivos específicos	Actividad (tareas)	Resultados esperados	Técnicas, Medios e Instrumentos
Realizar la revisión bibliográfica sobre métricas de calidad en arquitecturas de microservicios con énfasis en dependencia y complejidad, para la elaboración de la fundamentación teórica.	Búsqueda de información en fuentes confiables.	Marco Teórico	Técnica: Análisis bibliográfico Medios: Google académic, artículos y revistas Instrumentos: Ficha bibliográfica
	Fundamentar conocimiento sobre métricas de calidad y dependencia más influyentes.	Métricas de dependencia y complejidad	Técnica: Análisis bibliográfico Medios: Google académic, artículos y revistas Instrumentos: Ficha bibliográfica
Comparar los resultados obtenidos de las métricas de calidad entre la arquitectura propuesta por MOAM frente al grupo de investigación, y aplicarlo en el desarrollo de un prototipo web.	Realizar una comparación de resultados para tomar una decisión de arquitectura	Arquitectura a utilizar	Técnicas: Diseño basado en dominios y análisis de resultados Medios: MOAM Instrumentos: Gráficas de comparación

Objetivos específicos	Actividad (tareas)	Resultados esperados	Técnicas, Medios e Instrumentos
Desarrollar un prototipo web haciendo uso de una arquitectura optimizada con el fin de validar su aplicación.	Desarrollar el prototipo con la arquitectura seleccionada	Microservicios funcionales	Técnica: Arquitectura hexagonal, metodología de desarrollo XP Medios e instrumentos: Herramientas de desarrollo (Visual Studio Code) Instrumentos: Iteraciones
	Desplegar el prototipo en un ambiente web	Prototipo desplegado	

3. FUNDAMENTACIÓN TEÓRICA

3.1. ANTECEDENTES

En los últimos años, ha habido una evolución significativa en lo que respecta el desarrollo de aplicaciones web, tomando como punto de partida los sistemas monolíticos primitivos, como además arquitecturas orientadas a servicios (SOA), estas ofreciendo un gran beneficio al ser reutilizables en distintas interfaces con solo realizar su invocación, combinándose y creando funcionalidades más elaboradas [1].

En 2022 se realizó un estudio de métricas de calidad en la Universidad Técnica de Cotopaxi. Esta propuesta se desarrolló bajo la problemática de uso de una arquitectura antigua, la cual carece de flexibilidad y escalabilidad en el desarrollo de aplicaciones [2]. Este trabajo se enfocó en la implementación de una arquitectura basada en microservicios, evaluando distintas métricas de calidad para analizar cómo se estructuran los servicios y sus dependencias. Esta propuesta se desarrolló además con el fin de determinar un modelo de optimización haciendo uso de métricas de calidad para identificar problemas de dependencia y complejidad, los cuales son puntos clave para el proceso de desarrollo de una correcta arquitectura de microservicios, al mismo tiempo que se busca aplicar el modelo más óptimo en la implementación de un caso práctico de microservicios.

Como resultado, se obtuvo que el uso de este modelo permite a los desarrolladores crear aplicaciones basadas en microservicios con una menor dependencia y complejidad, además de facilitar la agrupación de historias de usuario de distintas maneras para reducir la dependencia en la fase de diseño.

3.2. MICROSERVICIOS

Los microservicios, son servicios pequeños que trabajan realizando sus propios procesos dentro de una aplicación mucho más grande, la cual está compuesta de varios servicios. La arquitectura ofrece ventajas, pero también plantea desafíos alrededor de las pruebas, puesto que una aplicación puede tener varios o cientos de servicios que funcionan juntas y cada una de ellas requieren ser probadas a medida que evolucionan [3].

Son aplicaciones pequeñas que funciona de manera independiente y realizan o generan operaciones que se dividen en los servicios que contienen las aplicaciones. Gonzalo Castillo se refiere a los microservicios como un estilo arquitectónico de desarrollo de software que se comunican mediante APIs (Application Programming Interface) [4]. En la actualidad los usuarios buscan mejoras en la respuesta de aplicaciones web en busca de la mejora de la flexibilidad y optimización, donde los programadores encuentran mejores métodos para realizar proyectos más escalables, como los microservicios se ejecutan de manera independiente de la actualización y escalamiento más sencillo que otros métodos.

3.2.1. Métricas de calidad

Se refiere a los estándares o indicadores diseñados y utilizados como medio de evaluación de desempeño, mantenibilidad y escalabilidad de un sistema de información, en la que ya no se incluye solo términos de precio, sino que también es de importancia el brindar funciones y características que aporten a la confiabilidad del software.

La evaluación de la calidad de la arquitectura debe llevarse de la mano de un modelo, este debe estar presente desde el proceso de creación de software, gestión de características y establecimiento de semejanzas entre la elaboración y pasos iniciales [5]. Los enfoques más importantes cuando se habla de métricas de calidad de arquitecturas de microservicios recaen en la dependencia y complejidad de estos.

3.2.1.1. Dependencia

Hace referencia a las veces que un componente requiere de otro para poder funcionar. Uno de los principios de la arquitectura de microservicios es que sus componentes estén acoplados en lo más mínimo posible, puesto que buscan no exceder en las interacciones o llamadas a otros servicios, esto con el fin de que puedan ser ejecutados independientemente [6].

3.2.1.2. Complejidad

Es un punto clave para que cualquier aplicación crezca sin disminuir su escalabilidad, además de permitir analizar lo difícil que puede llegar a ser entender, implementar y mantener el programa [7]. En un enfoque de métricas nos permite medir y estimar características del software para tomar decisiones ya sea preventivas o correctivas.

3.3. DESCOMPOSICIÓN EN MICROSERVICIOS

En el diseño de la arquitectura, es fundamental abordar la separación del sistema teniendo en cuenta las funcionales que cada módulo aporta o formula, así como los actores involucrados. Según Iranzo, el enfoque debe centrarse en las capacidades que aportan valor al negocio [8]. Cada microservicio agrupa sus propios modelos de datos de forma independiente y solo muestra aquellos que necesiten ser compartidos haciendo uso de una interfaz visual. Por ejemplo, al segmentar trabajador y padres se garantiza que cada usuario interactúe con sus respectivas funciones posibles.

3.3.1. Diseños basados en dominios (DDD)

Es un enfoque para el desarrollo de software que se basa en el análisis detallado y la presentación del modelo. Este enfoque de diseño de dominio protege la lógica del negocio que es compleja y más constante, lo que permite abstraer la complejidad de los casos de uso. Esto facilita la evolución y los cambios controlados en el software. Proporciona una ventaja a diseñar bien los microservicios. En el artículo de publicado por Microsoft dice que la regla general es que un servicio debe hacer “algo”, pero llevarla a cabo requiere una minuciosa reflexión [9].

El objetivo principal de DDD se basa centra en garantizar que el desarrollo sea correcto y este de la mano con los requerimientos y procesos de un negocio o empresa. Se asegura que el sistema demuestre de manera precisa la lógica y las normas del negocio.

Esta idea de separar el dominio se alinea perfectamente con la filosofía del patrón de diseño de Puertos y Adaptadores, que también se conoce como arquitectura hexagonal [10]. En el diseño basado en dominios estratégicos se define la estructura a gran escala de la aplicación, el cual ayuda a garantizar a permanecer centrado en los casos de negocio, todo esto se explica de mejor manera en la Figura 1.

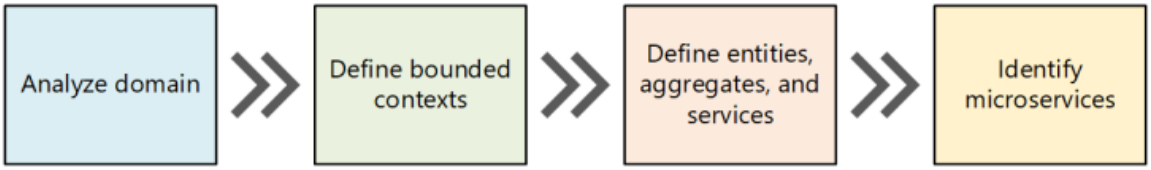

Figura 1. Proceso para identificar los microservicios [9].

3.3.1.1. Análisis del dominio (Analyze domain)

Hace referencia a las circunstancias del mundo real de una solución, el dominio proporciona información sobre las necesidades y los criterios que deben cumplir para que el creador implemente el sistema [11].

3.3.1.2. Contextos Delimitados (Bounded Contexts)

Un contexto particular se define como un marco conceptual donde se utiliza un modelo específico. Tratar el modelo de negocio en su totalidad es demasiado extenso y complicado, lo que complica su entendimiento y hace poco viable la administración de un único modelo para toda la empresa [10]. La definición exacta de contextos particulares ayuda a determinar si algún elemento de un modelo busca incluir diversos de estos.

3.3.1.3. Definición de entidades, agregadas y servicios (Define entities, aggregates, and services)

Se realiza un examen exhaustivo de las entidades clave y las conexiones entre ellas, así como de los servicios fundamentales que permiten la comunicación con estas entidades [12]. Este estudio es importante para diseñar el modelo de dominio de forma coherente, garantizando que represente correctamente los requerimientos y expectativas del negocio.

3.3.1.4. Identificación de los microservicios (Identify microservicios)

Se transforma en un servicio reducido que asume un grupo específico de tareas, promoviendo una sola responsabilidad y asegurando una alta cohesión junto con un bajo acoplamiento en su arquitectura [12]. Esto resulta crucial para que, cada parte del sistema coincida con los objetivos, favoreciendo una operación rápida.

3.4. SOFTWARE DE EVALUACIÓN DE CALIDAD DE MICROSERVICIOS

3.4.1 MOAM

Llamado Microservices Optimization Architectures Model, es un modelo inteligente desarrollado en la Universidad Técnica de Cotopaxi, tiene como característica principal la medición de métricas de calidad de arquitecturas de microservicios, específicamente en el ámbito de métricas de dependencia y complejidad. Sumamente útil en la evaluación de la arquitectura de microservicios [13].

Está conformado por un modelo matemático que divide en dependencia estimada y complejidad estimada, cada una con sus respectivas métricas que intervienen en el calcula de la arquitectura se muestran de mejor manera en la Figura 2.

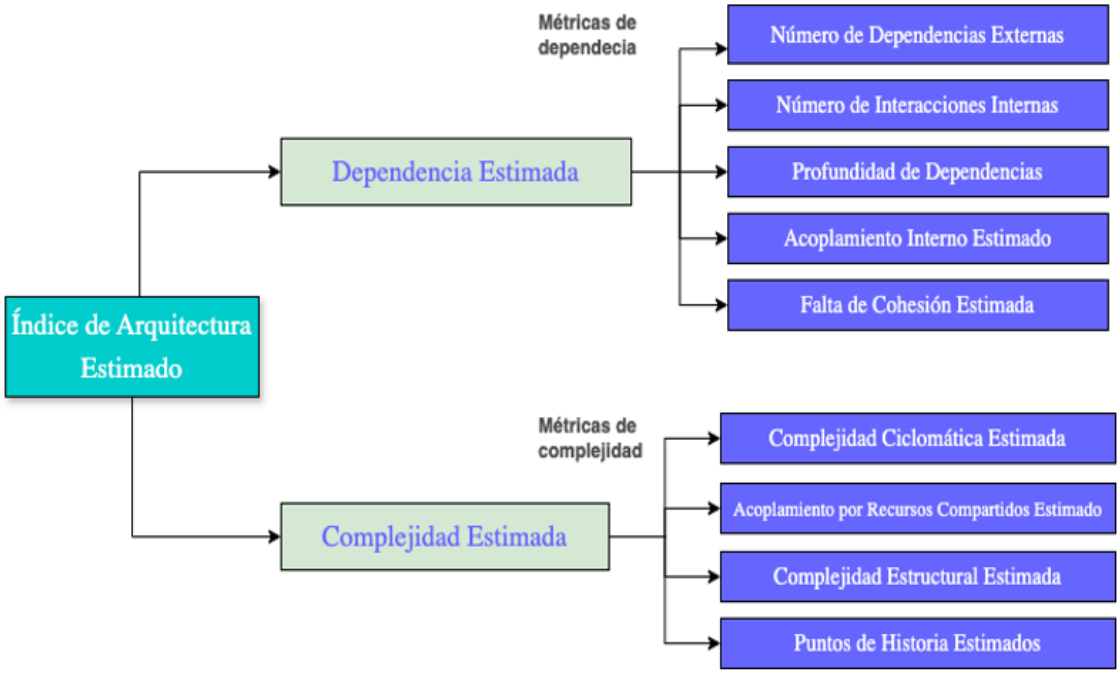

Figura 2. Métricas de dependencia y complejidad que interviene en MOAM [13].

3.4.1.1. Métricas de dependencia

Están enfocadas en medir como los microservicios se relaciona entre sí y con sistemas externos. Son cruciales para evaluar el acoplamiento entre servicios. Entre las cuales intervienen.

3.4.1.1.1. Número de dependencias externas (ND)

Cuenta los microservicios que tienen dependencias externas [14].

3.4.1.1.2. Número de interacciones internas (NI)

Suma el número total de dependencias internas entre microservicios [14].

3.4.1.1.3. Profundidad de dependencias (PD)

Determina la longitud del camino más largo de dependencias internas [14].

3.4.1.1.4. Acoplamiento estimado (AC_est)

Calcula el acoplamiento interno basado en el número de dependencias internas y la relación con el número máximo de conexiones posibles [14].

3.4.1.1.5. Falta de cohesión estimada (LCOH_est)

Mide la cohesión dentro de un microservicio utilizando similitud coseno entre las descripciones textuales de sus funcionalidades [14].

3.4.1.1.6. Dependencia estimada (D_EST)

Agrega ND, NI, PD y LCOH_est para obtener una medida global de dependencia [14].

3.4.1.2. Métricas de complejidad

Se enfocan en medir la dificultad de entender, mantener y escalar un microservicio o la arquitectura en general. Es un aspecto multifacético en el ámbito del desarrollo ya que representa la sofisticación de la arquitectura y el diseño general de un sistema [15].

3.4.1.2.1. Complejidad ciclomática estimada (CC_est)

Calcula la complejidad ciclomática de la arquitectura basada en sus dependencias internas [14].

3.4.1.2.2. Coeficiente de acoplamiento estimado (CA_est)

Evalúa el grado de acoplamiento debido a recursos compartidos entre microservicios [14].

3.4.1.2.3. Complejidad estructural estimada (CE_est)

Combina el número de microservicios y sus interacciones internas para lograr medir la complejidad estructural [14].

3.4.1.2.4. Puntos de historial estimada (PH_est)

Suma todas las funcionalidades de los microservicios para estimar el esfuerzo de desarrollo [14].

3.4.1.2.5. Complejidad estimada (C_est)

Combina CC_est, CA_est, CE_est y PH_est para obtener una medida global de complejidad [14].

3.4.1.2.6. Índice agregado estimado (AI_est)

Suma D_est y C_est para obtener un índice global de la arquitectura [14].

3.5. ARQUITECTURA HEXAGONAL

Esta arquitectura es una propuesta de implementación de separación por capas, la cual se divide en 3 niveles. En esta se definen componentes claves puerto y adaptadores.

El nombre de arquitectura de puertos y adaptadores radica en enfatizar que, la capa de presentación se encarga de alojar los puertos, mientras la capa de infraestructura se encarga de contener los adaptadores, encargados de conseguir datos de los puertos y adaptarlos a la lógica de negocio de la aplicación. La Figura 3 muestra la separación de responsabilidades entre los diferentes niveles.

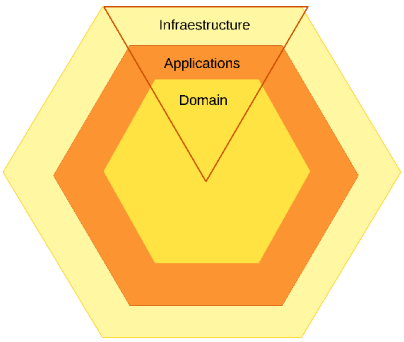

Figura 3. Capas de la arquitectura hexagonal.

Esta supone que el dominio sea el núcleo de todas las capas y que este no se encuentre acoplado a ningún componente externo, en lugar de hacer uso explícito y mediante el principio de inversión de dependencias se llega al acoplamiento de puertos y adaptadores en lugar de implementaciones concretas [16].

La aplicación se puede definir como el conjunto de servicios. Se encarga de coordinar la comunicación entre la capa de dominio y los adaptadores externos, gestionando de esta manera solicitudes y respuestas. Este hace enfoque a los servicios que contiene un microservicio y no tienen dependencia externa con la infraestructura.

La infraestructura hace referencia a los adaptadores que interactúa con el núcleo (dominio) y los puertos, esta capa se comunica con el exterior y gestiona las solicitudes de toda la aplicación, las relacionadas con la base de datos o la interfaz de usuario, la Figura 4 muestra de mejor manera esta conexión.

