UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

CARRERA DE INGENIERÍA DE SISTEMAS DE INFORMACIÓN

DESARROLLAR UN PROTOTIPO DE SISTEMA DE RECOMENDACIÓN Y DESCUBRIMIENTO DE CONTENIDOS MULTIMEDIA, APLICANDO UNA ARQUITECTURA DE MICROSERVICIOS, PARA COMPROBAR LAS MÉTRICAS DE CALIDAD DE DEPENDENCIA Y COMPLEJIDAD.

PROYECTO DE INVESTIGACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS DE INFORMACIÓN.

AUTOR(ES):

Gary Ismael Jami Chuquilla

Katherin Elizabeth Quinatoa Jami

TUTOR:

Ing. Verónica del Consuelo Tapia Cerda, PhD.

LATACUNGA, AGOSTO, 2025

Latacunga, agosto 2025

DECLARACIÓN DE AUTORÍA

Nosotros, Gary Ismael Jami Chuquilla con C.I.: 050313640-0 y Katherin Elizabeth Quinatoa Jami con C.I.: 050435024-0, ser los autores del presente proyecto de Investigación: "Desarrollar un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia, aplicando una arquitectura de microservicios, para comprobar las métricas de calidad de dependencia y complejidad", siendo la Ing. Verónica del Consuelo Tapia Cerda, PhD, tutora del presente trabajo de titulación; y eximo expresamente a la Universidad Técnica de Cotopaxi y a sus representantes legales de posibles reclamos o acciones legales.

Además, certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo de titulación, son de mi exclusiva responsabilidad.

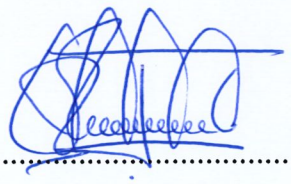
Atentamente,

.....	.....
Gary Ismael Jami Chuquilla	Katherin Elizabeth Quinatoa Jami
CI: 050313640-0	CI: 050435024-0

Latacunga, agosto 2025

AVAL DEL TUTOR DE PROYECTO DE TITULACIÓN

En calidad de Tutor del Trabajo de Investigación sobre el título: "Desarrollar un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia, aplicando una arquitectura de microservicios, para comprobar las métricas de calidad de dependencia y complejidad", propuesto por los estudiantes: Gary Ismael Jami Chuquilla y Katherin Elizabeth Quinatoa Jami de la Carrera de Ingeniería en Sistemas de Información, considero que dicho Trabajo de titulación cumple con los requerimientos metodológicos y aportes científico-técnicos suficientes para ser sometidos al tribunal de lectores.



.....

Ing. Verónica del Consuelo Tapia Cerda, PhD

CC.: 0502053697

TUTORA

AVAL DE APROBACIÓN DE LECTORES

Cumplimiento con el Reglamento de Titulación de la Universidad Técnica de Cotopaxi, en calidad de Lectores de tribunal del Proyecto de Investigación en el Título: **"Desarrollar un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia, aplicando una arquitectura de microservicios, para comprobar las métricas de calidad de dependencia y complejidad"**, propuesto por los estudiantes: **Gary Ismael Jami Chuquilla y Katherin Elizabeth Quinatoa Jami** de la Carrera de Ingeniería en Sistemas de Información, me permito indicar que los estudiantes han concluido todas las observaciones y realizado las correcciones señaladas por el Tribunal de Lectores, por lo cual presentamos el Aval de aprobación del Proyecto de Titulación correspondiente a la modalidad Proyecto de Investigación en virtud de lo cual los postulantes pueden presentarse a la Defensa de su Proyecto de Titulación.

Particular que pongo en su conocimiento por los finas legales pertinentes.

Atentamente,

Lector 1 (presidente)	Lector 2	Lector 3
CANTUÑA FLORES KARLA SUSANA	**BEDÓN SALAZAR EDISON PATRICIO**	**CHANCUSIG CHISAG JUAN CARLOS**
0502305113	0502253271	0502275779

AGRADECIMIENTO

Agradezco profundamente a mis padres, Blanca Chuquilla y Angel Vilca, por su apoyo incondicional durante este proceso de sacrificio y dedicación. A mi madre, por su comprensión en cada etapa de este desafío; a mi padre por su paciencia y fortaleza en los momentos más exigentes de esta etapa de mi vida.

A mis abuelitos, hermanos y familia en general, gracias por ser mi red de sostén emocional y motivación constante. Un reconocimiento especial a mi tío Marco Chuquilla, cuya presencia en los momentos más difíciles de mi vida fue mi ejemplo a seguir y el cual no permitió rendirme.

Al jurado calificador, valoro enormemente su tiempo y aportes para enriquecer mi formación.

Sobre todo, doy gracias a Dios por la fortaleza y oportunidades que me permitieron culminar este gran paso hacia mi titulación. Este logro es también el reflejo de cada persona que creyó en mí, y me comprometo a honrarlo con mayor esfuerzo y disciplina en los desafíos venideros.

Quiero expresar mi más profundo agradecimiento, en primer lugar, a Dios, por brindarme la sabiduría, la fortaleza y la vida necesarias para culminar con éxito esta etapa tan importante.

A mi madre, María Jami, gracias por ser mi pilar incondicional, por no soltar mi mano en los momentos difíciles y por animarme siempre a seguir adelante. A mi padre, Manuel Quinatoa, por enseñarme con su ejemplo a ser valiente y a enfrentar la vida con firmeza, sin importar las circunstancias.

A mis hermanas, quienes con su amor y alegría supieron regalarme sonrisas incluso en los días más complicados. Cada uno de ustedes ha sido parte fundamental de mi camino, de mis esfuerzos, de mis lágrimas y de mis logros.

Agradezco también a Alexander C., quien, aunque hoy ya no forma parte de mi vida, estuvo presente durante un periodo importante y me brindó su apoyo incondicional en momentos clave de mi formación. Su compañía en ese tiempo fue valiosa, y le guardo gratitud por ello.

Agradezco de corazón a quienes compartieron conmigo tantas vivencias durante esta etapa universitaria: **Deyaneira Vizuete** y **Gary Jami**, amigos leales que estuvieron presentes en cada aventura, apoyándome con su compañía, risas y amistad incondicional.

Mi sincero reconocimiento al tribunal evaluador, por sus observaciones constructivas que nos impulsan a crecer y a convertirnos en mejores profesionales.

Finalmente, agradezco a la Universidad Técnica de Cotopaxi, mi alma mater, por brindarme las herramientas académicas y formativas que hoy me permiten presentarme con orgullo como profesional.

DEDICATORIA

Con profundo amor y gratitud, dedico este logro académico a las personas que han sido mi soporte, mi motivación y mi razón para perseverar.

A mis padres, Blanca Piedad Chuquilla Tasinchana y Ángel Fernando Vilca Viturco, por su sacrificio incansable, sus enseñanzas y por ser los cimientos de quien soy hoy. A ti, padre, aunque no la sangre nos una, tu amor incondicional, tu paciencia ante mis errores y tu fe en mis metas han forjado en mí el mismo respeto y admiración que un hijo puede tener a su padre. Esta tesis lleva también sus nombres.

A mis hermanos, Fátima Anahí Vilca Chuquilla y Esteven Daniel Vilca Chancusig, por acompañarme en cada caída y celebración, demostrándome que la familia es el lazo más fuerte.

A mi querida abuelita, María Margarita Tasinchana Chancusig, por ser mi refugio de sabiduría y ternura. Y a mi tío, Marco Rodrigo Chuquilla Tasinchana, quien no solo me crió con la dedicación de un padre, sino que moldeó con ejemplo y disciplina el hombre en que me he convertido.

A ti, Carla Alomoto, mi pareja y compañera: tu paciencia en las noches de desvelo, tus palabras de aliento cuando dudaba hacen de este logro algo aún más valioso ya que también de lo dedico a ti.

Esta tesis no es solo mía; es el reflejo de sus risas, lágrimas y sueños compartidos. Gracias por creer en mí incluso cuando yo no lo hacía.

Con todo mi corazón,

Gary Ismael Jami Chuquilla

Desde lo más profundo de mi corazón, dedico esta tesis primeramente a Dios, por haberme dado la sabiduría, la fortaleza y el conocimiento necesarios para seguir adelante.

A mis padres, María Jami y Manuel Quinatoa, por su amor incondicional y por ser siempre mi mayor sostén. A mis hermanas Maribel Jami, Lisbeth Quinatoa y Maitee Quinatoa, y a mi querido sobrino Ian Taipe, quienes han sido mi inspiración diaria y el motor para esforzarme cada día con el deseo de darles lo mejor.

Con especial amor y gratitud, dedico este logro a mi abuelita Clementina Vásquez y a mi tío Jorge Jami, quienes partieron antes de ver cumplidos mis sueños, pero cuyo recuerdo y enseñanzas siguen vivos en mi corazón y me han acompañado a lo largo de este camino.

Esta tesis representa cinco años de esfuerzo, dedicación y constancia. A toda mi familia, gracias por enseñarme a ser fuerte y nunca rendirme, especialmente a mis padres, quienes siempre fueron mi refugio en los momentos difíciles.

Con todo mi amor,

Elizabeth Quinatoa

UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

TITULO: "Desarrollar un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia, aplicando una arquitectura de microservicios, para comprobar las métricas de calidad de dependencia y complejidad**"**

Autores:

Gary Ismael Jami Chuquilla

Katherin Elizabeth Quinatoa Jami

RESUMEN

El presente trabajo se realizó en 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", el objetivo principal fue desarrollar un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia aplicando una arquitectura de microservicios para comprobar las métricas de calidad de dependencia y complejidad.

La arquitectura generada fue evaluada a través de la herramienta MOAM (Microservices Optimization Architectures Model), obteniendo resultados que se ubicaron mayoritariamente en rangos bajos y moderados, lo que indica una estructura modular y manejable. El proceso de despliegue fue documentado paso a paso, desde la configuración del entorno hasta la implementación en el servidor, lo cual, valida la viabilidad técnica del sistema propuesto, se demuestra que el uso de microservicios junto con herramientas de análisis como MOAM representa una estrategia efectiva para optimizar el diseño de sistemas complejos.

Esta investigación abre la puerta a trabajos futuros que podrían profundizar en otras métricas como seguridad, rendimiento y mantenibilidad, fortaleciendo así la calidad integral de aplicaciones modernas basadas en microservicios.

Palabras Claves: Sistemas de recomendación, métricas de calidad, microservicios, herramientas.

TECHNICAL UNIVERSITY OF COTOPAXI

FACULTY OF ENGINEERING SCIENCES AND APPLIED

THEME: "To develop a prototype of a multimedia content recommendation and discovery system by applying a microservices architecture in order to evaluate the quality metrics of dependency and complexity."

Authors:

Gary Ismael Jami Chuquilla

Katherin Elizabeth Quinatoa Jami

ABSTRACT

This work was carried out at the Technical University of Cotopaxi, specifically in the Information Systems program, as part of the formative research project titled: "Study of Dependency and Complexity Metrics for the Design of Microservices Architectures." The main objective was to develop a prototype of a multimedia content recommendation and discovery system by applying a microservices architecture to evaluate quality metrics of dependency and complexity.