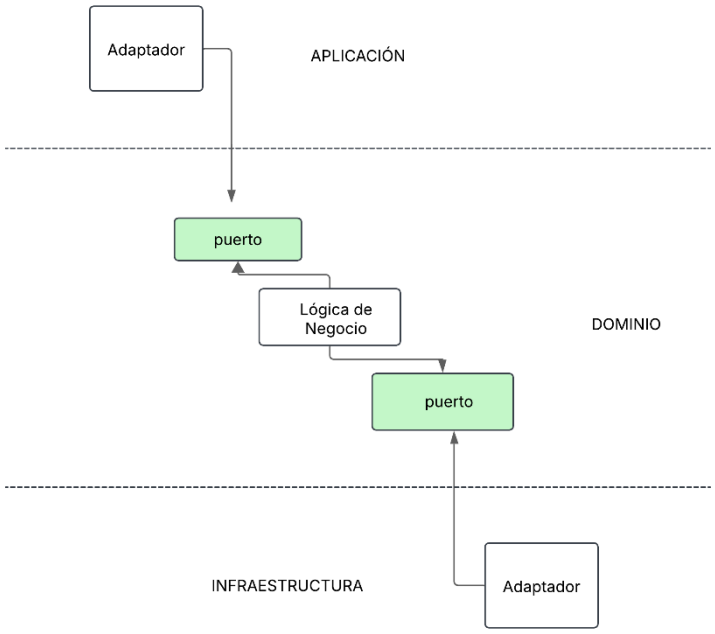

Figura 4. Representación de la arquitectura hexagonal.

Los adaptadores se crean para adecuarse a un punto de entrada muy específico del núcleo de la aplicación a través de un puerto. Un puerto es una especificación de cómo la herramienta puede usar o ser usado por el núcleo de la aplicación [17].

3.5.1. Vertical slicing

El vertical slicing (corte vertical) se orienta al desarrollo de software, en el enfoque de la arquitectura hexagonal, el proceso de desarrollo se lo hace en torno a capas horizontales, que abarca las capas de infraestructura, aplicación y dominio [18]. Esto mejora la detección más ágil de posibles errores, aunque se requiera mayor coordinación además de un diseño adecuado para evitar dependencias entre funcionalidades, este diseño se puede apreciar de mejor manera en la Figura 5.

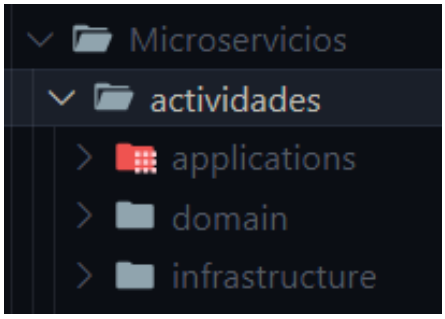

Figura 5. Ejemplo de vertical slicing.

3.5.2. Clean code

Tomando de referencia tesis de licenciatura, Jorge Godínez argumenta respecto a buenas prácticas de desarrollo de software la fusión de técnicas, principios y metodologías que se implementan durante las etapas del software, haciendo énfasis en la forma en la que se estructura el código sosteniendo que: “Aplicando dichas técnicas se está entregando un código limpio, reutilizable y escalable, evitando errores mayores e inconveniente en trabajos colaborativos” [19].

El siguiente listado muestra algunas de las buenas prácticas de programación

- Código sin errores
- Reutilización de código (Evitar duplicidad)
- Buen uso de comentarios
- Seguimiento de estándares tales como nomenclatura y organización de estructuras de proyectos

Esas son algunas de las buenas prácticas de programación, las cuales conllevan al buen dominio y manejo del código, lo que se traduce como un cumplimiento de la métrica de calidad a evaluar.

3.6. APLICACIONES WEB

Es una aplicación que utiliza un navegador web para el proceso de interpretación y ejecución, este tiene un requisito fundamental, el cual es la conexión estable al servicio de internet esto junto al uso del navegador.

Estas aplicaciones se desarrollan utilizando tecnologías web como HTML, CSS y JavaScript, y se alojan en servidores, lo que permite que sean accesibles desde cualquier dispositivo con conexión a internet [20].

3.6.1. Tipos de aplicaciones web

Existen tipos de aplicaciones web que se pueden generalizar, al mismo tiempo que también existen otras que tiene menor relevancia, de esta forma la clasificación se indica a continuación:

- Comercio Electrónico: Utilizadas para la venta de productos de consumo, debiendo ser capaces de gestionar los procesos de compra y pagos. A modo de ejemplo, existen aplicaciones basadas en PretaShop y tiendas en línea como Amazon.
- Marketing y presentación de productos: Su objetivo es la publicidad de forma llamativa de la imagen de una marca o línea de productos. Como ejemplo se tiene a apple.com.
- Sitios de noticias y blogs: Diseñadas para informar con noticias de la actualidad. Como un ejemplo de esta se tiene The Guardian o aplicaciones basadas en WordPress.
- Redes sociales: Teniendo como objetivo interconectar a las personas entre sí, de modo que cada uno de cada uno forma su propio conjunto de contactos de forma independiente. Entre las que más destacan se tienen sitios como Facebook o X, antes llamado Twitter.
- Plataformas de gestión: Específicamente usadas para administrar recursos, empleados, clientes, etc. Como ejemplo se tiene CRM y ERP

3.7. HERRAMIENTAS DE DESARROLLO WEB

3.7.1. Django

Es un framework de desarrollo de alto nivel que se basa en el lenguaje de programación Python. Tiene como principal característica la construcción de aplicaciones web de manera rápida, segura y con menos código.

Se basa en una arquitectura MVC (Model, View, Controller), lo que sirvió como base para crear su propio modelo MVT (Model, View, Template). En la que la capa “Model”, se encuentra la información que se encarga de manipular datos de la aplicación web, la capa “Template” es donde se encuentran las decisiones relacionadas con la presentación. Y finalmente en la capa “View” se encuentra toda la lógica que accede al modelo y encarga al template adecuado para presentar al usuario la respuesta a su petición [21]. En la Figura 6 se puede apreciar de mejor manera lo explicado.

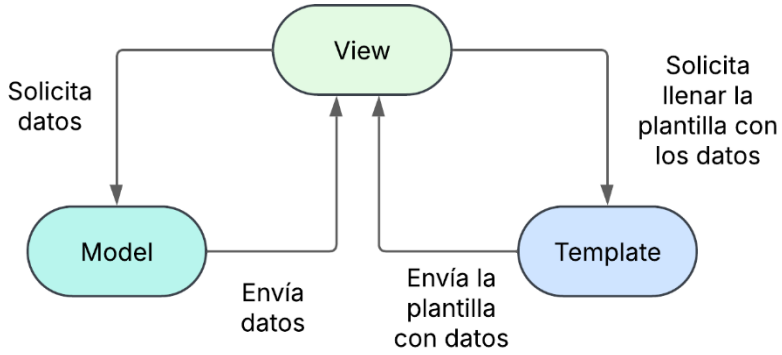

Figura 6. Modelo MVT.

3.7.1.1. Django Rest framework

De manera predeterminada Django no sirve para crear API RESTful, ya que no tiene una forma sencilla de crearla. Debido a esto existe una forma de trabajar con API, llamado Django Rest Framework. Este marco es flexible y adaptado con herramientas para crear API web. Al usar este marco REST de Django, la lógica MVC se mueve a la parte del front-end [22].

Este framework convierte al rol de Django en el de un servidor, es responsable de manejar las solicitudes del cliente para proporcionar datos y modificar datos. Este modelo se explica de mejor manera en la Figura 7.

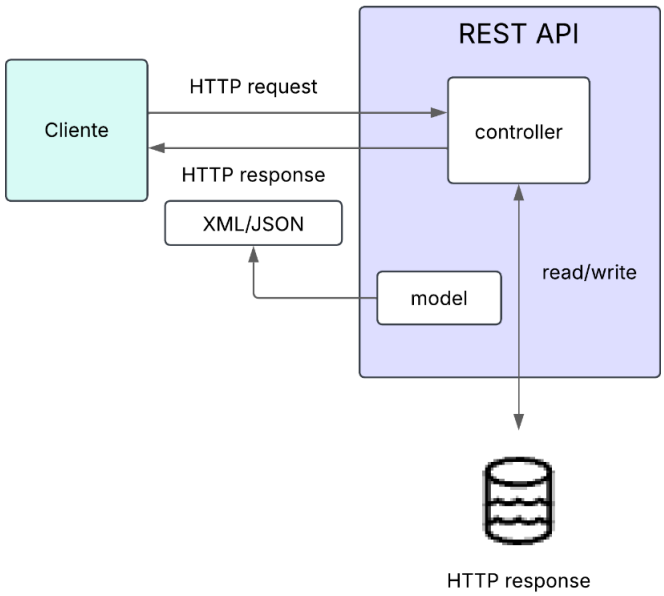

Figura 7. Flujo de datos de API RESTfull.

La vista (el controlador) ya no se representa los datos (el modelo) en la vista, en su lugar los datos son serializados y recuperados por la propia vista. Ahora la vista se encarga del proceso de renderizar.

3.7.2. React

Se trata de una biblioteca para construir interfaces de usuario, lo cual es la parte que se ve la mayor parte del tiempo. Según Jiménez en su libro sobre React, la principal premisa de React es "Solo me preocupo de la Interfaz de Usuario", lo que quiere decir esto es no va a tener routers, models o templates en el caso de que se lo necesite se podrá hacer uso de cualquier otro framework [23].

En términos simples es una librería de JavaScript desarrollada por Facebook, usada para construir interfaces de usuario Single Page Application (SPA) y permite crear componentes reutilizables. Jaime Moreno Canto en su investigación manifestó que "React permite a los desarrolladores crear aplicaciones web de tamaño considerables, que pueden cambiar datos, sin recargar la página, similar a cuando se utiliza la librería AJAX de JavaScript para hacer peticiones a APIs" [24].

3.8. METODOLOGÍAS DE DESARROLLO DE SOFTWARE

El desarrollo de software no es una actividad simple. Como prueba de ello es la existencia de algunas metodologías que intervienen acorde la complejidad del proyecto que se quiere llevar a cabo. Aquí intervienen las propuestas que están enfocadas más al lado tradicional, en especial en lo que respecta al control del proceso, estableciendo las actividades que se llevaran a cabo, las herramientas de las que se hará uso y el producto que se va llevar a cabo.

Una metodología para el desarrollo de software es un manera ordenada y estructurada que permite gestionar y administrar un proyecto, para poderlo desarrollarlo con altas probabilidades de éxito.

Es una forma de trabajo usada para estructurar, planificar y controlar el proceso de desarrollo de sistemas de información. Esta no tiene por qué ser utilizada para todos los proyectos, si no que cada una de las metodologías existentes es útil según el tipo de proyecto correspondiente. Esta entiende los pasos que se van a seguir y aplicar desde que la necesidad surge hasta que se llega al cumplimiento de la misma [25].

3.8.1. Metodologías ágiles

Se refieren al conjunto de principios y prácticas que se utilizan en la gestión de proyectos y desarrollo de software para mejorar la eficiencia y la satisfacción del cliente mediante ciclos de trabajo iterativos y colaborativos. La aparición de las metodologías ágiles, se dieron gracias a las limitaciones que presentaban los métodos tradicionales en proyectos con requisitos actualizables, las cuales comenzaron a presentar precariedad, dificultad en los cambios y sobre exceso de documentación.

El uso de metodologías en la actualidad supone un aumento de la productividad de los trabajadores y un resultado de calidad, además de una mayor comunicación entre los desarrolladores y los usuarios [26] .

3.8.1.1. Metodología XP

Forma parte de las metodologías ágiles, está centrada en enfocar la comunicación como un punto de éxito en el desarrollo de software, fortaleciendo el trabajo en equipo y el avance profesional de los desarrolladores, se basa en la retroalimentación continua entre el usuario y el equipo de trabajo.

XP se distingue de otras metodologías por sus ciclos de desarrollo que ofrecen una retroalimentación continua, además de una planificación y evolución a lo largo de la vida del proyecto, de esta manera demostrando las buenas prácticas utilizadas [27].

El ciclo de vida de esta metodología se compone en cuatro fases, las cuales son:

- **Fase de exploración:** Busca la reunión con todos los interesados y define el alcance que tendrá el proyecto [28].
- **Fase de planificación:** Es una fase corta, en donde los involucrados (cliente, gerentes y desarrolladores) definen las historias de usuario que han de desarrollarse, se procede a establecer una arquitectura inicial.
- **Fase de iteraciones:** Es la fase principal en el ciclo de desarrollo, donde se desarrollan las funcionalidades del sistema con las historias de usuario asignadas a la iteración [28]. Se procede a la separación de historias de usuario en base a dominios y se establece una arquitectura tomando en cuenta puntos clave de mejora.
- **Fase de puesta en producción:** En esta fase se tiene por culminado las historias de usuario en base a la arquitectura y se ha dado por terminado el ciclo de desarrollo de

software, aunque ya no se desarrollen procesos funcionales pueden tener ajustes (“fine tuning”) [28].

3.8.1.1.1. Roles principales

Programador: Es el encargado de realizar el código del sistema y realizar pruebas unitarias.

Cliente: Define los requisitos del sistema mediante historias de usuario y prioriza las tareas y requisitos del proyecto.

Tester: Diseña casos de prueba acorde a historias de usuario además de garantizar que el producto cumpla con su funcionalidad.

Coach: Encargado de implementar prácticas y principios de XP dentro del equipo, supervisando el uso de la metodología.

Tracker: Supervisa el avance del proyecto rastreando cada tarea.

3.8.1.2. Metodología Scrum

El método Scrum se muestra como un enfoque flexible para el desarrollo de software se caracteriza por la capacidad de promover el desarrollo autoorganizado y usar el enfoque de paso por paso [29]. Esto le permite lograr mejor resultados, ya que los equipos pueden adaptarse rápidamente a los cambios y mantener un control constante sobre el proyecto.

La revisión sistemática revela que Scrum facilita la implementación de prácticas maduras, que promueve la productividad y la competitividad, incluso en el contexto global. Además, la correcta dimensión del proyecto, así como la adecuada conformación de los equipos de trabajo son fundamentales para maximizar los beneficios que se obtendrán.

Básicamente, SCRUM es un marco que se centra en la cooperación y la adaptación continua, por lo que es un método efectivo para administrar proyectos de software diferentes y variables.

3.8.1.3. Metodología Kanban

La metodología Kanban descrita es un enfoque suave para la gestión del trabajo que utiliza tableros visuales para organizar tareas en diferentes etapas, como "retrasos", "suceder" y "claro" [30]. Su origen se puede encontrar en el sistema de producción de Toyota y se basa en principios Lean, destinado a aumentar la eficiencia y reducir los desechos. Kanban se caracteriza por limitar el trabajo en curso al promover la cooperación entre los miembros del equipo y

garantizar una clara visualización del flujo de trabajo, facilitando su adaptación a varios contextos de desarrollo de software y lenguajes de programación [30].

Sin embargo, también se reconocen desafíos, como la necesidad de una buena coordinación del equipo y la identificación de cuellos de botella, que pueden afectar el rendimiento del método si no se abordan adecuadamente [30].

Para comprender de mejor manera se tiene la Tabla 3, que muestra un resumen de las principales características de las metodologías abordadas.

4. MÉTODOS Y PROCEDIMIENTOS

4.1. TIPOS DE INVESTIGACIÓN

4.1.1. Investigación bibliográfica

Por una parte, se tiene la investigación bibliográfica, esta constara en la búsqueda de información de fuentes certificadas, entre estos documentos se incluyen, informes, trabajos de titulación etc.

Con esto dicho la investigación bibliográfica permitirá construir una base teórica solida al revisar la literatura existente sobre métricas de dependencia y complejidad, de esta manera lograr proporcionar una visión detallada del estado actual de medición de estas métricas, así como también sobre microservicios y arquitectura hexagonal,

Este enfoque teórico permitirá fundamentar el marco conceptual del proyecto, garantizando de esta manera que las decisiones de diseño y desarrollo estén respaldadas por evidencias validadas.

4.1.2. Investigación descriptiva

Mientras que por otro lado en lo que consta el ámbito descriptivo, se describirá características, no busca explicar el por qué sucede algo, sino más bien proporcionar una imagen de lo que se está investigando. Permitirá detallar como se aplican las métricas de dependencia y complejidad en el contexto de arquitectura de microservicios, de esta manera logrando complementar la investigación bibliográfica a una visión práctica del tema de estudio, además de permitir comparar los resultados obtenidos de las métricas de dependencia y complejidad en arquitecturas de microservicios.

4.1.3. Investigación Tecnológica

Por último, la investigación tecnológica permitirá indagar e identificar la manera en que se puede aplicar la arquitectura evaluada, y también ayudar en la selección de la herramienta que permita su implementación.

4.2. MÉTODOS DE INVESTIGACIÓN

4.2.1. Método hipotético

En cuanto al método de investigación del proyecto, este será un hipotético, el cual es crucial y útil cuando se quiere explorar relaciones causales y comprobar teorías mediante la formulación y prueba de hipótesis, permitirá investigar las relaciones de causa, como el impacto de la evaluación de métricas de calidad en las aplicaciones basadas en microservicios. De modo que si ya existen teorías o modelos propuestos sobre como esta arquitectura influyen en las métricas de calidad el método hipotético permitirá refutar estas ideas a través de pruebas empíricas. Este método no solo permite validar teorías existentes, sino también generar nuevo conocimiento.