The generated architecture was evaluated using the MOAM (Microservices Optimization Architectures Model) tool, obtaining results that mostly fell within low and moderate ranges, indicating a modular and manageable structure. The deployment process was documented step by step, from environment configuration to server implementation, validating the technical feasibility of the proposed system. This demonstrates that the use of microservices, together with analysis tools like MOAM, represents an effective strategy for optimizing the design of complex systems.

This research opens the door for future work that could delve into other metrics such as security, performance, and maintainability, thereby strengthening the overall quality of modern microservices-based applications.

Keywords: Recommendation systems, quality metrics, microservices, tools.

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 Iidioma Inglés del proyecto de investigación cuyo título versa: "**DESARROLLAR UN PROTOTIPO DE SISTEMA DE RECOMENDACIÓN Y DESCUBRIMIENTO DE CONTENIDOS MULTIMEDIA, APLICANDO UNA ARQUITECTURA DE MICROSERVICIOS, PARA COMPROBAR LAS MÉTRICAS DE CALIDAD DE DEPENDENCIA Y COMPLEJIDAD**" presentado por **Gary Ismael Jami Chuquilla** y **Katherin Elizabeth Quinatoa Jami**, egresados de la carrera de **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, Julio 2025

Atentamente,

CENTRO DE IDIOMAS

Mg. Lidia Rebeca Yugla Lema.
DOCENTE DEL CENTRO DE IDIOMAS-UTC
0502652340

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INFORMACIÓN GENERAL

TEMA DEL PROYECTO:

Desarrollar un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia, aplicando una arquitectura de microservicios, para comprobar las métricas de calidad de dependencia y complejidad.

MODALIDAD DE TITULACIÓN:

Tabla 1. Modalidad de titulación

MODALIDAD DE TITULACIÓN	HOMOLOGACIONES PARA INFORME FINAL DE TITULACIÓN	SELECCIÓN
Propuesta tecnológica	Informe de propuesta tecnológica	
	Patente, Modelo de utilidad, Certificado de propiedad intelectual.	
	Artículo científico	
Proyecto de investigación	Informe de Proyecto de investigación	X
	Artículo científico	
	Patente, Modelo de utilidad, Certificado de propiedad intelectual.	
Exámen de indicadores de RDA		

TRABAJO DE TITULACIÓN VINCULADO AL PROYECTO:

Estudio de métricas de dependencia y complejidad para el diseño de arquitecturas de microservicios.

EQUIPO DE TRABAJO DEL TRABAJO DE TITULACIÓN:

ESTUDIANTES:

Gary Ismael Jami Chuquilla

Katherin Elizabeth Quinatoa Jami

TUTOR:

Dra. Verónica del Rocio Tapia Cerda, PhD

GRUPO DE INVESTIGACIÓN:

Desarrollo tecnológico para sistemas de información automatizados.

ÁREA DE CONOCIMIENTO:

Tabla 2. Área de conocimiento Unesco

06 Información y Comunicación (TIC)	061 Información y Comunicación (TIC)	0611 El uso del Ordenador
		0612 Base de datos, diseño y administración de redes
		0613 Software y desarrollo y análisis de aplicativos

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, el desarrollo de aplicaciones ya sea web, móvil o de escritorio se basan en arquitectura monolíticas este enfoque es una actividad esencial para satisfacer las necesidades de automatización de procesos internos. Este desarrollo influenciado por tendencias tecnológicas, lenguajes de programación y la experiencia de los equipos de desarrollo, ha llevado a la implementación predominante de arquitecturas monolíticas, aunque este enfoque es ampliamente utilizado presenta limitaciones importantes, especialmente en el despliegue y mantenimiento de las aplicaciones.

Las aplicaciones basadas en arquitecturas monolíticas agrupan toda su funcionalidad en una única unidad ejecutable. Esto implica que ante la necesidad de realizar actualizaciones o escalabilidad el proceso se vuelve más complejo. Según [1] En este enfoque monolítico presenta las siguientes limitaciones:

- Siempre que se realiza una actualización, es necesario desplegar toda la aplicación.
- Es muy difícil que una aplicación monolítica pueda ser escalable en partes individuales del sistema, lo que hace que afecte negativamente la eficiencia del desarrollo del sistema.

Por otro lado, la arquitectura de microservicios ha emergido como una solución innovadora que divide las aplicaciones en módulos independientes, cada uno responsable de una función específica del sistema. Este enfoque modular no solo mejora la flexibilidad y la escalabilidad, sino que también facilita el mantenimiento y reduce los riesgos asociados con el acoplamiento entre componentes.

Sin embargo, a pesar de sus beneficios el desarrollo de aplicaciones basadas en microservicios enfrenta desafíos importantes particularmente en el cálculo de las métricas como la dependencia y complejidad.

La complejidad en la arquitectura de microservicios no solo es esencial, lo que es crucial para garantizar la flexibilidad y capacidad de adaptación del sistema ya que implica la fragmentación de aplicaciones en componentes autónomos lo cual permite integrar funcionalidades diversas, pero también exige una gestión rigurosa para evitar problemas de escalabilidad.

El cálculo de las métricas es indispensable para evaluar la calidad de las aplicaciones en distintas etapas de la aplicación. Ya que integra consideraciones como el tiempo de desarrollo y calidad de la aplicación, donde los desarrolladores pueden optimizar sus sistemas, logrando mayor eficiencia y calidad en periodos cortos de tiempo.