La hipótesis planteada dice “La evaluación de métricas de dependencia y complejidad en microservicios mejorará su arquitectura y así poder aplicarla de un prototipo de aplicación web”. Esta hipótesis podrá ser probada mediante la evaluación de métricas de calidad y el estudio de casos donde se han implementado estas arquitecturas. De forma que, esto no solo fortalece la investigación, sino que además aporta un valor a la comunidad científica y profesional.

4.3. TÉCNICAS DE INVESTIGACIÓN

4.3.1. Análisis bibliográfico

Técnica fundamental en investigación que consiste en la revisión bibliográfica de literatura existente, que servirá para la contextualización del problema de estudio.

4.3.2. Análisis de resultados

Hace referencia al procesamiento e interpretación de la información obtenida, la cual es parte del análisis de la investigación.

4.4. INSTRUMENTOS DE INVESTIGACIÓN

4.4.1. Ficha bibliográfica

Será usado para recopilar información clave de fuentes (libros, artículos, páginas web) para poder sistematizar la revisión bibliográfica.

4.4.2. Gráficas de comparación

Presentará resultados del análisis cuantitativo y cualitativo de manera visual, para comparar los resultados de las dos arquitecturas.

4.4.3. Iteraciones

Será de utilidad para desarrollar el prototipo, ya que gracias a ella se podrá aplicar las mejoras propuestas progresivamente.

4.5. TÉCNICA DE DESARROLLO DE SOFTWARE

4.5.1. Diseño basado en dominios

Será usada para realizar la separación de servicios, con el fin de obtener una arquitectura en base a las recomendaciones del software usado.

4.5.2. Arquitectura Hexagonal

Se usará esta arquitectura para el desarrollo del prototipo web, con el fin de organizarlo y tener control de las diferentes capas que esta ofrece.

4.5.3. Metodología de desarrollo

En base a los objetivos a cumplir, se debe llevar a cabo la utilización de una metodología de software, para esto se tiene la Tabla 3, la cual otorga una comparación de las características de las metodologías más usadas, la cual, una vez realizado en análisis y retrospectiva adecuada, se tiene que XP es más favorable para el desarrollo de microservicios.

Tabla 3. Comparativa de metodologías ágiles

Criterio	Extreme Programming (XP)	Scrum	Kanban
Manejo de la complejidad	Refactorización constante y diseño evolutivo reducen la complejidad del código.	Complejidad controlada a nivel de planificación y alcance del sprint.	Se gestiona reduciendo el trabajo en progreso, pero sin prácticas específicas para reducir la complejidad
Modularidad	Pair programming favorecen código modular.	Depende más de la estrategia del equipo y las historias de usuario bien definidas.	La modularidad depende de cómo se diseñen las tareas
Gestión de dependencias	Se minimizan mediante integración continua.	Puede haber más riesgo de dependencias si las historias no se diseñan con independencia.	No impone reglas estrictas para manejar dependencias, lo que puede generar acoplamiento
Impacto en métricas de complejidad	Código más simple y mantenible, reduciendo métricas como ciclomática y acoplamiento.	Complejidad puede acumularse si no se refactoriza con frecuencia.	No tiene un enfoque específico en refactorización, por lo que la complejidad puede aumentar
Impacto en métricas de dependencia	Microservicios diseñados con menor dependencia, favoreciendo independencia funcional.	Dependencias pueden surgir si las historias abarcan múltiples microservicios	Riesgo de alta dependencia si las tareas no se organizan de manera adecuada.

Esto debido a los siguiente puntos, teniendo en cuenta que la evaluación de la complejidad y la dependencia como métricas en las metodologías de XP, Scrum y Kanban tienen efectos diferenciados.

XP, con su enfoque de integración continua y la retroalimentación constante, tienden a crear arquitectura con menos complejidad y menor dependencia entre los microservicios.

Por otro lado, Scrum, al centrarse en la entrega de funcionalidades, puede conducir a una mayor complejidad si las historias de los usuarios no consideran la modularidad del sistema que se desarrolló. Para el grupo de investigación, si el sprint no incluye un espacio para la reflexión, la dependencia entre los microservicios puede aumentar con el tiempo.

Por el lado de Kanban, esta no impone iteraciones fijas ni roles específicos, lo que facilita la adaptación a cambios constantes. Sin embargo, para la investigación al no tener reglas estrictas

sobre diseño y evolución del sistema, las dependencias pueden crecer sin control y la complejidad puede aumentar progresivamente.

Una vez enfatizado todo esto se tiene que, XP proporciona un entorno más apropiado para mantener la arquitectura de microservicios con menos complejidad y menos adicción, además proporciona el pair programing que fomenta el desarrollo en parejas.

4.5.4. Metodología XP

Teniendo en cuenta el análisis anterior, el desarrollo de este proyecto será bajo la metodología XP, la cual brindará la ventaja de trabajo en pares antes mencionada, mejorando la comunicación dentro del equipo y asegurando que todos los miembros estén alineados con los objetivos del proyecto.

Esta metodología es particularmente adecuada para el desarrollo del prototipo basado en microservicios, entre las razones por las cuales se optó por esta recae que, se podrán implementar el uso de historias de usuario que serán fundamentales para la elaboración de la arquitectura, gracias a su fase de exploración y planificación. Además de poder realizar el proceso de evaluación de las métricas de la arquitectura obtenidas en su fase de iteración, tanto de MOAM y la del grupo de investigación, reduciendo la dependencia y la complejidad, dando paso a la elaboración del prototipo basado en microservicios en la fase de implementación.

Aplicará la práctica retrospectiva frecuente, en donde el equipo evalúa su proceso de trabajo en busca de mejoras, esto con el propósito de mantener la eficacia y calidad de la arquitectura mediante las métricas de dependencia y complejidad lo cual será útil para realizar ajustes.

4.5.4.1. Fase de exploración

Aquí se establecer además las métricas que se evaluaran en la arquitectura, así como su importancia, con el fin de mejorarlas.

Se evalúa los resultados de las métricas obtenidos por el segundo componente de MOAM, en el cual interviene cálculos de dependencia y complejidad.

4.5.4.1.1. Métricas de dependencia

Se tiene como primer punto métricas enfocadas en dependencia, estas miden la confiabilidad de las diferentes relaciones entre los microservicios dentro de la arquitectura propuesta.

Número de dependencias externas

Estas se refieren a servicios externos, es decir ajenos a los que el grupo de investigación uso dentro del proyecto, estas se complementan con las dependencias internas. Los distintos rangos de aceptación se encuentran detallados en la Tabla 4.

Tabla 4. Rangos de puntuación de dependencias externas.

Rango	**Interpretación número de dependencias externas**
(0-2)	Bajas dependencias externas
(3-5)	Dependencias externas moderadas
(6+)	Alta dependencia externa

Número de dependencias internas

Es la cantidad de dependencias que los servicios tienen con los demás dentro del mismo sistema. Una de las partes más esenciales en este proceso de evaluación, ya que el número de dependencias internas afectara de cierta forma otras métricas de la arquitectura. Los distintos rangos de aceptación se encuentran detallados en la Tabla 5.

Tabla 5. Rangos de puntuación de dependencias internas.

Rango	**Interpretación número de dependencias externas**
(0-5)	Bajas dependencias internas
(6-10)	Dependencias internas moderadas
(11+)	Alta dependencia interna

Profundidad de las dependencias

Se mide la profundidad entre microservicios, es decir la longitud de niveles que existe para realizar una llamada a un servicio, en la que un número pequeño de niveles es lo más óptimo. Los distintos rangos de aceptación se encuentran detallados en la Tabla 6.

Tabla 6. Rangos de puntuación de profundidad de dependencias.

Rango	**Interpretación profundidad de dependencias**
(0-2)	Bajas profundidad
(3-5)	Profundidad moderada
(6+)	Alta profundidad de dependencias

Acoplamiento estimado

Toma en cuenta el número de dependencias internas en la arquitectura propuesta y la relación en el número máximo de conexiones. Los distintos rangos de aceptación se encuentran detallados en la Tabla 7.

Tabla 7. Rangos de puntuación de acoplamiento estimado.

Rango	**Interpretación acoplamiento estimado**
(0-3)	Bajo acoplamiento
(4–6)	Acoplamiento moderado
(7+):	Alto acoplamiento

Falta de cohesión estimada

Se enfoca en el número de responsabilidades que tiene un microservicio. Los distintos rangos de aceptación se encuentran detallados en la Tabla 8.

Tabla 8. Rangos de puntuación de falta de cohesión.

Rango	Interpretación falta de cohesión
(0.5 - 1.0)	Baja falta de cohesión
(1.1 - 2.0)	Falta de cohesión moderada
(2.1+)	Falta de cohesión alta

Dependencia estimada

Es un estimado del total de métricas de dependencia, para evaluar la complejidad total de un microservicio. Los distintos rangos de aceptación se encuentran detallados en la Tabla 9.

Tabla 9. Rangos de puntuación de dependencias estimada.

Rango	Interpretación dependencia estimada
(5 - 8)	Bajas dependencias internas
(9 - 12)	Dependencias internas moderadas
(13+)	Alta dependencia interna

4.5.4.1.2. Métricas de complejidad

Servirán para identificar las partes más complejas en la arquitectura propuesta.

Complejidad ciclomática estimada

Servirá para determinar la complejidad acorde al número de caminos posibles, es decir ayudará para evaluar la dificultad de la arquitectura en base a sus dependencias internas. Los distintos rangos de aceptación se encuentran detallados en la Tabla 10.

Tabla 10. Rango de puntuación de complejidad ciclomática.

Rango	Interpretación complejidad ciclomática
(3 - 5)	Complejidad ciclomática baja
(6 - 8)	Complejidad ciclomática moderada
(9+)	Complejidad ciclomática alta

Coeficiente de Acoplamiento estimado

Es un indicador del grado de acoplamiento entre microservicios tomando en cuenta toda la arquitectura para evaluar su modularidad. Los distintos rangos de aceptación se encuentran detallados en la Tabla 11.

Tabla 11. Rangos de puntuación de coeficiente de acoplamiento.

Rango	Interpretación coeficiente de acoplamiento
(0 – 1)	Coeficiente de acoplamiento bajo
(1.1 – 2.0)	Coeficiente de acoplamiento moderado
(2.1+)	Coeficiente de acoplamiento alto

Complejidad estructural estimada

Servirá para determinar la complejidad de toda la arquitectura combinando el número de microservicios y sus dependencias internas. Los distintos rangos de aceptación se encuentran detallados en la Tabla 12.

Tabla 12. Rangos de puntuación de complejidad estructural.

Rango	Interpretación complejidad estructural
(10 – 15)	Complejidad estructural baja
(16 - 20)	Complejidad estructural moderada
(21+)	Complejidad estructural alta

Puntos de Historia Estimados

Son el número total de funcionalidades que se deberán cumplir, para estimar el esfuerzo. Su número depende de los requisitos que debe tener el sistema.

Complejidad estimada

Es la complejidad total de la arquitectura, tomando en cuenta todas las métricas de complejidad, para tener una medida global de la arquitectura. Los distintos rangos de aceptación se encuentran detallados en la Tabla 13.

Tabla 13. Rangos de puntuación de complejidad estimada.

Rango	Interpretación complejidad estimada
(30 – 45)	Complejidad baja
(46 - 60)	Complejidad moderada
(61+)	Complejidad alta

Índice Agregado Estimado

Resume la calidad general obtenida en la arquitectura, ayuda a dar una idea de la salud de la misma. Los distintos rangos de aceptación se encuentran detallados en la Tabla 14.

Tabla 14. Rangos de puntuación de índice agregado.

Rango	Interpretación de índice agregado
(40 – 55)	Índice bajo
(56 - 70)	Índice moderado
(71+)	Índice alto

4.5.4.2. Fase de planificación

Poniendo en práctica la fase de exploración, se recopilo historias de usuario obtenidas en [31], especificadas en la Tabla 5. Estas han de utilizarse para la elaboración de la arquitectura, estableciendo una propuesta inicial, haciendo uso del primer componente del software MOAM, el cual en base a los requerimientos establecido otorga una opción inicial que está abierta a mejoras, las cuales serán dependiendo la elección del desarrollador.

4.5.4.3. Fase de iteración

Se establecen las mejoras tomando en cuenta las falencias encontradas en la primera arquitectura, buscando la forma en la que se pueden separarse, procurando que guarde sentido de funcionalidad, esto haciendo uso de DDD.

Se procederá con la medición de las métricas de calidad de la arquitectura como también la comparativa individual de los distintos resultados, explicando el porqué de cada resultado.

4.5.4.4. Fase de puesta a producción

Se establecerá el proceso de puesta a prueba de la arquitectura que una vez medidas y aprobadas las métricas correspondientes, será desarrollado el prototipo con el finde poder comprobar la hipótesis,

4.5.4.4.1. Arquitectura hexagonal

La estructura interna del código se regirá bajo una arquitectura hexagonal, muy usada para el desarrollo de software. Promoverá la separación entre la lógica de negocios y dependencias externas tales como interfaces de usuario y sistemas de bases de datos, mejorando el entendimiento y mantenimiento.

Al usar los puertos y adaptadores se permite reemplazar componentes sin afectar la lógica de negocio. Permitiendo así desarrollar y desplegar funcionalidades, además de desplegar independientemente cada microservicio, lo cual es útil en el equipo de investigación ya que se trabajará bajo la arquitectura XP

Permitirá la inclusión de nuevas tecnologías sin reescribir la lógica de negocios, dando paso a la escalabilidad independiente, dejando cualquier fallo de un módulo separado de los demás.

5. ANÁLISIS Y DISCUSIÓN DE LOS RESULTADOS

5.1. SEGUIMIENTO DE LA METODOLOGÍA DE DESARROLLO

5.1.1. Fase de exploración

5.1.1.1. Caso de estudio: Camperus un prototipo basado en microservicios

Camperus es un sistema de información que sirve para gestionar y optimizar los procesos dentro de un campamento, puede ser usado por campamentos dentro del Ecuador. Camperus permite crear y gestionar padres o trabajadores, crear grupos de campamento y asignar campistas a esos grupos, asignar actividades a grupos de campamento además de la opción de enviar notificaciones a destinatarios registrados en el campamento.

El desarrollo de Camperus parte desde la definición de historias de usuario, las cuales son el punto inicial de cualquier proceso de desarrollo de software. Al momento de comenzar con el desarrollo, Camperus no era un sistema existente como tal, por lo que toda la aplicación con sus microservicios, así como su arquitectura deberán ser desarrollados desde cero.

5.1.1.2. Métricas de calidad a evaluar

Se establecen las métricas de calidad que se evaluaran en l arquitectura.

- Número de dependencias externas (ND)
- Número de dependencias internas (NI)

- Profundidad de las dependencias (PD)
- Acoplamiento Estimado (AC_est)
- Falta de Cohesión (LCOH_est)
- Dependencia estimada (D_est)
- Complejidad ciclomática estimada (CC_est)
- Coeficiente de Acoplamiento estimado (CA_est)
- Complejidad Estructural Estimada (CE_est)
- Puntos de Historia Estimados (PH_est)
- Complejidad Estimada (C_est)
- Índice Agregado Estimado (AI_est)

5.1.2. Fase de planificación

5.1.2.1. Establecimiento de las historias de usuario

Una vez contextualizado lo que se espera conseguir, se procede con el establecimiento de historias de usuario que se van a realizar, las cuales se especifican en la tabla 15.

Tabla 15. Historias de Usuario para Camperplus.

Código de la historia	Descripción de la historia de usuario
H01	Como administrador del campamento, quiero poder agregar campistas para poder realizar un seguimiento de cada uno de ellos individualmente.
H02	Como administrador del campamento, quiero poder eliminar a los campistas de la base de datos para poder mantener mis páginas ordenadas.
H03	Como administrador del campamento, quiero poder programar actividades para los campistas, para que los trabajadores del campamento puedan realizar un seguimiento fácil de quién está dónde en todo momento.
H04	Como administrador del campamento, quiero poder suspender a un campista que tenga problemas de conducta.
H05	Como administrador del campamento, quiero poder asignar diferentes puestos a los miembros del personal, de modo que sea posible organizar las actividades en términos de responsabilidades.

Código de la historia	Descripción de la historia de usuario
H06	Como administrador del campamento, quiero poder advertir a un trabajador del campamento que creo que hizo su trabajo de manera inapropiada.
H07	Como administrador del campamento, quiero poder crear y modificar reglas que los campistas y trabajadores del campamento deben seguir.
H08	Como administrador del campamento, quiero almacenar la información de los padres o tutores inmediatos del campista, para poder llamarlos fácilmente y notificarlos en caso de que se produzca un comportamiento extremadamente inaceptable.
H09	Como administrador del campamento, quiero poder agregar padres para que puedan inscribir a sus hijos en el campamento.
H10	Como administrador del campamento, quiero poder crear grupos y agregar campistas a los grupos, para poder organizarlos fácilmente.
H11	Como administrador del campamento, quiero poder ver todos mis grupos de campamento y las actividades programados para cada grupo de campamento, para poder notificar al consejero lo que hará su grupo durante el día.
H12	Como administrador del campamento, quiero poder programar tareas/actividades para un grupo de campamento específico, de modo que pueda realizar un seguimiento de las actividades diarias, semanales y mensuales de cada grupo.
H13	Como administrador del campamento, quiero poder eliminar actividades que programé, para poder mantener el cronograma limpio de actividades innecesarias.
H14	Como administrador del campamento, quiero poder modificar las actividades que programé en caso de que haya un cambio, para poder mantener siempre el cronograma actualizado.
H15	Como administrador del campamento, quiero poder modificar la información de los campistas inscritos, para poder mantener la información de los campistas actualizada.

Código de la historia	Descripción de la historia de usuario
H16	Como administrador del campamento, quiero poder modificar la información de los padres agregados, para poder mantener la información de los padres actualizada.
H17	Como administrador del campamento, quiero poder eliminar a los padres de la base de datos, para poder mantener mis páginas libres de desorden innecesario.
H18	Como administrador del campamento, quiero poder modificar la información de un grupo, para poder mantenerlos actualizados.
H19	Como padre, quiero poder realizar un seguimiento de la actividad y el horario de mi hijo en el campamento, para poder tener tranquilidad.
H20	Como padre, quiero poder crear una cuenta para poder inscribir a mis hijos en el campamento en línea.
H21	Como padre, quiero ver qué trabajadores están asignados a mis hijos, para poder tener tranquilidad.
H22	Como padre, quiero poder enviar mensajes a los trabajadores de mi hijo para poder expresar mis inquietudes o verificar el progreso de mi hijo.
H23	Como padre, quiero poder inscribir a mis hijos para que puedan ser admitidos en el campamento.
H24	Como padre, quiero poder ver y editar a mis hijos inscritos en el campamento, para poder saber a quién ya inscribí en el campamento, quién todavía está pendiente de admisión.
H25	Como padre, quiero poder ver un calendario de las actividades en las que participan mis hijos en el campamento, para poder estar más informado sobre lo que están haciendo en el campamento.
H26	Como trabajador del campamento, quiero poder ver qué campistas tengo a mi cargo y dónde están, para poder asegurarme de que estoy haciendo mi trabajo correctamente.
H27	Como trabajador del campamento, puedo denunciar al administrador sobre un campista que tenga un comportamiento inapropiado.

Código de la historia	Descripción de la historia de usuario
H28	Como trabajador del campamento, puedo informar al administrador sobre una lista de suministros de los que hay escasez en el campamento.
H29	Como trabajador del campamento, puedo informar al administrador sobre las reparaciones necesarias que el campamento necesita.

Para iniciar con el desarrollo de las demás fases de la metodología, es necesario establecer responsabilidades, las cuales se detallan en el Anexo A.

5.1.2.2. Evaluación de la primera propuesta de arquitectura otorgada por MOAM

5.1.2.2.1. Arquitectura por el primer módulo de MOAM

En base a las historias de usuario planteadas, se buscó obtener una idea inicial de la estructura, para lo cual se hizo de un software MOAM, con el cual se obtuvo la arquitectura del sistema dada por el primer componente, el cual se encarga de analizar las historias de usuario, con el fin de agruparlas en base a dominios funcionales. Usando algoritmos de agrupamiento, se obtuvo la gráfica de la arquitectura, como se aprecia en la Figura 8.

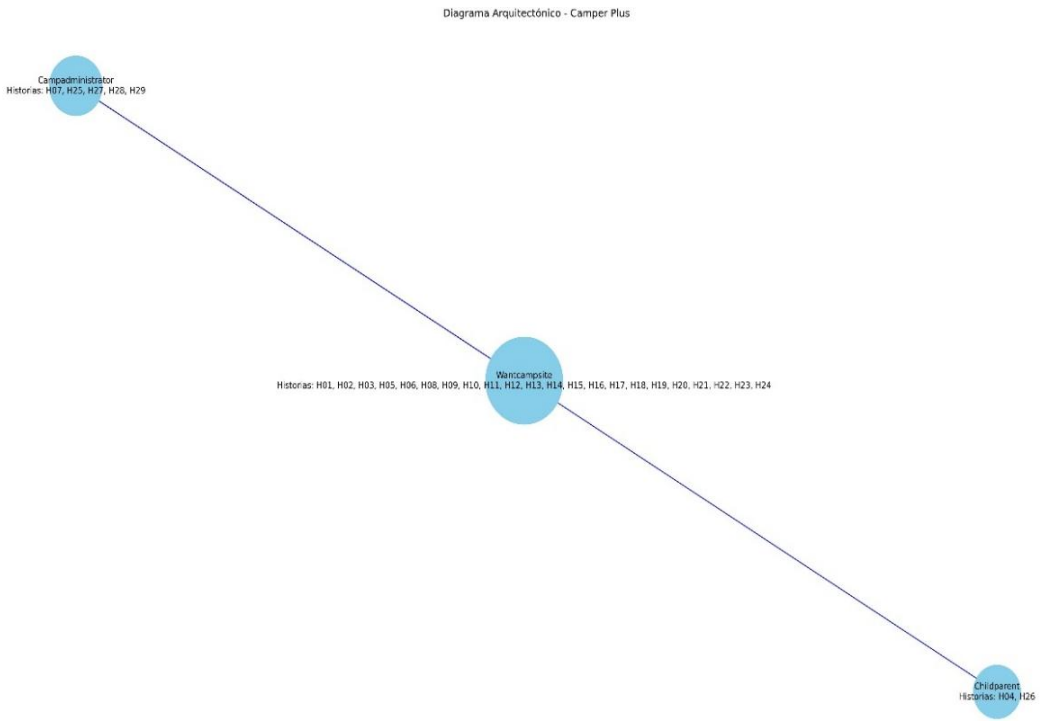

Figura 8. Arquitectura de microservicios por MOAM.

Junto a esta arquitectura, se generó un archivo, el cual detalla la arquitectura en lenguaje JSON, tomando puntos importantes que serán usados para la medición de métricas de calidad en el segundo componente de MOAM. Dicho archivo tiene una estructura similar a la detallada en la Figura 9.

Figura 9. Ejemplo de arquitectura en formato JSON.

En esta se especifica el nombre del servicio, su dominio y la historia de usuario englobada, pero lo más importante es el número de dependencias internas y externas, que son una de las bases para el cálculo de las métricas de dependencia y complejidad de la arquitectura.

5.1.2.2.2. Métricas por el segundo módulo de MOAM

Una vez obtenido la gráfica de la arquitectura, se la evalúa mediante el segundo componente de MOAM, el cual recepta el archivo JSON y procede con la evaluación de las diferentes métricas de calidad. Como resultado se tiene la Figura 10, esta detalla las métricas de calidad obtenidas.

Figura 10. Métricas de calidad otorgadas por MOAM.

Todos los resultados de este primer proceso serán utilizados la mejora y la posterior comparación frente a una arquitectura diseñada por el grupo de investigación tomando en cuenta las recomendaciones basadas en este proceso.

5.1.3. Fase de iteración

5.1.3.1. Desarrollo de la arquitectura propuesta por el grupo de investigación

Para la obtención de la arquitectura propuesta por el grupo de investigación, en base a las recomendaciones, se buscada una nueva forma de separar los servicios e historias, para lo cual se procede con los siguientes procesos DDD.

5.1.3.1.1. Diseño basado en dominios

Análisis de dominio (Analyze domain)

Se procede a la separación en el contexto de funcionalidades que se cumplirán para satisfacer las necesidades de las historias.

- **Administrador del campamento:** Responsable de gestionar a los campistas, actividades, reglas, grupos, trabajadores y notificaciones.
- **Padres o tutores legales:** Encargados de registrar y supervisar a los campistas, así como de comunicarse con los trabajadores.
- **Trabajadores del campamento:** Supervisan a los campistas, gestionan actividades y reportan problemas o necesidades al administrador.
- **Gestión de campistas:** Incluye la inscripción, eliminación, modificación y suspensión de campistas.
- **Gestión de actividades:** Programación, modificación y eliminación de actividades grupales.
- **Gestión de reglas:** Creación y actualización de normas para el campamento.
- **Comunicación:** Proceso de notificaciones y mensajes entre administradores, padres y trabajadores.
- **Gestión de padres o tutores:** Manejo de información personal de los tutores legales.

Contextos delimitados (Bounded Contexts)

Con base en el análisis anterior, se definieron los siguientes contextos delimitados, cada uno con responsabilidades claras que facilitarán su implementación en microservicios.

- **Tutor:** Manejo de la información de padres o tutores.

Funciones: Almacenamiento, modificación y eliminación de datos.

- **Auth:** Gestión de autenticación y autorización.

Funciones: Registro de usuarios y manejo de permisos.

- **Reglas:** Administración de las normas del campamento.

Funciones: Creación y actualización de reglas.

- **Trabajadores:** Supervisión de trabajadores y roles asignados.

Funciones: Gestión de roles, advertencias y reportes.

- **Campamento:** Gestión de los grupos del campamento.

Funciones: Creación, modificación y eliminación de grupos.

- **Actividades:** Gestión de la programación de actividades.

Funciones: Creación, modificación y eliminación de actividades grupales.

- **Campistas:** Gestión de información y estado de los campistas.

Funciones: Inscripción, modificación, eliminación y suspensión.

- **Notificaciones:** Comunicación entre actores del sistema.

Funciones: Notificaciones y mensajes.

Definición de entidades, agregadas y servicios

En cada contexto delimitado, se identifican las entidades principales, sus relaciones y los servicios necesarios, los cuales se representan en la serie de tablas.

Tabla 16. Entidades y servicios de Auth.

Contexto:	**Auth**
Entidades:	• Auth: Representa el registro y validación sesión
Servicios:	• Registrar usuarios • Validar la sesión

Tabla 17. Entidades y servicios de Campistas.

Contexto:	**Campistas**
Entidades:	• Campista: Representa a los niños inscritos en el campamento. • Grupo: Agrupa a campistas
Servicios:	• Registrar campistas. • **Modificar** información de campistas. • Suspender campistas en caso de incumplimiento de normas

Tabla 18. Entidades y servicios de Campamento.

Contexto:	**Campamento**
Entidades:	• Regla: Representa a los grupos que se crean en el campamento • Tutor: Relación con el campamento por los campistas registrados • Trabajador: Relación con el grupo asignado
Servicios:	• Crear nuevos grupos • Modificar grupos existentes • Eliminar grupos no necesarios

Tabla 19. Entidades y servicios de Actividades.

Contexto:	**Actividades**
Entidades:	• Actividad: Representa cada actividad programada en el campamento. • Grupo: Relación entre actividades y los grupos que participan en ellas.
Servicios:	• Crear nuevas actividades. • Modificar actividades existentes. • Eliminar actividades no necesarias.

Tabla 20. Entidades y servicios de Notificaciones.

Contexto:	**Notificaciones**
Entidades:	• Notificación: Mensaje enviado entre actores del sistema. • Auth: Relacionado con el redireccionamiento de las notificaciones
Servicios:	• Enviar notificaciones a padres o trabajadores.

Tabla 21. Entidades y servicios de Trabajador.

Contexto:	**Trabajador**
Entidades:	• Trabajador: Representa a cada trabajador del campamento. Incluye detalles como nombre, rol, contacto y grupo asignado. • Grupo: Relación entre trabajadores y los grupos a los que están asignados. Cada trabajador puede estar asignado a un grupo.
Servicios:	• Registrar nuevos trabajadores • Modificar registros de trabajadores existentes • Eliminar registros de trabajadores no necesarios

Tabla 22. Entidades y servicios de Tutor.

Contexto:	**Tutor**
Entidades:	• Tutor: Representa a cada tutor de los campistas. Incluye detalles como nombre, contacto y relación con el campista. • Campamento: Relación entre tutores y el campamento
Servicios:	• Registrar nuevos tutores • Modificar registros de tutores existentes • Eliminar registros de tutores no necesarios

Tabla 23. Entidades y servicios de Reglas.

Contexto:	**Reglas**
Entidades:	• Regla: Representa cada regla del campamento. Incluye detalles como descripción, penalización y grupo al que aplica.
Servicios:	• Crear nuevas reglas • Modificar reglas existentes • Eliminar reglas no necesarias:

Identificación de los microservicios

Microservicios propuestos

- **Tutor Microservicio:**
  Gestión de información de padres y tutores.
  Funciones: CRUD de tutores.
- **Auth Microservicio:**
  Manejo de autenticación y permisos de acceso.
  Funciones: Registro, inicio de sesión y autorización.
- **Reglas Microservicio:**
  Administración de las normas del campamento.
  Funciones: Crear y modificar reglas.
- **Trabajadores Microservicio:**
  Gestión de trabajadores y sus reportes.
  Funciones: Asignación de roles, reportes de incidentes, CRUD trabajadores.
- **Campamento Microservicio:**
  Manejo de grupos.
  Funciones: CRUD Grupos, asignar grupos a campistas.
- **Actividades Microservicio:**
  Control de actividades grupales.
  Funciones: Crear, modificar y eliminar actividades.
- **Campistas Microservicio:**
  Gestión completa de los datos de los campistas.
  Funciones: Inscripción, modificación, suspensión.
- **Notificaciones Microservicio:**
  Facilita la comunicación entre usuarios.
  Funciones: Envío y recepción de notificaciones.

5.1.3.1.2. Definición de historias por cada microservicio

En este apartado, teniendo en cuenta el punto anterior se obtiene las historias de usuario separadas por dominios funcionales, haciendo uso de los pasos DDD, las cuales se pueden apreciar en las siguientes tablas.

Tabla 24. Historias de usuario del microservicio Campamento.

Microservicio	Número de historia	Descripción
Campamento	H10	Como administrador del campamento, quiero poder crear grupos y agregar campistas a los grupos, para organizarlos fácilmente.
	H11	Como administrador del campamento, quiero poder ver todos mis grupos de campamento y los eventos programados para cada grupo de campamento.
	H18	Como administrador del campamento, quiero poder modificar la información de un grupo, para mantenerlos actualizados.
	H21	Como padre, quiero ver qué consejeros están asignados a mis hijos, para tener tranquilidad.

Tabla 25. Historias de usuario del microservicio Campista.

Microservicio	Número de historia	Descripción
Campista	H01	Como administrador del campamento, quiero poder agregar campistas para poder realizar un seguimiento de cada uno de ellos individualmente.
	H02	Como administrador del campamento, quiero poder eliminar a los campistas de la base de datos para mantener mis páginas ordenadas.
	H04	Como administrador del campamento, quiero poder suspender a un campista que tenga problemas de conducta.
	H15	Como administrador del campamento, quiero poder modificar la información de los campistas inscritos.
	H23	Como padre, quiero poder inscribir a mis hijos para que puedan ser admitidos en el campamento.
	H24	Como padre, quiero poder ver y editar a mis hijos inscritos para el año de campamento, para poder saber a quién ya inscribí en el campamento, quién todavía está pendiente de admisión, etc.

Tabla 26. Historias de usuario del microservicio Actividades.

Microservicio	Número de historia	Descripción
Actividades	H03	Como administrador del campamento, quiero poder programar actividades para los campistas.
	H12	Como administrador del campamento, quiero poder programar actividades para un grupo de campamento específico.
	H13	Como administrador del campamento, quiero poder eliminar actividades para mantener el cronograma limpio.
	H14	Como administrador del campamento, quiero poder modificar actividades para mantener el cronograma actualizado.
	H19	Como padre, quiero poder realizar un seguimiento de la actividad y el horario de mi hijo en el campamento, para poder tener tranquilidad.
	H25	Como padre, quiero poder ver un calendario de las actividades en las que participan mis hijos en el campamento, para poder estar más informado sobre lo que están haciendo en el campamento.

Tabla 27. Historias de usuario del microservicio Tutor.

Microservicio	Número de historia	Descripción
Tutor	H08	Como administrador del campamento, quiero almacenar la información de los padres o tutores de los campistas.
	H09	Como administrador del campamento, quiero poder agregar padres para que inscriban a sus hijos.
	H16	Como administrador del campamento, quiero poder modificar la información de los padres agregados.
	H17	Como administrador del campamento, quiero poder eliminar a los padres de la base de datos.

Tabla 28. Historias de usuario del microservicio Auth.

Microservicio	Número de historia	Descripción
Auth	H20	Como padre, quiero poder crear una cuenta para poder inscribir a mis hijos en el campamento en línea.

Tabla 29. Historias de usuario del microservicio Trabajador.

Microservicio	Número de historia	Descripción
Trabajador	H05	Como administrador del campamento, quiero poder asignar diferentes puestos a los miembros del personal.
	H26	Como trabajador del campamento, quiero poder ver qué campistas tengo a mi cargo y dónde están.

Tabla 30. Historias de usuario del microservicio Regla.

Microservicio	Número de historia	Descripción
Regla	H07	Como administrador del campamento, quiero poder crear y modificar reglas que los campistas y trabajadores deben seguir.

Tabla 31. Historias de usuario del microservicio Notificación.

Microservicio	Número de historia	Descripción
Notificación	H06	Como administrador del campamento, quiero poder advertir a un trabajador del campamento por su trabajo inapropiado.
	H22	Como padre, quiero poder enviar mensajes a los consejeros de mi hijo para expresar mis inquietudes.
	H27	Como trabajador del campamento, puedo denunciar al padre sobre un campista con comportamiento inapropiado.
	H28	Como trabajador del campamento, puedo informar al gerente sobre escasez de suministros.
	H29	Como trabajador del campamento, puedo informar al gerente sobre reparaciones necesarias en el campamento.