En este contexto, los microservicios representan una evolución significativa en el desarrollo de aplicaciones frente a las arquitecturas tradicionales. Este enfoque de modularidad permite superar ampliamente a la arquitectura monolítica el cual promueve las aplicaciones de mejor calidad y manteniendo su relevancia en entornos dinámicos.

2.1. SITUACIÓN PROBLEMÁTICA

En la actualidad las arquitecturas basadas en microservicios presentan desafíos significativos, siendo la evaluación de la calidad de estas arquitecturas, especialmente las características de dependencia y complejidad, algunas de las más relevantes. A diferencia de otros enfoques, estas métricas no reflejan un estándar general lo que dificulta su cálculo y genera discrepancias entre autores e investigadores del área.

Los sistemas de recomendaciones tradicionales, a menudo son desarrollados bajo una arquitectura monolítica, tienen limitaciones en cuanto a la escalabilidad, flexibilidad y mantenibilidad.

La arquitectura de microservicios se perfila como una alternativa prometedora para superar estas limitaciones, pero la falta de estandarización resalta la necesidad urgente de investigaciones más profundas que establezcan métodos claros y confiable para medir y evaluar estas métricas, permitiendo así un cálculo más preciso y una mayor calidad en los sistemas de microservicios.

2.2. FORMULACIÓN DEL PROBLEMA

¿Cómo evaluar la dependencia y complejidad en una arquitectura de microservicios, a través de un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia, aplicando una arquitectura de microservicios para comprobar las métricas de calidad de dependencia y complejidad?

2.3. OBJETO Y CAMPO DE ACCION

2.3.1. Objeto de investigación:

Evaluación de métricas de dependencia y complejidad.

2.3.2. Campo de Acción:

1203.17 Informática

2.4. BENEFICIARIOS

Tabla 3 Beneficiarios

Beneficiarios	*Cargo*	*Descripción*
Directos	*Usuarios Finales*	*Disfrutan de sistemas con mayor disponibilidad, rendimiento y capacidad de respuesta, gracias a la escalabilidad independiente de los microservicios que mejoran su experiencia general.*
Indirectos	*Clientes y Clientes Potenciales*	*Perciben una experiencia más fluida y confiable en los servicios ofrecidos, lo que genera mayor confianza en el sistema y mejora la satisfacción general, incluso sin interacción directa con los microservicios.*
	Equipo de Soporte Técnico	*Facilitan sus labores de mantenimiento y resolución de problemas gracias a la modularidad y claridad del sistema, reduciendo tiempos de inactividad y complejidad en la operación.*

2.5. JUSTIFICACIÓN

El presente trabajo ofrece un aporte teórico al definir un marco riguroso para medir dependencias y complejidad en arquitecturas de microservicios, relacionando dichas métricas con atributos de calidad como mantenibilidad, escalabilidad y tolerancia a fallos, y proponiendo un procedimiento replicable para comparar alternativas de diseño.

De manera complementaria, el aporte práctico se materializa en el desarrollo de un prototipo operativo de recomendación y descubrimiento de contenidos multimedia instrumentado para la captura de métricas.

La factibilidad del estudio radica en el uso de herramientas disponibles como MOAM y tecnologías estándar, en un enlace acotado y modular que permite replicar el prototipo en otros entornos.

Finalmente, el trabajo es importante porque la calidad de la arquitectura incide directamente en el rendimiento, la sostenibilidad y los costos operativos de las aplicaciones: además, aporta evidencia empírica y conocimiento transferible que puede guiar mejoras continuas y decisiones de diseño en el desarrollo de sistemas distribuidos modernos.

2.6. OBJETIVOS

2.6.1. Objetivo General

Desarrollar un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia, aplicando una arquitectura de microservicios, para comprobar las métricas de calidad de dependencia y complejidad.

2.6.2. Objetivos Específicos

- Realizar una revisión bibliográfica sobre el objeto y campo de estudio para obtener la fundamentación teórica del proyecto.
- Implementar el sistema de recomendación y descubrimiento de contenidos multimedia, aplicando prácticas ágiles, para obtener una arquitectura de microservicios.
- Comprobar las métricas de calidad de dependencias y complejidad, usando la herramienta MOAM (Microservices Optimization Arquitectures Model).

2.6.3. Sistema de tareas

Tabla 4: Planificación de las actividades

Objetivos Específicos	Actividades (tareas)	Resultados esperados	Técnicas, Medios e Instrumentos
Realizar una revisión bibliográfica sobre el objeto y campo de estudio para obtener la fundamentación teórica del proyecto.	Revisión de documentación teórica y científica sobre efectos de la implementación de arquitecturas de microservicios.	Fundamentación teórica.	Técnica: Revisión bibliográfica. Medios: libros, revistas, etc. Instrumentos: ficha bibliográfica
Implementar el sistema de recomendación y descubrimiento de contenidos multimedia aplicando prácticas ágiles.	Desarrollo de un prototipo utilizando los frameworks Django y React.	Prototipo funcional.	Técnica: XP, algoritmo de recomendación basado en contenido. Medios: Django y React. Instrumentos: Historias de usuarios.
Comprobar las métricas de calidad de dependencia y complejidad, usando la herramienta MOAM.	Realización de pruebas para evaluar métricas de calidad de dependencia y complejidad.	Informe técnico.	Técnica: Evaluación de métricas. Medios: Herramienta MOAM. Instrumentos: Reportes generado por MOAM, tablas de resultados.

Fuente: Los desarrolladores.