5.1.3.2. Evaluación de la arquitectura propuesta por el grupo de investigación

5.1.3.2.1. Arquitectura desarrollada por el grupo de investigación.

Ya que se busca comparar resultados de la arquitectura propuesta por MOAM, se tiene el diagrama de la propuesta mejorada por el grupo de investigación, obteniendo así la arquitectura que se detalla en la Figura 11.

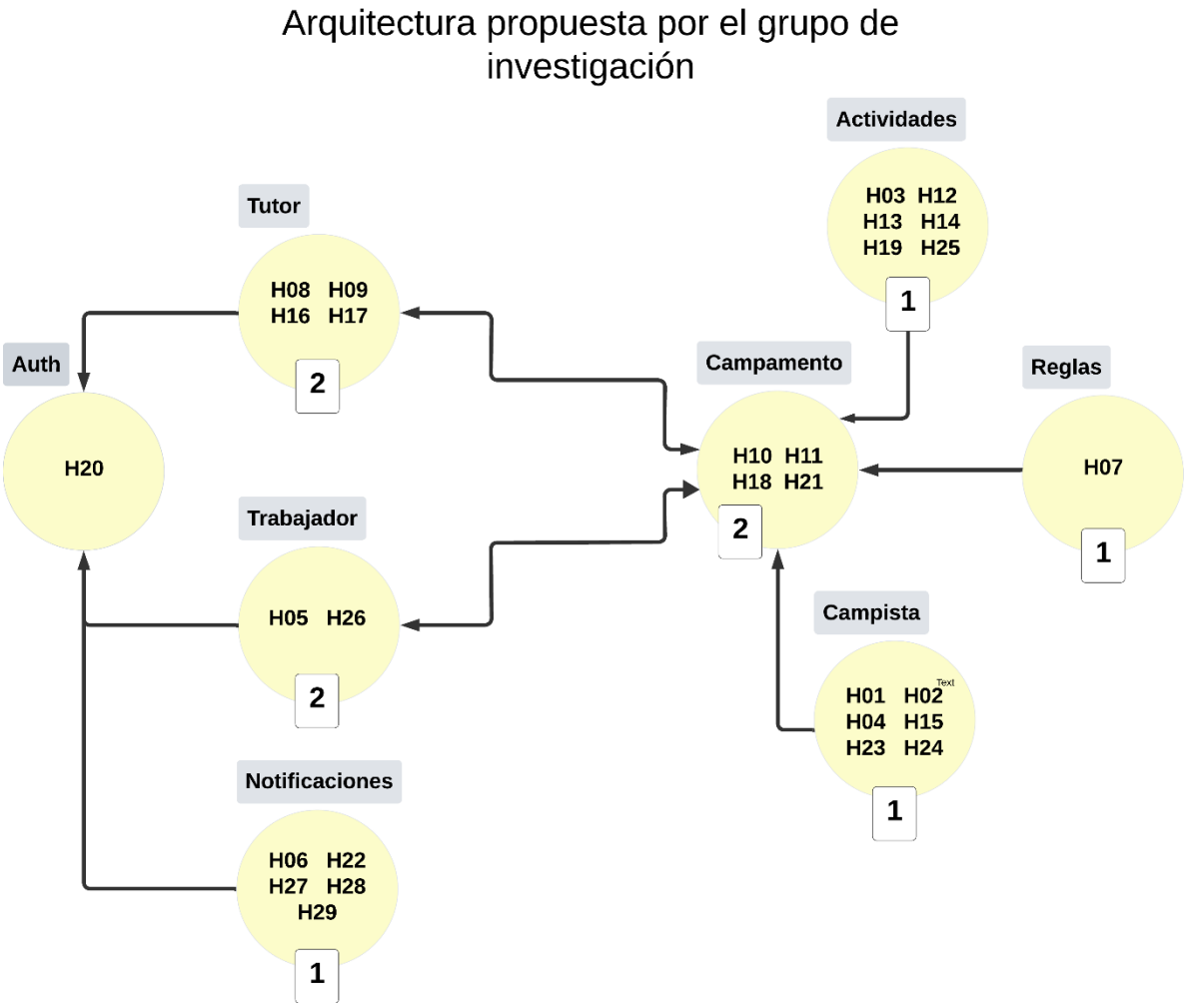

Figura 11. Diseño de arquitectura propuesto por el grupo de investigación.

5.1.3.2.2. Medición de métricas de calidad usando el software MOAM

Por otro lado, se evaluó la arquitectura mostrada en la Figura 11, haciendo uso del software MOAM, esto con el fin de comparar el modelado inicial otorgado por el software frente al modelado desarrollado por el grupo de investigación, obteniendo los siguientes resultados detallados en la Figura 12.

Figura 12. Métricas obtenidas a base de la arquitectura propia.

5.1.3.3. Comparativa de resultados de arquitecturas

Luego de haber realizado los procesos anteriormente mencionados se logró obtener una tabla recopilatoria de datos de las distintas métricas de calidad evaluadas, teniendo la Tabla 32.

Tabla 32. Recopilación de resultados de métricas de calidad.

Métrica de calidad	Resultado MOAM	Resultado Arquitectura propuesta por el grupo de investigación
Número de dependencias externas (ND)	0,0	0,0
Número de dependencias internas (NI)	3,0	8,0
Profundidad de las dependencias (PD)	0,0	0,0
Acoplamiento Estimado (AC_est)	6,5	4,0
Falta de Cohesión (LCOH_est)	1,7	1,6
Dependencia estimada (D_est)	4,7	9,6

Métrica de calidad	Resultado MOAM	Resultado Arquitectura propuesta por el grupo de investigación
Complejidad ciclomática estimada (CC_est)	2,0	7,0
Coeficiente de Acoplamiento estimado (CA_est)	0,0	0,0
Complejidad Estructural Estimada (CE_est)	6,0	16,0
Puntos de Historia Estimados (PH_est)	29,0	29,0
Complejidad Estimada (C_est)	37,0	52,0
Índice Agregado Estimado (AI_est)	41,7	61,6

Con el fin de mejorar la interpretación de los datos de la tabla anterior, se muestra la Figura 13, que permite visualizar la interpretación de los datos de manera fácil y comprensiva.

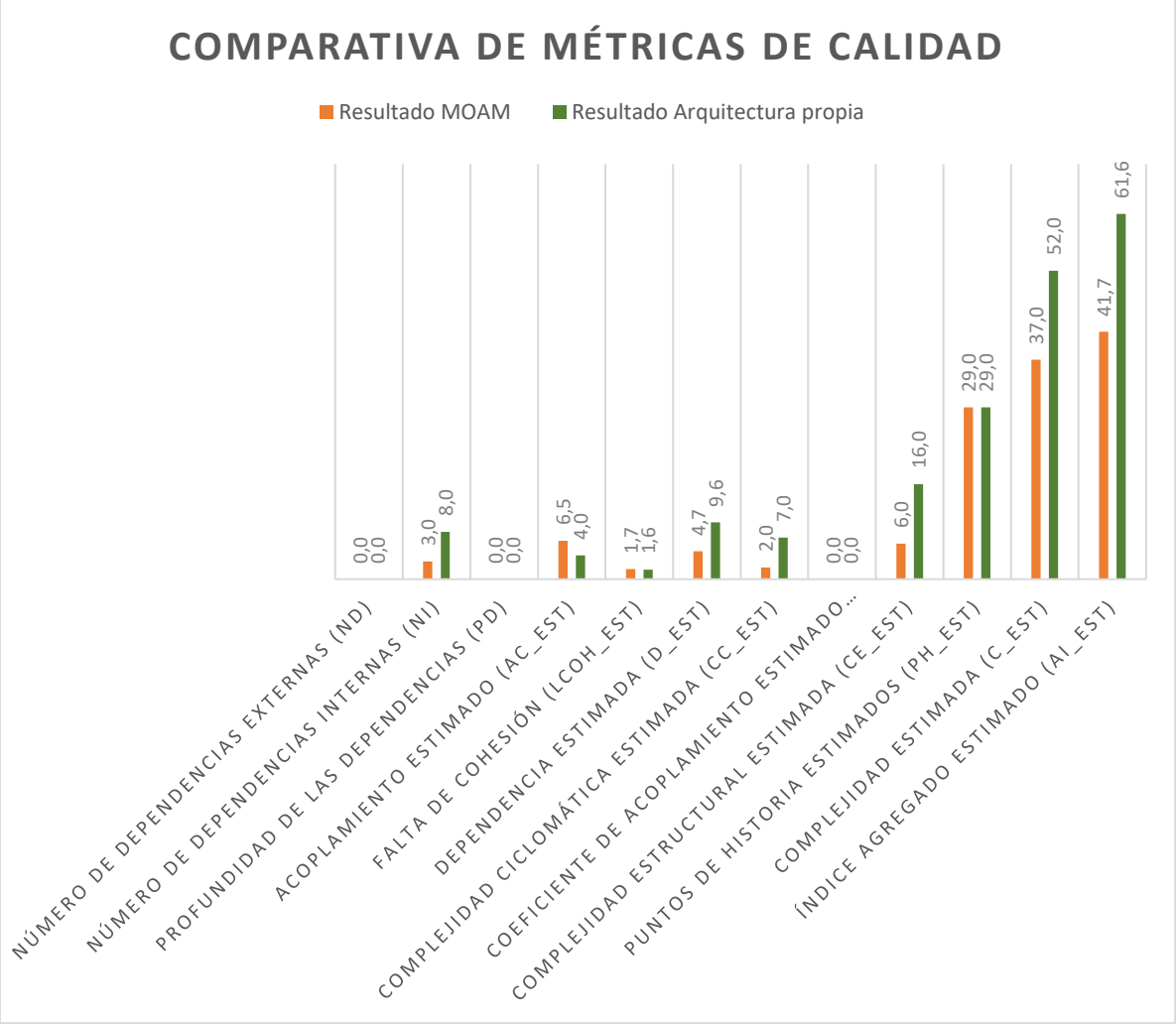

Figura 13. Comparativa general de la arquitectura.

5.1.3.4. Comparación por métricas de calidad

Partiendo de la Figura 15 y de los datos obtenidos, se podrá realizar la interpretación de los datos de manera individual, clasificándolos por métricas de dependencia y complejidad.

5.1.3.4.1. Métricas de dependencia

Número de dependencias internas

Comenzando con los resultados de esta métrica se tiene que, el número de dependencias internas propuesta por MOAM es mucho menor que el propuesto por el grupo de investigación, dicho resultado se puede ver en la Figura 14.

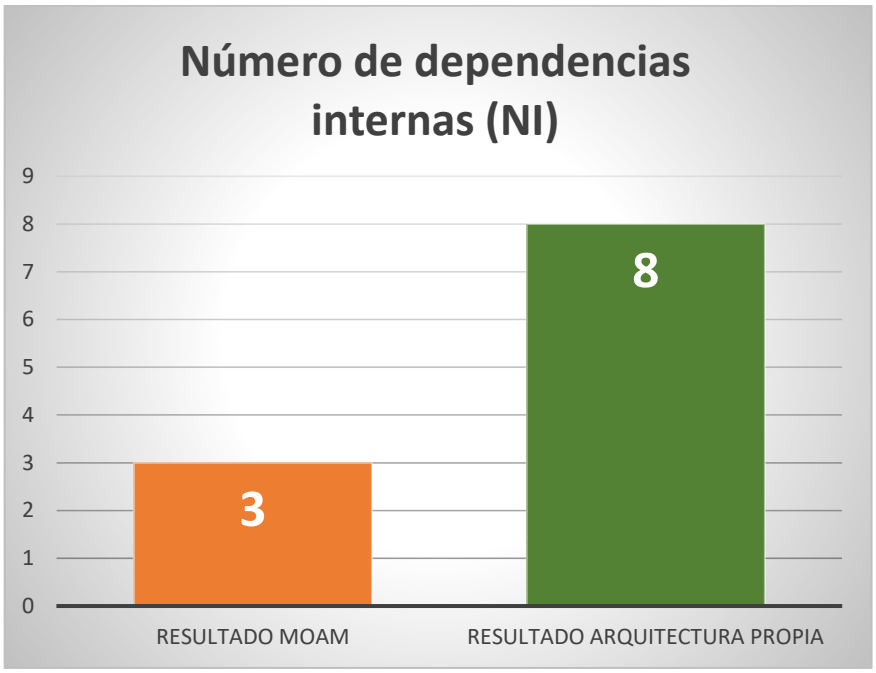

Figura 14. Dependencias Internas (NI)

Si bien es cierto, un número reducido de dependencias internas es mucho mejor para la arquitectura, la realidad es que de esta forma se produce un sobre acoplamiento de historias de usuario entre los servicios de la arquitectura

Para explicar esto de mejor manera, se muestra la Tabla 33 en la que se aprecia que el servicio Wantcampsite de la arquitectura MOAM, tiene alojada 22 historias de usuario, la cual teniendo en cuenta la Tabla 7 de rangos de acoplamiento, establece que un servicio que tiene más de 7 historias de usuario es muy complicado lo cual puede perjudicar el mantenimiento al tener muchas responsabilidades.

Tabla 33. Servicios MOAM

Servicio	Historias alojadas
Camp administrador	H07, H25, H27, H28, H29
Wantcampsite	H01, H02, H03, H05, H06, H08, H09, H10, H11, H12, H13, H14, H15, H16, H7, H18, H19, H20, H21, H22, H23, H24
Childparent	H04, H26

Como se observa el sobre acoplamiento en el módulo Wantcampsite, es más que evidente, por lo que el grupo de investigación optó por desacoplar las historias de usuario, tal y como se muestra en la Tabla 34, lo cual, si bien resulta en un aumento de servicios y de dependencias internas, reduce el acoplamiento de la arquitectura, manteniendo un número de servicios y de dependencias internas aun aceptable como se puede apreciar en la Tabla 5 que detalla los rangos de acoplamiento, en la que en cada módulo el número de historias no sobrepasa las 6 historias acopladas, con lo que se tiene que está en un rango medio de entre 6 y 10, por lo tanto es mantenible.

Tabla 34. Servicios grupo de investigación.

Servicio	Historias alojadas
Auth	H20
Tutor	H08, H09, H16, H17
Trabajador	H05, H26
Notificaciones	H06, H22, H27, H28, H29
Campamento	H10, H11, H18, H21
Campista	H01, H02, H04, H15, H23, H24
Actividades	H03, H12, H13, H14, H19, H15
Reglas	H07

Número de dependencias externas

Debido a que no se hallaron dependencias internas dentro de la arquitectura de MOAM como tampoco dentro de la arquitectura del grupo de investigación por lo cual da como resultado 0, lo cual según la Tabla 4 demuestra que está dentro de un rango bajo de 0 a 2, esto resulta beneficioso ya que no dependen de funcionalidades externas. Para visualizar de mejor manera estos resultados de tiene la Figura 15.

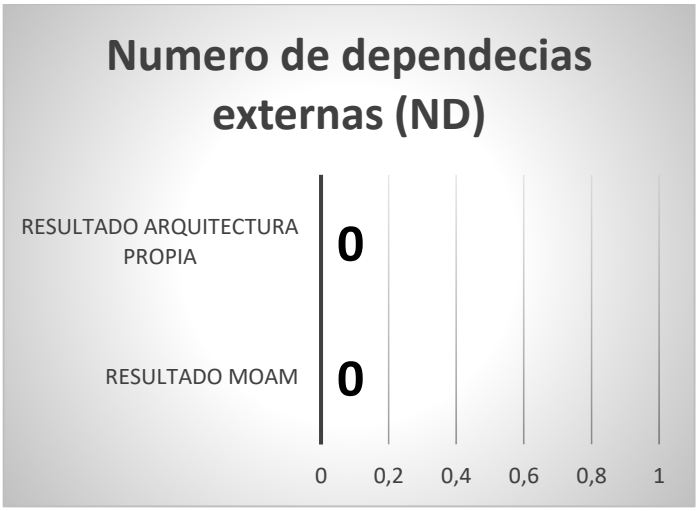

Figura 15. Total, dependencias externas

Profundidad de dependencias

Lo mismo ocurre en los resultados de esta métrica, ya que los resultados por ambos lados resultaron 0, lo cual según la Tabla 6 define que está en un rango bajo, que se traduce en que no necesita pasar por otros servicios para cumplir su propósito, esto validando que ambas arquitecturas tienen baja profundidad, como se muestra en la Figura 16.

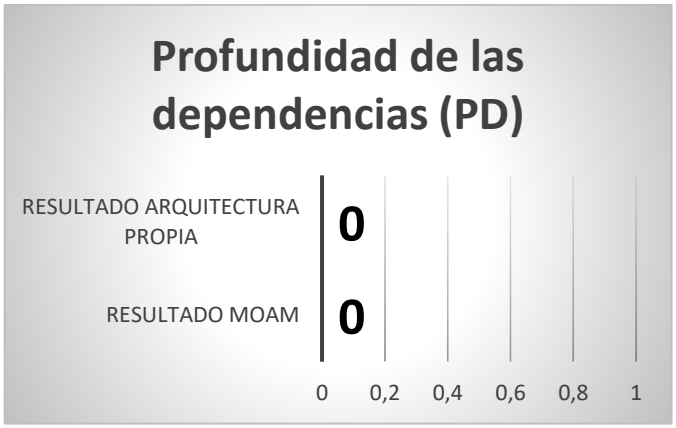

Figura 16. Comparativa Profundidad de las dependencias.

Acoplamiento estimado

En base a la recomendación otorgada por MOAM acerca de las métricas de su arquitectura la cual se encuentra en la Figura 12, que dice: “El acoplamiento interno estimado es 6.5. Un acoplamiento alto puede dificultar el mantenimiento y la escalabilidad. Considere implementar patrones de diseños basado en eventos o utilizar APIs para desacoplar los microservicios” se siguió dicha recomendación desacoplando las historias de usuario en más servicios, con lo cual una vez desarrollada la arquitectura propuesta por el grupo de investigación se volvió aplicar el componente de evaluación de métricas de MOAM y se obtuvo una reducción significativa de acoplamiento, esto ya que se redujo el número de historias de usuario acopladas en cada servicios, gracias al aumento de los mismo, entrando de esta manera en un rango medio de 4 a 6 historias de usuario alojadas en un servicio, lo que según la Tabla 6 está en un rango medio, dejando en claro que los servicios serán mantenibles,. Esto resultados se presentan de mejor manera en la Figura 17.

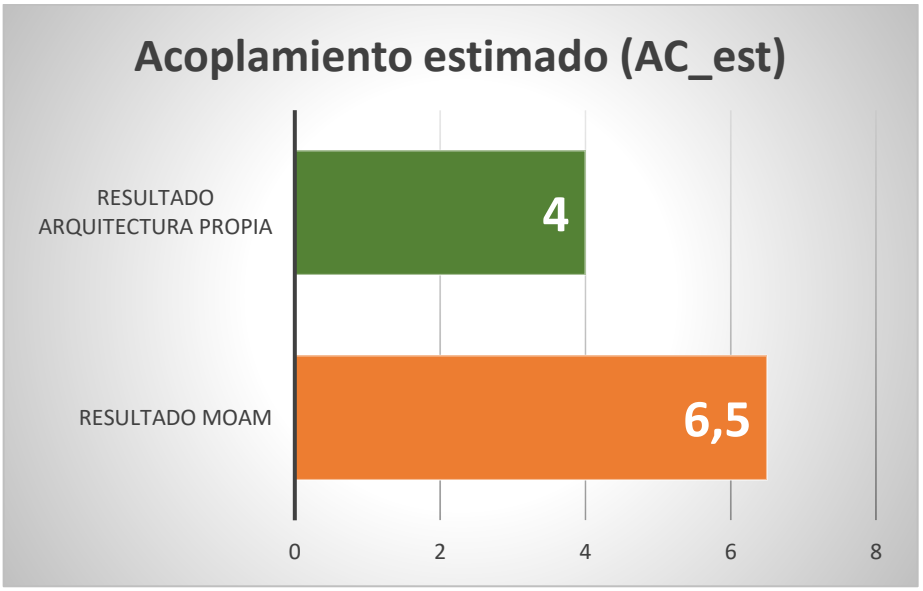

Figura 17. Comparativa Acoplamiento estimado.

Falta de cohesión estimada

En base a los resultados obtenidos entre la evaluación de ambas arquitecturas, el promedio de la propuesta de parte de MOAM es de 1.7, mientras que la del grupo de investigación fue de 1.6, lo cual en base a la Tabla 8, nos dice que está dentro de un rango medio de 1.1 a 2.0, con lo que se tiene que es un valor aceptable y posee una falta de cohesión aprobable, . La Figura 18 representa estos resultados de mejor manera.

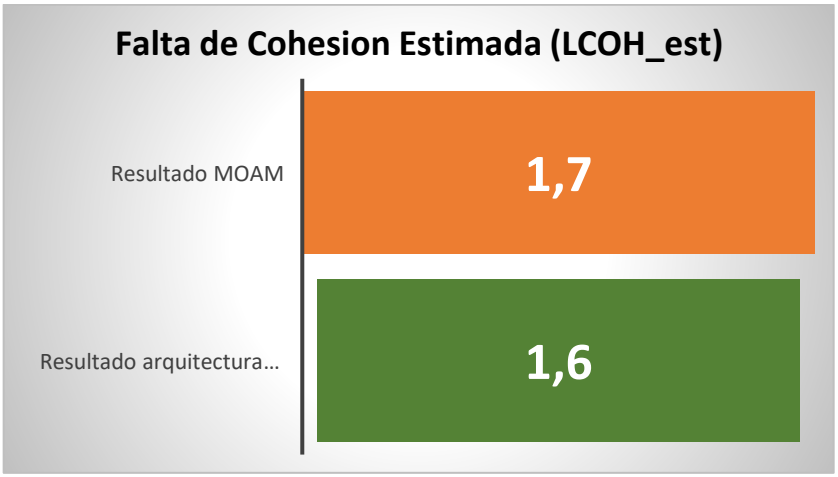

Figura 18. Comparativa Falta de Cohesión.

Dependencia estimada

Como último ámbito enfocado en dependencia se tiene la Figural 19 el total de dependencia de la arquitectura, en la que, se puede apreciar un aumento considerable, lo cual es comprensible teniendo en cuenta todos los cambios anteriormente mencionados, pero aun así considerando los rangos especificados en la Tabla 8 se dice que está en un rango medio de 9 a 12, lo que significa que es justa y no tendrá problemas en ámbito de dependencia en la arquitectura.

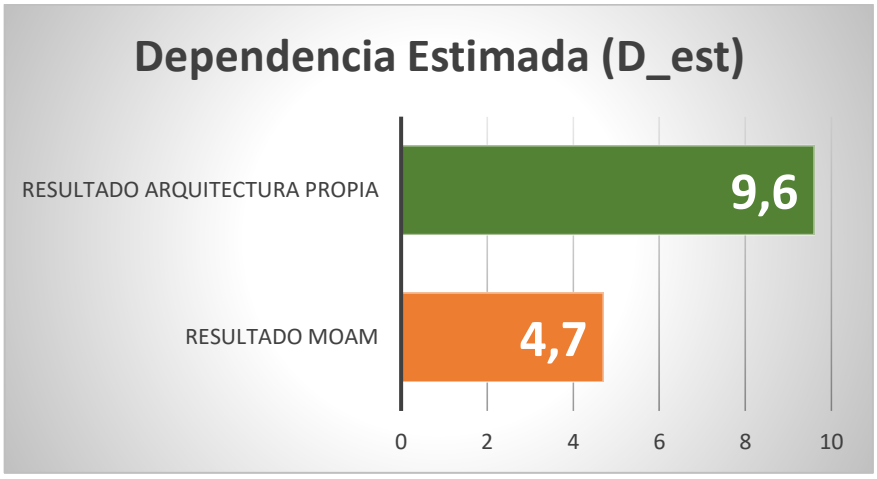

Figura 19. Comparativa dependencia

5.1.3.4.2. Métricas de complejidad

Complejidad ciclomática

Debido a que se incrementó el número de microservicios para reducir el acoplamiento de la arquitectura, la métrica de complejidad ciclomática subió, lo que da como resultado un valor de 7, frente al valor de 2 de la arquitectura de MOAM.

Aunque el valor aumentó, aún se encuentra dentro del rango de complejidad moderada especificada en la Tabla 9 que nos dice que está entre un rango medio de 6 a 8, que significa que la arquitectura no es muy compleja y será fácil de mantener. Dicho esto, se tiene la Figura 20 que nos muestra los datos.

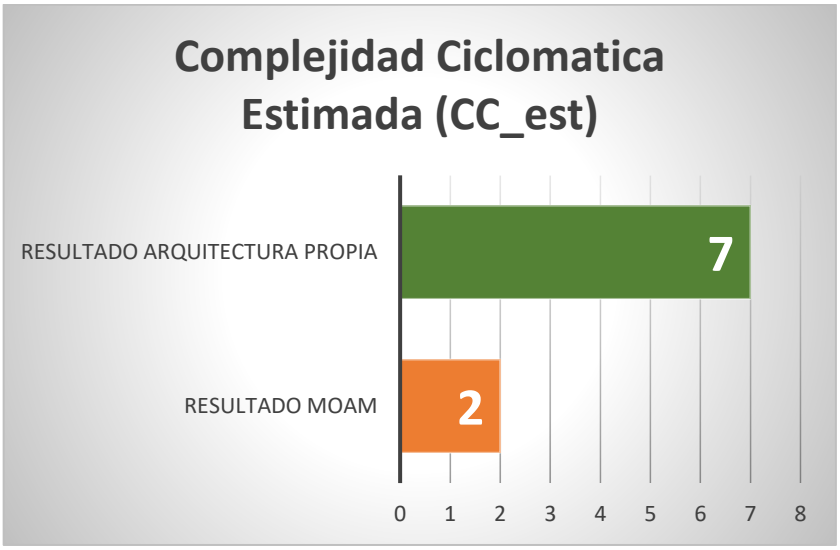

Figura 20. Comparativa Complejidad Ciclomática Estimada.

Coeficiente de acoplamiento estimado

Los resultados en ambas arquitecturas dieron como resultado 0, ya que se logró reducir la complejidad al redistribuir las historias de usuario al crear más servicio. Se tiene la Figura 21 que representa lo dicho y demuestra según la Tabla 10 que está en un rango bajo y no tendrá problemas en la dependencia debido al bajo acoplamiento.

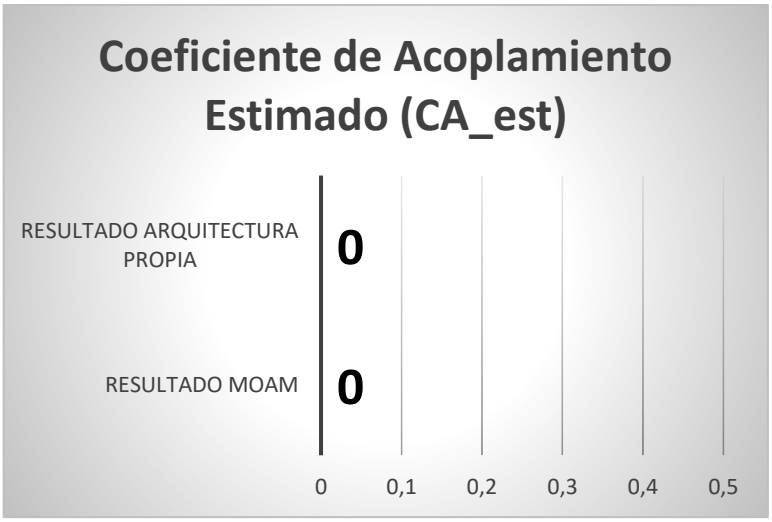

Figura 21. Comparativa Coeficiente de Acoplamiento Estimado.

Complejidad estructural estimada

Debido al aumento en el número de servicios, la arquitectura sufrió un aumento de complejidad, lo cual es entendible y no sobresale de la Tabla 11, que especifica como un rango medio de 16 a 20 puntos, lo que demuestra que la arquitectura no es muy compleja y podrá ser mantenible, con lo cual se tiene los resultados mostrados en la Figura 22.

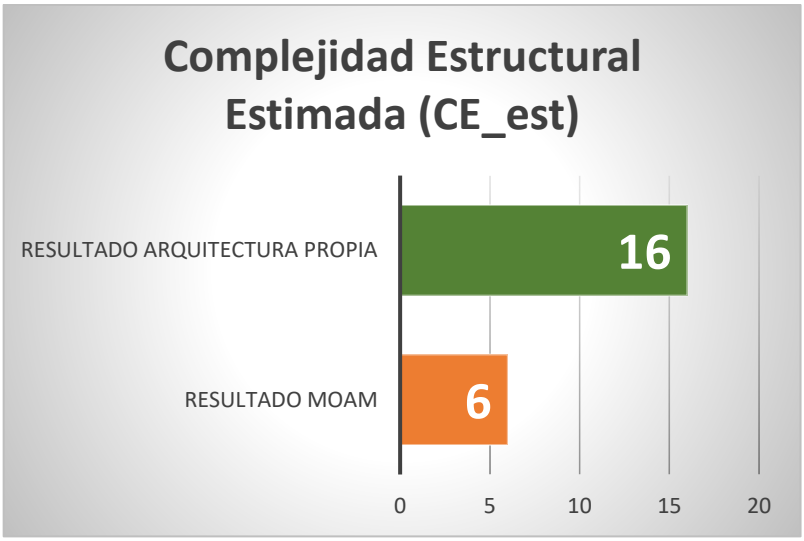

Figura 22. Comparativa Complejidad Estructural Estimada.

Puntos de historia estimados

Los puntos de historia aplicados en ambas arquitecturas fueron las mismas, por lo que no se encuentran varianzas en sus valores, como se puede apreciar en la Figura 23.

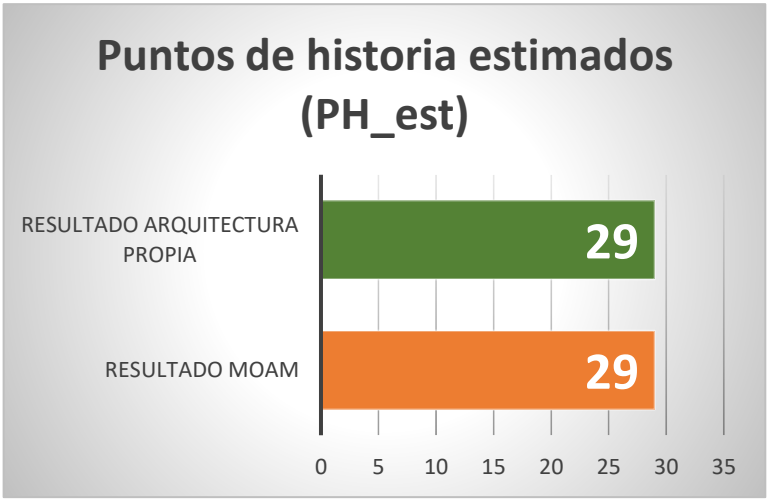

Figura 23. Comparativa Historias de Usuario.

Complejidad estimada

El resultado viene de la mano con el aumento de servicios antes mencionado, de tal forma que el resultado propuesto por el grupo de investigación, según la Tabla 12 está en un rango medio de 46 a 60 puntos, con lo que se dice que la complejidad total de la arquitectura no sobresale de lo óptimo para poder mantenerse, esto se representa en la Figura 24.

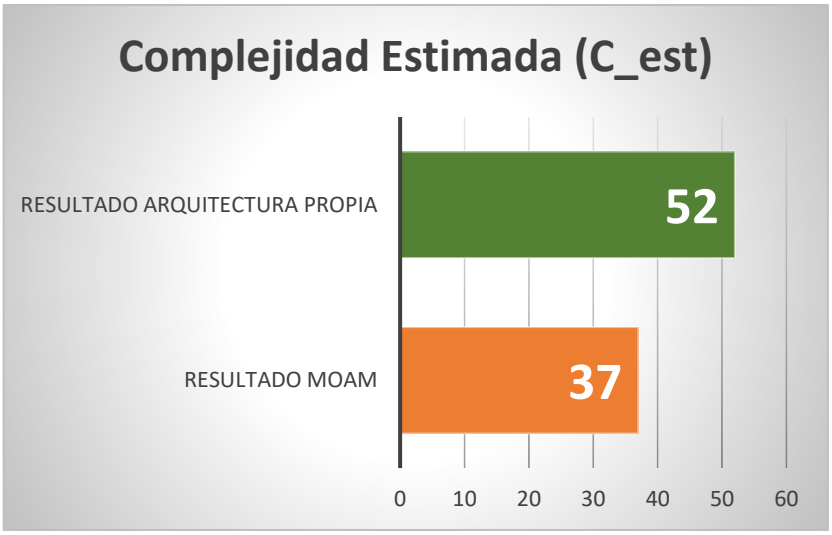

Figura 24. Comparativa Complejidad Estimada.

Índice agregado estimado

Esta métrica resume la calidad general de las arquitecturas evaluadas, ya que permite visualizar la salud general de la arquitectura, en la Figura 25 se puede apreciar un aumento en la arquitectura propuesta por el grupo de investigación de casi 50% dando como resultado 61.6 punto, en comparación a la propuesta por MOAM que tiene 41.7 puntos, lo cual es razonable y aceptable, para justificar esto se tiene en cuenta la Tabla 13 de rangos de índice estimado, que clasifica este valor en un rango medio al estar dentro de 46 y 60 puntos, dejando en claro que la salud de la arquitectura es óptima y se podrá actualizar y mejorar a futuro con otras funcionalidades.