3. FUNDAMENTACIÓN TEÓRICA

3.1. Sistemas de recomendación

Un sistema de recomendación es una herramienta que se usa para sugiere productos o servicios al usuario. Su propósito principal es, a través de una serie de valoraciones y criterios sobre los datos del usuario o del ítem, predecir que productos o servicios podrían ser del agrado del usuario, con el fin de mejorar su experiencia.

Los sistemas de recomendación se usan en una variedad de industrias, como por ejemplo la publicidad, la música, los libros, los juegos, las series y el cine. En el caso del cine in sistema de recomendación de películas puede sugerir películas que se ajusten a los intereses y preferencias de los usuarios [2]. Los sistemas de recomendación no solo sirven para mostrar cosas que le gustan al usuario, sino que también ayudan a descubrir nuevas opciones que tal vez no conocía pero que podrían interesarle.

3.1.1. Tipos de sistemas de recomendación

La finalidad de un sistema de recomendación es predecir la valoración que un usuario va hacer de un ítem que todavía no ha evaluado. Esta valoración se genera al analizar una de dos cosas, o las características de cada ítem, o las valoraciones de cada usuario a cada ítem, y se usan para recomendar contenido personalizada a los usuarios. Existe varias clasificaciones de los sistemas de recomendaciones de las cuales se utilizó el sistema de recomendación basado en contenido [3]. La incorporación de un sistema de recomendación baso en contenido en este prototipo representa un avance significativo que la personalización de la experiencia del usuario.

3.1.1.1. Sistema de recomendación basado en contenido.

En los sistemas de recomendación basado en contenido, los elementos a sugerir se describen a partir del usuario conforme a los ítems que ha valorado previamente, identificando patrones en sus elecciones. A medida que el sistema recibe nuevas interacciones del usuario, este perfil se ajusta de forma dinámica, reflejando sus interese de manera progresiva y personalizada [3]. El uso de recomendaciones basadas en contenido en este trabajo permite una personalización progresiva y autónoma, lo que resulta útil en entornos donde se requiera independencia de otros usuarios. Este enfoque mejores la adaptabilidad del sistema ante nuevos usuarios y contenidos.

3.1.1.2. Tipo de algoritmo de inteligencia artificial utilizado

3.1.1.2.1. Content-Based Recommendation using Bag-of-Words and Cosine Similarity

El método de recomendación basado en contenido utiliza la técnica de **Bag-of-Words (Bolsa de palabras)** para representar el contenido de cada ítem. En esta técnica, se considera un documento (o ítem) como una colección de palabras sin importar el orden, pero su tomando en cuenta la frecuencia con la que aparecen. Esto permite transformar el texto en un vector numérico que refleja la importancia de cada palabra. Luego, para determinar que tal similares sin dos ítems, se utiliza la similitud coseno. Esta medida calcula el ángulo entre dos vectores en un espacio multidimensional. De esta manera el sistema recomienda aquellos ítems cuyo contenido es más aparecido a las preferencias del usuario [4]. Esta combinación de bolsa de palabras y similitud del coseno es una forma sencilla y efectiva de entender que le gusta a un usuario basándose únicamente en las palabras que describen los ítems. No necesita información de otros usuarios, por lo que es ideas para situaciones en donde no hay muchas valoraciones o cuando se quiere personalizar la recomendación rápidamente.

3.1.1.2.2. Funcionamiento de algoritmo.

3.1.1.2.2.1. Construcción de perfil de usuario

Se identifica los ítems que el usuario ha consumido previamente y se extraen sus características principales: (películas, libros, artículos), estas características se combinan para formar un vector que representa los gustos del usuario, es decir, un perfil personalizado con sus preferencias [4]. Es como armar una lista con todo lo que te gusta. El sistema noma nota de los temas o estilos que has disfrutado antes, y los guarda.

3.1.1.2.2.2. Representación de los ítems

Cada ítem del catálogo también se transforma en un vector, utilizando técnicas como Bag-of-Words o TF-IDF, que permite representar el contenido textual (Como descripciones, sinopsis o etiquetas) en forma numérica. Esto permite que el sistema compare ítems entre si y con los perfiles de los usuarios [4]. El sistema convierte todo lo que hay en su catálogo en un formato que se puede comprar fácilmente: como convertir libros o películas en números, según las palabras clases que contiene.

3.1.1.2.2.3. Cálculo de similitud

Se utilizo una métrica como la similitud del coseno para comparar el perfil del usuario como los ítems del sistema. Esta técnica mide el ángulo entre los vectores del usuario y del ítem: cuanto más pequeño sea ese ángulo, mayor es la probabilidad de que el ítem sea del gusto del usuario [4]. El sistema trata de ver que tan parecidos son tus gustos con cada cosa nueva del catálogo, y escoge lo que más se parece a ti.

3.1.1.2.2.4. Generación de recomendaciones

Los ítems con mayor puntuación de similitud son seleccionados como recomendaciones para el usuario. Este proceso se repite y el perfil del usuario se actualiza continuamente a medida que el sistema recibe nueva información sobre sus elecciones [4]. El sistema no solo te recomienda cosas una vez: va aprendiendo más de ti cada vez que usas la plataforma, y así mejorar sus sugerencias.

3.1.1.2.3. Elementos utilizados

3.1.1.2.3.1. Vectores de características

Las textos o metadatos asociados a cada ítem (Como títulos, descripciones, etiquetas) se convierten en vectores numéricos mediante técnicas como **Bag-of-Words** o **TF-IDF.** Estos vectores permiten representar el contenido de cada ítem de forma que pueda ser procesado matemáticamente por el sistema [4]. Se refiere a transformar palabras a números que indican que tan importantes son y así el sistema puede compararlas.