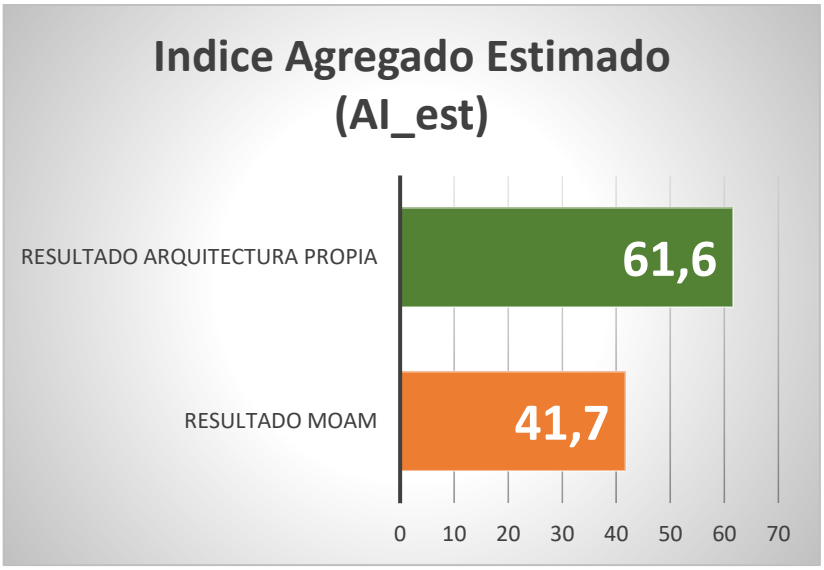

Figura 25. Comparativa Índice Agregado Estimado.

Una vez realizado todo el proceso de medición y comparación de métricas de calidad evaluadas se ha optado por usar la arquitectura propuesta por el grupo de investigación una vez que se obtuvieron las primeras recomendaciones de MOAM, y se propuso el modelo arquitectónico del grupo de investigación por las siguientes razones:

Posee un acoplamiento reducido, lo cual se vio evidenciado en el número de dependencias que posee cada servicio teniendo que la arquitectura propuesta por MOAM solo cuenta con 3 servicios en los cuales se distribuyen las 29 historias de usuario, lo cual produce un sobre acoplamiento.

Posee un número de servicios acorde a la cantidad de historias de usuario a aplicar. Además de estar dentro del rango aceptable en todas las métricas de calidad evaluadas.

5.1.4. Fase de puesta en producción

5.1.4.1. Gráfica resultante de la aplicación de la arquitectura hexagonal

Para la comprensión del funcionamiento de los microservicios se implementó la arquitectura hexagonal mediante un diagrama que muestra la interacción de las diferentes capas que la conforman como lo son: dominio, aplicación e infraestructura.

5.1.4.1.1. Arquitectura hexagonal de padre y trabajador

Para visualizar la interacción entre las diferentes capas de esta arquitectura, se puede visualizar mediante la Figura 26 los microservicios de padre y trabajador.

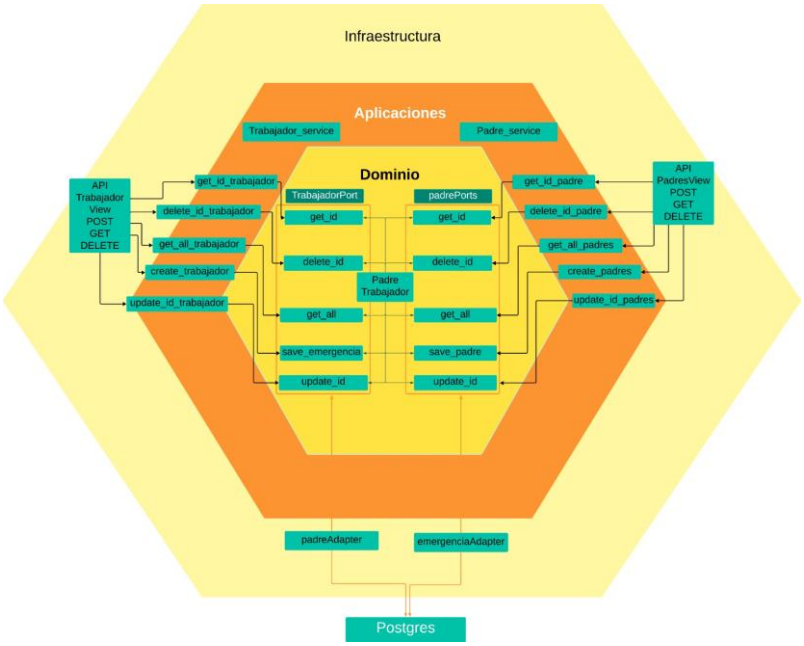

Figura 26. Arquitectura hexagonal.

5.1.4.1.2. Arquitectura hexagonal de actividades y grupo

Para visualizar la interacción entre las diferentes capas de esta arquitectura, se puede visualizar mediante la Figura 27 los microservicios de padre y trabajador.

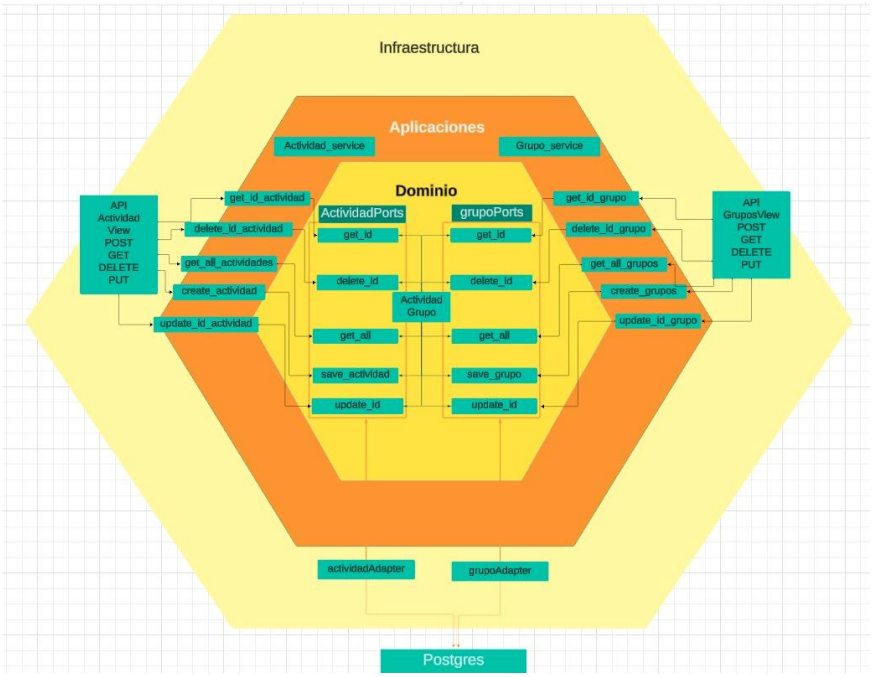

Figura 27. Arquitectura hexagonal de Actividades.

Una vez realizado todos los pasos anteriores lo que da es poner en producción la aplicación en base a la arquitectura propuesta por el grupo de investigación para lo cual se hace uso de la base de datos especificada en el Anexo B.

5.1.4.2. Elaboración del prototipo

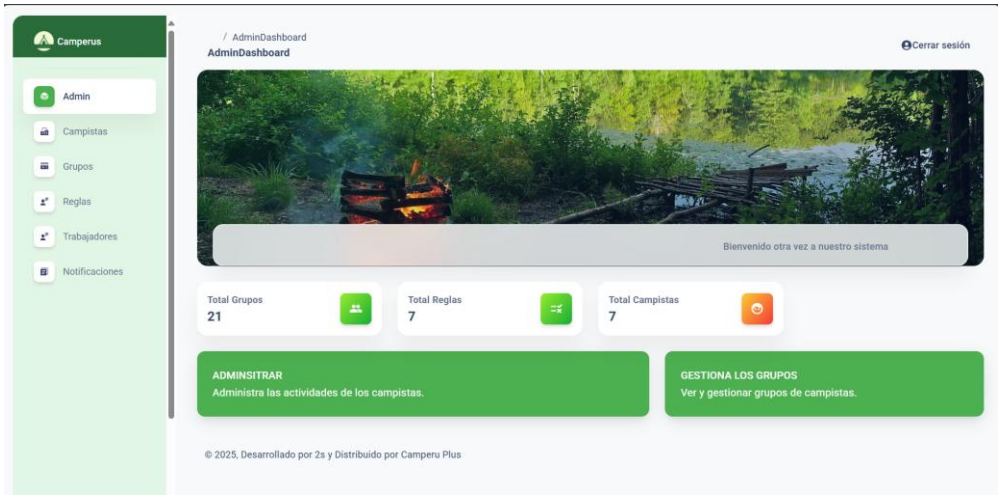

Figura 28. Prototipo en base a la arquitectura

Las demás capturas del prototipo se encuentran en el Anexo C. De la misma manera que la visualización de los servicios realizados se detalla mejor en el Anexo D.

5.1.4.3. Configuraciones del servidor de despliegue

- Crear Una aplicación para el front-end y el backend
- Crear una aplicación en Heroku para el front-end y comunicado con las APIs.

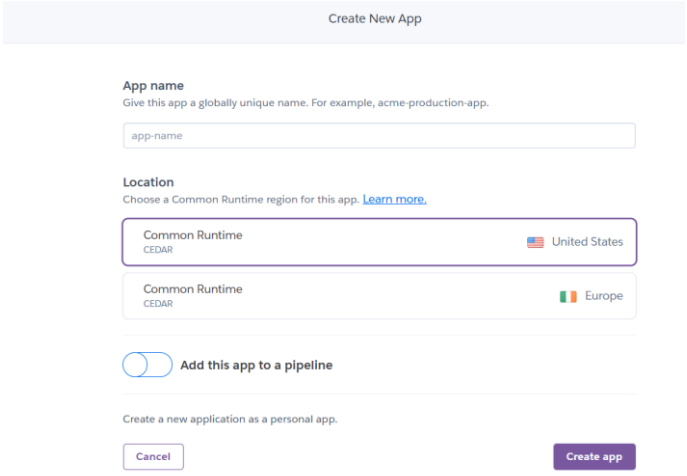

Figura 29. Creación de aplicación en Heroku.

- La aplicación creada proporcionará información del repositorio en el cual se podrá desplegar la aplicación del backend.

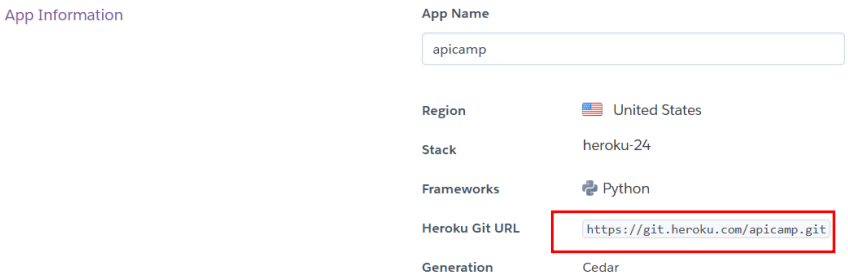

Figura 30. Información de la aplicación creada y repositorio asignando del backend.

- Crear la segunda app para el front-end, que está hecho en React el cual dará otro repositorio para el despliegue.

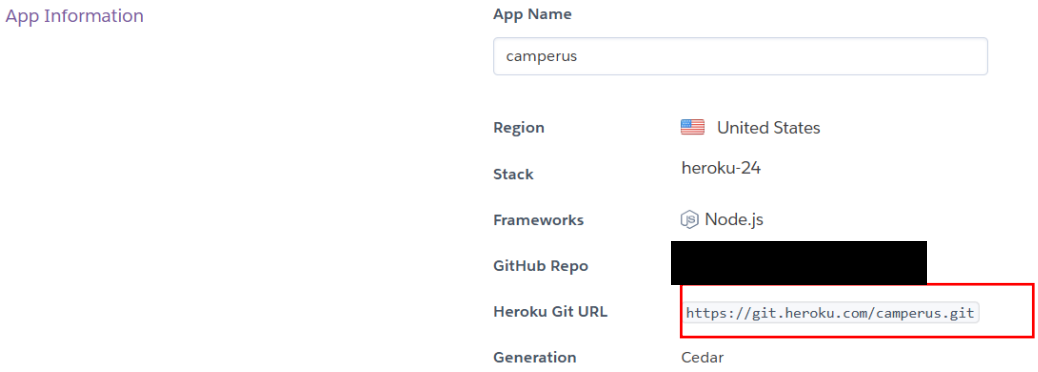

Figura 31. Información de la aplicación creada y repositorio asignado del front-end.

5.1.4.4. Despliegue aplicación en django Rest framework

- A continuación, se realiza cambios en settings de Django:

- dj_database_url permite configurar la base de datos a partir de una URL, lo que es útil para servicios como Heroku que proporcionan la URL de la base de datos a través de variables de entorno.

```
DATABASES = {
    'default': dj_database_url.config(conn_max_age=600)
}
```

Figura 32. Configuración de conexión a la BDD

- Esta configuración permite controlar el modo DEBUG a través de una variable de entorno.

```
DEBUG = os.environ.get('DEBUG', 'False') == 'True'
```

Figura 33. Configuración el modo de depuración de Django basado en una variable de entorno.

- CORS_ALLOWED_ORIGINS especifica los dominios permitidos, en este caso hace referencia hacia el front-end para que lo pueda usar.

```
#Manejo de ruta permitida para el uso de las rutas de la API
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "https://camperus-d319d8960732.herokuapp.com"
]
CORS_ALLOW_CREDENTIALS = True
```

Figura 34. Uso de cors para el manejo de las rutas.

- Añadir el archivo Procfile en la raíz del proyecto de django es necesario para que Heroku sepa cómo ejecutar la aplicación. Gunicorn es un servidor WSGI para aplicaciones Python que es adecuado para producción.

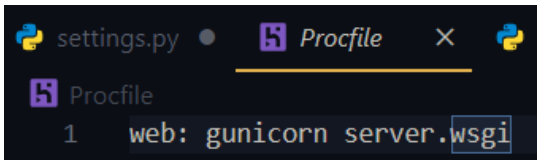

Figura 35. Inicio del servidor Gunicorn para servir la aplicación Django.

- Preparar los requeriment con el código pip freeze > requeriments.txt

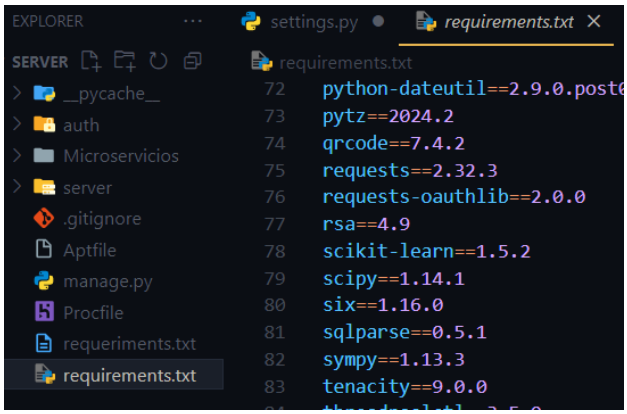

Figura 36. Lista de dependencias de Python

- Para finalizar se realiza un commit con los cambios al repositorio del proyecto al main de Heroku.

```
b\server> git add .
b\server> git commit -m "Despliegue"


b\server> git push heroku main
```

Figura 37. Commit de archivos de despliegue.

- Revisar los Logs de Heroku si es exitoso el despliegue, el correcto funcionamiento se vería de la siguiente forma.

```
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Updated 167 paths from ce55e43
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Building on the Heroku-24 stack
remote: -----> Using buildpack: heroku/python
remote: -----> Python app detected
remote: -----> No Python version was specified. Using the same major version as the last build: Python 3.13
remote:        To use a different version, see: https://devcenter.heroku.com/articles/python-runtimes
remote: -----> Restoring cache
remote: -----> Using cached install of Python 3.13.2
remote: -----> Installing pip 24.3.1
remote: -----> Installing dependencies using 'pip install -r requirements.txt'
remote: -----> $ python manage.py collectstatic --noinput
remote:        163 static files copied to '/tmp/build_4602c46a/staticfiles'.
remote:
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:        Done: 244.3M
remote: -----> Launching...
remote:        Released v14
remote:        https://apicamp-47b87972d284.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/apicamp.git
   55e84d3..1189826  main -> main
```

Figura 38. Registro de eventos y actividades de despliegue de Heroku.

- Se crean las migraciones si la aplicación lo requiera, o si no se han realizado previamente antes de desplegar la aplicación.

```
C:\Users\User\Documents\Tesis\repositorios_github\server>heroku run python manage.py makemigrations
Running python manage.py makemigrations on ⬢ apicamp... up, run.2433
No changes detected
```

Figura 39. Creación de migraciones.

- Ejecución de las migraciones para la base de datos o cambios de las tablas de la misma, para usar el gestor de base de datos de postgres.

```
C:\Users\User\Documents\Tesis\repositorios_github\server>heroku run python manage.py migrate
Running python manage.py migrate on ⬢ apicamp... up, run.4462
Operations to perform:
  Apply all migrations: actividades_infrastructure, admin, auth, auth_infrastructure, campamento_infrastructure, campista_infrastructure, contenttypes, notificacion_infrastructure, reglas_infrastructure, sessions, trabajador_infrastructure, tutor_infrastructure
Running migrations:
  No migrations to apply.
```

Figura 40. Realización de migraciones creación de la BDD y las tablas de la misma

- Una vez cumplido con los pasos, el despliegue correcto podría verse de esta forma

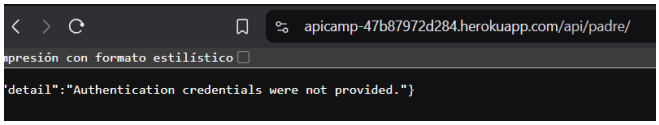

Figura 41. Vista de API desplegado.