3.1.1.2.3.2. Técnicas de Machine Learning

Se utiliza algoritmos como **TF-IDF** para determinar la relevancia de cada termino en un ítem respecto al catálogo, y la similitud del coseno para comparar cuan cercanos son los vectores (ítem y usuario). Esto permite decidir qué tan recomendables es un ítem para un usuario especifico [4]. **TF-IDF** le dice al sistema que palabras con importantes. Luego, con la similitud del coseno, el sistema compara tu perfil con los ítems para ver cuales se parecen más a lo que te gusta.

3.1.1.2.3.3. Datos históricos del usuario

El sistema analiza el comportamiento pasado del usuario: valoraciones que ha dado, contenido que ha visto o en el que ha hecho clic. Esta información sirve para crear y ajustar

su perfil de preferencias a lo largo del tiempo [4]. El sistema aprende de lo que haces: Que vez, que calificas o que contenido reproduces, con el fin de entender mejor tus gustos y ajustar sus recomendaciones.

3.2. Microservicios

La arquitectura de microservicios es una forma de desarrollar aplicaciones dividiéndolas en partes pequeñas, llamadas servicios. Cada uno funciona por separado, con su propio proceso y se comunica con los demás por medio de conexiones simples, como APIs que se usan el protocolo HTTP. Estos servicios están diseñados para cumplir funciones específicas de un negocio y pueden ser desarrollados, implementados y actualizados de forma independiente y automática. No se necesita una administración centrar complicada, además puede estar programados distintos lenguajes y usan diferent4es bases de datos, este concepto no es totalmente nuevo ya que el termino fue mencionado por primera vez por Martin Fowler en un taller con arquitectos de software, donde notaron que estaban trabajando en un nuevo estilo de arquitectura, no existe una única definición oficial, pero una forma simple de entenderlos es como "pequeños servicios independientes que colaboran entre sí", a diferencia de las aplicaciones tradicionales (monolíticas), los microservicios permiten construir sistemas compuestos por modelos autónomos, lo que facilita su desarrollo, mantenimiento y escalabilidad. [5]. Una aplicación con microservicios se construye dividiéndola en partes pequeñas independientes. Cada parte hace una tarea especifica y funciona por separado, lo que permite que se pueden usar distintos lenguajes y tecnologías. Estos microservicios se comunican entre sí, lo importante de esto es que cada uno se puede actualizar o cambias sin afectar a los demás. Esta forma de trabajar es diferente a alas aplicaciones tradicionales, que se construyen como un solo bloque.

3.2.1. Desafío de los microservicios

Aunque los microservicios ofrecen múltiples ventajas, su adopción conlleva ciertos retos que deben ser cuidadosamente considerados al diseñar la arquitectura de las aplicaciones. Uno de los principales desafíos es la complejidad inherente a los sistemas distribuidos, donde se requiere seleccionar y aplicar mecanismos adecuados de comunicación entre servicios, manejar posibles fallos y enfrentar situaciones de indisponibilidad, además la gestión de transacciones distribuidas representa una dificultad importante, ya que coordina r y revertir cambios en operaciones empresariales

distribuidas pueden resultar complicado, lo que podría generar inconsistencias en los datos. [6].

3.2.2. Arquitecturas de microservicios

La arquitectura de microservicios se asemeja a un sistema de gobierno descentralizado, donde cada componente funciona de manera autónoma. Por ejemplo, puede tener su propia base de datos. Esto evita la sobrecarga de una unca base de datos y reduce el riesgo de que una falla afecte a toda la aplicación. Cuando varios servicios están interconectados, suelen incorporar mecanismos de alerta y respuesta entre fallos, como mostrar a los usuarios sobre interrupciones temporales. Además, se optimiza el flujo de información dirigido a cada módulo, lo que contribuye a una mejor administración de los recursos entre los distintos componentes [7].

Los microservicios son como un equipo donde cada miembro trabaja por su cuenta y tiene sus propias herramientas, en lugar de compartir una sola. Así, si algo falla en uno, los demás pueden seguir funcionando sin problema. Además, si un servicio tiene un error, hay alertas que avisan al equipo o a los usuarios para que tomen medidas sin afectar todo el sistema.

3.2.3. Arquitectura hexagonal

Según [8] la arquitectura hexagonal, también llamada arquitectura de puertos y adaptadores, es un modelo que busca dividir la aplicación en varias partes, donde cada una de estas partes cumplen un rol específico. Esto permite que las partes tengan menos conexiones ente sí, lo que facilita el mantenimiento y su evolución. Este tipo de arquitectura está compuesta por tres capas principales: infraestructura, aplicación y dominio.

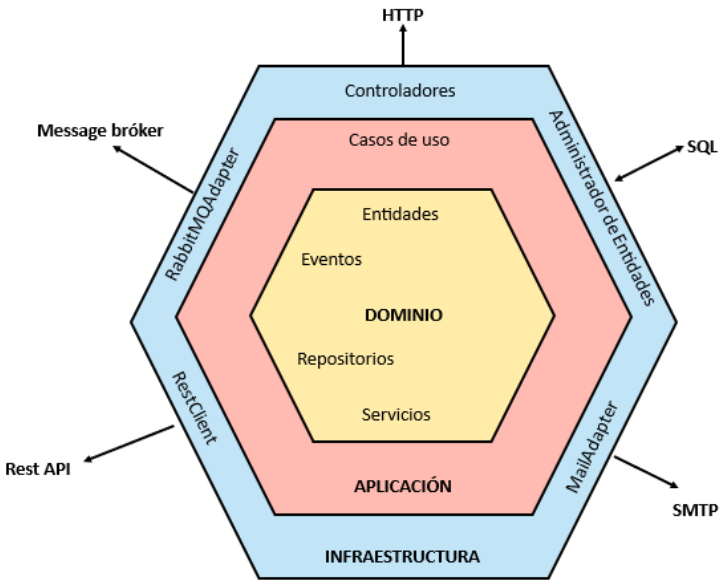


Figura 1 Arquitectura Hexagonal [8]