5.1.4.5. Despliegue aplicación en React

- En el inicio start del proyecto se cambia el archivo package.json, este script indica a Heroku cómo iniciar la aplicación. Serve -s build utiliza el paquete serve para servir los archivos estáticos desde la carpeta build. Esto es necesario ya que Heroku necesita saber cómo ejecutar la aplicación una vez que está desplegada. Heroku ejecuta el script start por defecto cuando se despliega una aplicación Node.js.

Figura 42. Define scripts necesarios para preparar y desplegar la aplicación

- Se crea el archivo de Procfile, este le dice a Heroku qué comando ejecutar para iniciar la aplicación. En este caso, se utiliza el mismo comando serve -s build para servir los archivos estáticos desde la carpeta build. Esto es necesario porque Heroku necesita saber cómo ejecutar la aplicación una vez que está desplegada.

Figura 43. Inicio de la aplicación, archivos estáticos desde la carpeta build.

- Modificar conf.js en el llamado de las APIs, desplegado anteriormente de Django Rest framework

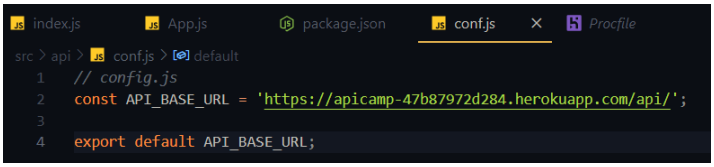

Figura 44. Configuración de la URL del uso de las APIs.

- Subir el proyecto al repositorio de Heroku, de la aplicación del proyecto en React.

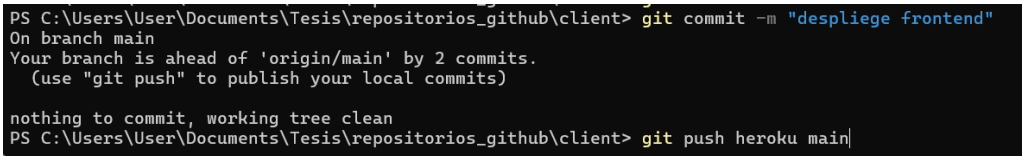

Figura 45. Despliegue del front-end al repositorio de Heroku

- Confirmar con los Logs arrojados por Heroku el correcto despliegue

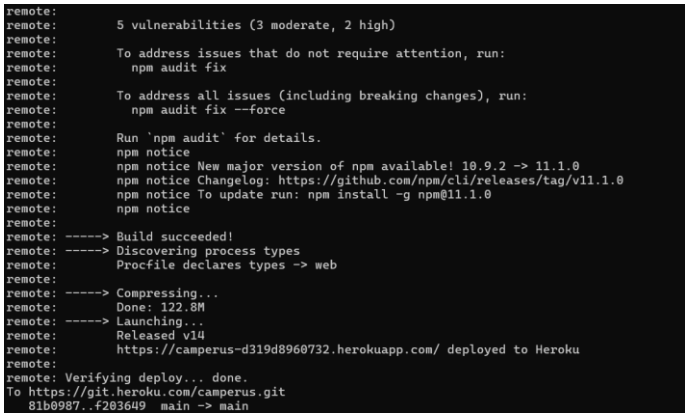

Figura 46. Registro de eventos y actividades de despliegue de Heroku.

- Con las correctas configuraciones el desplegado del proyecto está terminado, con lo cual el resultado será el siguiente.

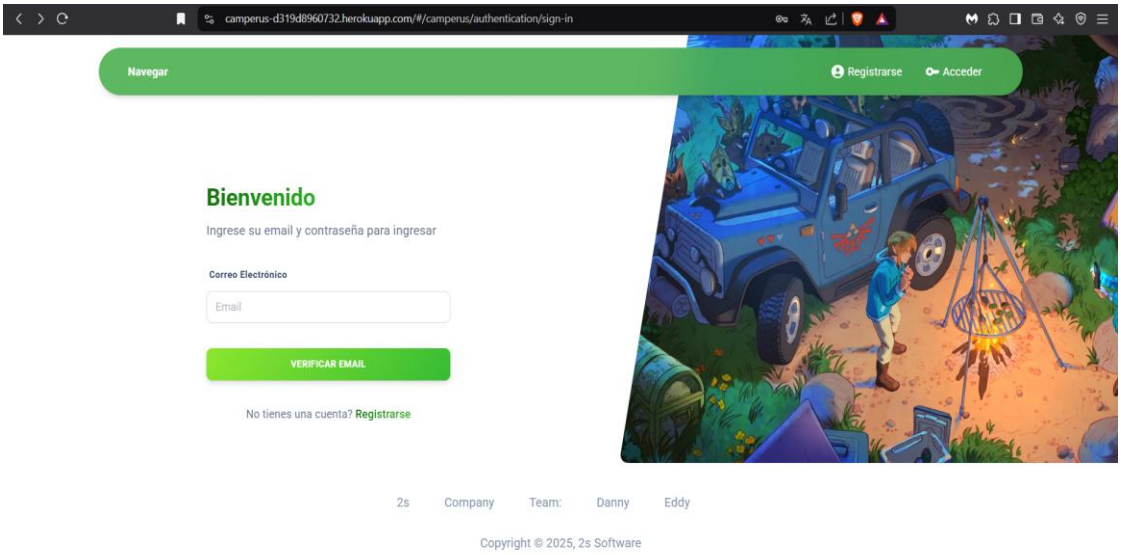

Figura 47. Aplicación front-end desplegada.

Para poder cumplir con la función de despliegue de la aplicación y todo el proceso de desarrollo se realizaron distinta serie de gastos, los cuales se encuentran detallados en el Anexo E

5.2. VALIDACIÓN DE LA HIPÓTESIS

5.2.1. Definición de métricas de calidad

Para poder validar la hipótesis que hace referencia a dependencia y complejidad, se hace uso de las distintas métricas que estas engloban estos aspectos y se evaluaron en el transcurso del proyecto, las cuales son las siguientes listadas:

- Número de dependencias externas (ND)
- Número de dependencias internas (NI)
- Profundidad de las dependencias (PD)
- Acoplamiento Estimado (AC_est)
- Falta de Cohesión (LCOH_est)
- Dependencia estimada (D_est)
- Complejidad ciclomática estimada (CC_est)
- Coeficiente de Acoplamiento estimado (CA_est)
- Complejidad Estructural Estimada (CE_est)
- Puntos de Historia Estimados (PH_est)
- Complejidad Estimada (C_est)

- Índice Agregado Estimado (AI_est)

5.2.2. Establecimiento de línea de mejoras

Se establecieron recomendaciones en base a la primera arquitectura evaluada la cual se observa en la Figura 48.

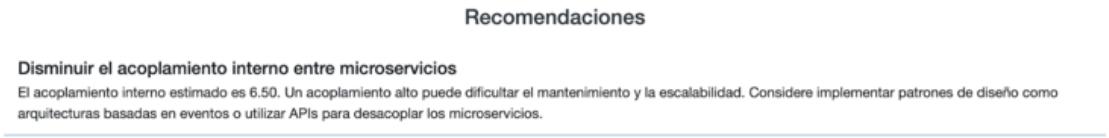

Figura 48. Recomendación MOAM

Con esto se obtuvo que, el acoplamiento estimado era muy alto con un valor de 6.5, lo cual fue un punto en el cual se estableció el análisis para mejorar, para ello se hizo énfasis en las métricas que afectan este valor, las cuales son número de dependencias internas y acoplamiento estimado.

5.2.3. Aplicar cambio y mejoras basadas en las métricas

En base a la recomendación se planteó una arquitectura, en la cual se obtuvo una reducción en el acoplamiento, reduciendo su valor a 4 puntos, lo cual se obtuvo al aumentar el número de servicios

De esta manera se obtuvo una reducción de acoplamiento, pero sin salir del rango aceptable mostrado en la Tabla 7, la que dice que es aceptable de 4 entre 6 puntos, dejando esta manera la dependencia en un rango moderado de entre 9 y 12, mostrado en la Tabla 9 .

5.2.4. Impacto en el desarrollo del prototipo

Gracias a los ajustes realizados, se logró obtener un balance en las métricas de calidad, ya que el resultado arrojado por MOAM acerca de la arquitectura propuesta, no resulto en ninguna recomendación, lo cual se puede apreciar en la Figura 49 y en las Tablas de rangos especificadas en el punto 3.4.1.1

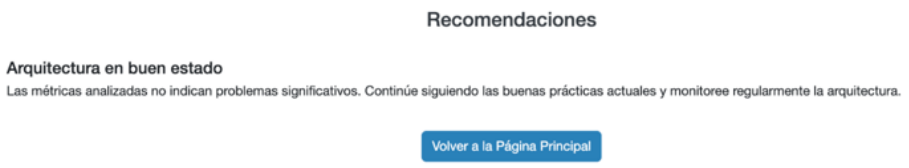

Figura 49. Recomendaciones Arquitectura propia

Con esto se evidencia que la medición de métricas de calidad nos ayuda a reducir la dependencia y complejidad de la arquitectura, para lo cual se ha logado desarrollar el prototipo Camperplus, el cual se basa en la arquitectura propuesta por el grupo de investigación la cual esta correctamente evaluada y sin ninguna novedad.

6. CONCLUSIONES Y RECOMENDACIONES

6.2. CONCLUSIONES

La fundamentación teórica ayudó a llegar a la resolución de que un alto acoplamiento puede aumentar la complejidad del sistema, ya que los cambios en un servicio pueden afectar a muchos otros, mientras que una alta complejidad interna en un microservicio puede dificultar su integración con otros servicios, aumentando las dependencias.

En base a la comparación de los resultados obtenidos de la evaluación de métricas de calidad de las arquitecturas propuestas, se logró determinar cómo más conveniente a la propuesta realizada por el grupo de investigación, al incorporar las recomendaciones de MOAM, esta alcanzo un equilibrio adecuado entre modularidad e independencia.

La implantación de la arquitectura evaluada confirmó que la aplicación de métricas de calidad en fases de análisis y diseño contribuye significativamente en la construcción de un sistema correctamente estructurado.

6.3. RECOMENDACIONES

Realizar una contextualización de información, en base a las métricas de dependencia y complejidad a evaluar, con el fin de identificar puntos clave y asegurar que el diseño de la arquitectura mantenga un equilibrio aceptable.

Tener en cuenta mejoras progresivas en base a la optimización de arquitecturas de microservicios, teniendo en consideración resultados de métricas de calidad enfocadas en dependencia y complejidad.

Al trabajar con microservicios se debe tomar en cuenta la adopción de una metodología ágil, además de tener en cuenta los servicios que este va a tener, procurando implementar una arquitectura que sea lo más modular posible, evitando así acoplamiento excesivo dentro de la misma para poder escalar a futuro.

7. BIBLIOGRAFIA

[1] L. H. J. Daniel, «Arquitectura de software basada en microservicios para desarrollo de aplicaciones web de la asamblea nacional,» Ibarra, 2017.

[2] S. Alexandra y T. Maria, «Diseño e implementación de aplicaciones basadas en microservicios utilizando un modelo de optimización de dependencia y complejidad,» p. 167, 2022.

[3] C. A. L. Mamami, «Pruebas de software para microservicios,» *Revista Innovacion y Software*, vol. 4, nº 1, p. 160, 2023.

[4] G. Castillo, «Microservicios:Descubra cómo hacer apps más escalables y rápidas,» 27 marzo 2024. [En línea]. Available: https://www.itmastersmag.com/transformacion-digital/microservicios-descubre-como-hacer-apps-mas-escalables-y-rapidas-de-desarrollar/. [Último acceso: 4 Octubre 2024].

[5] A. C. A.-A. y. A. M. Á.-C. M. Callejas-Cuervo, «Modelos de calidad del software, un estado del arte,» *Entramado,* vol. 13, nº 1, pp. 236-250, 2017.

[6] E. A. Gómez, «Arquitecturas de Software para Microservicios: Una Revisión Sistemática de la Literatura.,» p. 118, 2018.

[7] F. H. Vera-Rivera, «Cognitive complexity points: a metric to evaluate the design of microservices-based applications,» *Ingeniería y Competividad,* vol. 26, nº 1, p. 14, 2024.

[8] V. A. Iranzo, «Desarrollo de software basado en microservicios: un caso de estudio para evaluar sus ventajas e inconvenientes,,» de *Trabajo Fin de Grado, Escola Técnica Superior de ingenieria informatica*, Universidad Politécnica de Valéncia, 2018.

[9] Martin Ekuan,Courtney Wales, «Diseño de arquitectura de microservicios,» *Azure Archictecture Center,* p. 20, 11 Noviembre 2023.

[10] R. G. Fernández, «izertis,» 17 10 2023 . [En línea]. Available: https://www.izertis.com/es/-/blog/microservicios-domain-driven-design. [Último acceso: 2 1 2025].

[11] E. Norelus, «Medium,» Diseño y Tech.Co, 28 4 2019. [En línea]. Available: https://medium.com/design-and-tech-co/implementing-domain-driven-design-for-microservice-architecture-26eb0333d72e.

[12] A. M. Paredes, «Diseño de software aplicando el patrón domain-driven design,» Escuela Politécnica Superior, Madrid, 2021 .

[13] C. G. V. C. Tapia, «Microservices Optimization Architectures Model,» vol. (Versión 1.0).

[14] M. V. D. C. Tapia C, "Modelo inteligente para especificar dependencia y complejidad en el diseño de aplicaciones basadas en microservicios", Tesis doctoral, Universidad Técnica de Cotopaxi, Latacunga, 2024, en preparación.

[15] «Qué es la complejidad del software y cómo se puede gestionar,» In-com, 5 Marzo 2024. [En línea]. Available: https://www.in-com.com/es/blog/software-management-complexity/. [Último acceso: 19 Febrero 2025].

[16] A. Selim, «HexaGame, biblioteca online de videojuegos mediante Arquitectura Hexagonal.,» etsinf, Valencia, 2023.

[17] F. D. N. M. Mauro Cambarieri, «Implementación de una Arquitectura de Software guiada por el Dominio.,» *ASSE*, p. 18, 2020.

[18] N. D. V. Rodríguez, «Interfaz de usuario WEB para la accesibilidad de los diferentes desarrollos basados en Optimización e Inteligencia Artificial.,» Universidad de la Laguna, San Cristóbal de La Laguna, 2023.

[19] J. R. G. Barrios, «Inspección y mejoramiento de la calidad de código según métricas en sonarqube basadas en cobertura de pruebas unitarias, patrones y estandares de lenguajes de programacion implementados,» Guatemala, 2023.

[20] «Aplicacionew web: en que consisten y cuales son sus ventajas,» 24 noviembre 2020. [En línea]. Available: https://postgradoingenieria.com/que-son-aplicaciones-web/?form=MG0AV3. [Último acceso: 2 enero 2025].

[21] R. E. Hurtado, «Análisis comparativo para la evaluación de frameworks usados en el desarrollo de apliaciones web,» *CEDEMAZ*, vol. 11, nº 20, p. 9, 2021.

[22] J. Vainikka, «Full-stack web development using Django REST framework and React,» Vantaa, 2018.

[23] J. J. Villar, Descrubre React, Leanpub, 2016.

[24] M. Cantó, «Arquitectura Domótica de bajo coste.,» p. 69, 2018.

[25] E. G. Maida y J. Paciencia, «Metodologias de desarrollo de software,» Argentina, 2015.

[26] «kiwop,» 12 abril 2020. [En línea]. Available: https://www.kiwop.com/blog/metodologias-agiles-en-el-desarrollo-web?form=MG0AV3. [Último acceso: 2 enero 2025].

[27] M. L. J. German, «Metodología de conversión de aplicaciones monoliticas a microservicios desplegable en la nube para pequeñas empresas,» Peru, 2022`.

[28] W. O. L. U. J. A. Marcos Carrasco, «Metodología híbrida de desarrollo de software combinando xp y scrum,» *Mikarimin*, p. 8, 2019.

[29] Marco Vinicio Estrada Velasco,Jenny Alexandra Núñez-Villacis, Pedro Rubén Saltos Chávez, Wilmer Clemente Cunuhay Cuchipe, «Revisión Sistemática de la Metodología Scrum para el Desarrollo de Software,» *Revista Cientifica Dominio de las Ciencias* , vol. 7, nº 4, pp. 434-447, 2021.

[30] Marlon Castillo Anzules, Javier Guaña Moya, «Kanvan: Una metodología para la getion eficiente del flujo de trabajo en el desarrollo de software, una revisión de sistemática,» *ING EniLobal*, vol. 3, nº 1, 2024.

[31] F. Dalpiaz, «Requirements data sets (user stories),» Mendeley Data, July 2018. [En línea]. Available: https://data.mendeley.com/datasets/7zbk8zsd8y/1. [Último acceso: 21 10 2024].

[32] M. Lacchia, «Radon 4.1.0 documentation,» 2020. [En línea]. Available: https://radon.readthedocs.io/en/latest/commandline.html#cyclomatic-complexity. [Último acceso: 2025 Enero 9].