3.2.4. Arquitecturas de software

Según [9] la arquitectura de aplicaciones toma los requisitos de los clientes, los analiza y de esta manera diseña un software que satisfará sus necesidades, los mismos que deben cumplir con los principios de diseño y las técnicas de procedimiento que han evolucionado con el tiempo, también implica definir una solución totalmente organizada para cumplir las exigencias técnicas y operativas, buscando mejorar características clave como la calidad, el rendimiento, la seguridad y la facilidad de gestión.

3.2.5. Patrones de Diseño de Microservicios

Los patrones de diseño de microservicios (PDMS) son patrones de diseño de software que generan servicios autónomos reutilizables. Consiste en emplear estrategias y enfoques para el diseño e implementación de arquitecturas basadas en microservicios [10]. El objetivo es utilizar patrones que ayuden a manejar aspectos como la comunicación entre los servicios, el acceso y gestión de datos, la escalabilidad, etc.

3.2.5.1. Api Gateway

Una puerta de enlace API es una variante de un servicio de agregación que desempeña un papel fundamental en DevOps como "el habilitador de las API". Estas puertas de enlace son una solución para abordar desafíos como la obtención de información de múltiples microservicios y la gestión de solicitudes de protocolo simultáneas. Puede servir como el punto de entrada para todos los microservicios y por lo tanto los microservicios pueden

comunicarse entre sí a través de un servidor sin estado, utilizando solicitudes HTTP o un bus de mensajes [10].

3.2.6. ATRIBUTOS DE CALIDAD DE MICROSERVICIOS

Según [11], cada sistema busca cumplir con ciertas características que aseguren se buen funcionamiento, como la seguridad, la rapidez o la facilidad de uso, estas características, que son conocidas como atributos de calidad, son importantes por que ayudan a saber si el sistema realmente cumple con las expectativas de las personas que van hacer uso de ello.

Existen diferentes atributos que se toman en cuenta para calificar la calidad de los microservicios, pero según [11] los principales atributos que son relacionados con la arquitectura son la escalabilidad, la independencia y la mantenibilidad.

3.2.6.1. Escalabilidad

La escalabilidad es la capacidad de un sistema de microservicios para que tenga un correcto funcionamiento y se adapte al crecimiento cuando aumenten los usuarios o el trabajo, sin perder estabilidad [11]. Esto se logra agregando más recursos o instancias de los servicios cuando es necesario, sin que el sistema se vuelva lento o falle.

3.2.6.2. Independencia

Es la capacidad que posee los microservicios para funcionar de forma autónoma, permitiendo que casa servicios se desarrolló, despliegue y mantenga sin afectar a los demás [11].

3.2.6.3. Mantenibilidad

La mantenibilidad en microservicios se refiere a la capacidad que tiene el sistema para ser manipulado fácilmente, como puede ser para corregir errores, mejorar el rendimiento o adaptarse a posteriores actualizaciones [11].

3.2.7. MÉTRICAS DE CALIDAD DE MICROSERVICIOS

De acuerdo en el Institute of Electrical and Electronics Engineers (IEEE), una métrica se calidad de software es “una función cuyas entradas son datos de software y cuya salida es un valor numérico que puede ser interpretado como el grado en que el software posee un atributo dado que afecta su calidad”. Esta definición puede extenderse tanto a productos

como a servicios de software e infraestructura. Si se utiliza las métricas adecuadas, el objeto de evaluación, que en este caso es el software, podrá llegar a tener un menos número de falas en su funcionamiento [12]. Las métricas de calidad sirven para medir que tan bueno es un software. Si se eligen, ayuda a que el programa tenga menos errores. Hay varios tipos de métricas, pero la más usadas son las que revisan el producto final, para asegurarse de que funcione bien antes de entregarlo.

3.2.7.1. Granularidad

Se refiere al tamaño de un microservicio. Una gran granularidad implica que contiene servicios grandes y con muchas funcionalidades, mientras que una granularidad menos implica servicios pequeños y enfocados [13]. La granularidad ayuda a organizar mejor los microservicios, es más fácil mantenerlos y hacer cambios sin afectar a todo el sistema.

3.2.7.2. Líneas de Código (LOC)

Es una de las métricas que indica la cantidad de código escrito para un microservicio. No siempre representan con precisión su complejidad o funcionalidad, sirve como una forma práctica de comparar tamaños entre microservicios dentro de un mismo sistema [13]. Se refiera a medir el tamaño del código para ver qué tan grande es un microservicio, de la misma manera sirve para comparar y mantener un equilibrio entre ellos.

3.2.7.3. Interfaz Abierta (Open Interface)

Se refiere al número de operaciones o funciones que un microservicio expone a otros servicios o usuarios, mientras más interfaces tiene un servicio, mayor es la complejidad. Evaluar este atributo ayuda a identificar servicios con sobrecarga o mal definidos [13]. Ayuda a revisar cuantas funciones está cumpliendo, y con referencia a ellas observar si está realizando más de lo que debería.

3.2.7.4. Alta Cohesión (High Cohesion)

La cohesión mide que tan relacionadas están las funcionalidades de un microservicio. Un servicio tiene alta cohesión cuando todas sus funciones están enfocadas en una sola tarea o responsabilidad [13]. Ayuda a verificar si un microservicio es claro y ordenada, si todas sus funciones están conectadas y hacen lo mismo de esta manera es más fácil de entender y mantener.

3.2.7.5. Bajo Acoplamiento (Loose Coupling)

El acoplamiento se refiere al nivel de dependencia de un microservicio con otros. Un bajo acoplamiento es ideal, ya que el servicio puede funcionar de manera más independiente [13]. Mientras menos dependencia exista entre microservicios significa que si uno falla, los demás pueden seguir funcionando sin problemas.

3.2.7.6. Costo de Ejecución (Execution Cost)

Hace referencia al gasto económico de ejecutar un microservicio en una infraestructura especifica, a esto también le incluye costos por usar los recursos como CPU, memora y almacenamiento [13]. Saber cuánto cuesta ejecutar un microservicio ayuda a usar mejor los recursos y evitar gastar más de lo debido en la infraestructura.

3.2.7.7. Tiempo de Respuesta (Response Time)

Es el tiempo que tarda el microservicio en responder a una solicitud. Un menor tiempo de respuesta mejora la eficiencia del sistema y la experiencia del usuario [13]. Una respuesta rápida del sistema hará que los usuarios no tengan que esperar y por ende prefieran y confíen más en dicho sistema,

3.2.7.8. Disponibilidad (Availability)

Indica el tiempo en que un microservicio está disponible para ser usado durante un tiempo determinado [13]. Cuando un microservicio está disponible la mayor parte del tiempo hace que el sistema sea más confiable.

3.2.7.9. Tasa de Ejecución Exitosa (Successful Execution Rate)

Mide que tan frecuente el microservicio completa correctamente las tareas que se le asignan, sin errores o fallos [13]. Un microservicio al cumplir bien sus tareas siempre hace que el sistema funcione sin problema alguno.

3.2.7.10. Frecuencia de Uso (Usage Frequency)

Refleja que tan utilizado es el microservicio en comparación con los demás del sistema. Un alto valor indica que es un componente central o muy solicitado [13]. Si tiene un alto porcentaje de uso significa que es muy importante para el sistema y debe mantenerse siempre funcionando correctamente.

3.2.7.11. Escalabilidad (Scalability)

Es la capacidad del microservicio para seguir funcionando correctamente, aun cuando aumente el número de usuarios o la carga de trabajo. Se puede escalar vertical u horizontalmente [13]. La escabilidad evita que el sistema se vuelva lento o colapse cuanto muchos usuarios lo utilicen al mismo tiempo.

3.2.7.12. Independencia (Independence)

Se refiere a cuando un microservicio es independiente puede ser desarrollado, implementado y gestionado sin afectar ni depender de otros, esto ayuda a tener una mejor flexibilidad y facilidad al realizar cambios [13]. Al ser independiente se puede mejorar o arreglar sin detener todo el sistema lo que ahorra tiempo y evita errores en otros servicios.

3.2.7.13. Mantenibilidad (Maintainability)

Se encarga de medir la facilidad con que se puede realizar cambios, correcciones o mejoras en el microservicio sin que afecte su funcionalidad general [13]. Mientras más fácil sea de modificar, se podrá hacer mejoras más rápido y con menos riesgo de causar errores en su funcionamiento.

3.2.7.14. Despliegue (Deploymente)

Se refiera a la disponibilidad que facilita los microservicios al poner en marcha el servidor. Un buen despliegue debe ser rápido, automatizado y repetible [13]. Un despliegue fácil y rápido permite actualizar el sistema.

3.2.7.15. Gestión de Salud (Health Management)

Describe la capacidad del microservicio para recuperarse de fallos sin perder datos. Incluye mecanismos como reinicio automático o respaldo de estado [13]. Si un microservicio se recupera bien de fallos el sistema sigue funcionando y así no pierde datos importantes.

3.2.7.16. Modularidad (Modularity)

Muestra el grado en que un microservicio sigue el principio de responsabilidad única, separando las distintas funcionalidades del sistema [13]. Cumplir con una sola función ayuda a que un microservicio sea más claro y fácil de mantener.

3.2.7.17. Gestionabilidad (Manageability)

Se encarga de ver que tan fácil es monitorear, auditar y controlar el comportamiento del microservicio con herramientas automatizadas o mínimas intervenciones [13]. Un microservicio fácil de monitorear permite detectar problemas más rápido y así mantener el sistema fusionando correctamente

3.2.8. RANGO DE EVALUACIÓN DE MÉTRICAS DE CALIDAD DE DEPENDENCIA Y COMPLEJIDAD.

3.2.8.1. Métricas de calidad evaluadas

Se evaluará los resultados de las métricas obtenidas en la herramienta MOAM en la cual intervine los cálculos de la dependencia y la complejidad.

3.2.8.1.1. Métricas de dependencia

Las métricas centradas en la dependencia se encargan de evaluar que tan seguras y estables son las conexiones entre los distintos microservicios dentro de la arquitectura.

Rango de número de dependencias externas

Esto tomara en cuenta la cantidad de otros microservicios o sistemas externos se los que depende un microservicio. Los rangos se encuentran en la tabla

Tabla 5. Interpretación de dependencias externas [14].

Rango	Interpretación de dependencias externas
(0-2)	Bajas dependencias externas
(3-5)	Dependencias externas moderadas
(6 o más)	Alta dependencia externa

Fuente: Los desarrolladores.

Numero de dependencias Internas

Hace referencia a la cantidad de módulos o componentes internos (dentro del mismo sistema) de los que depende un microservicio.

Tabla 6. Interpretación de dependencias internas [14].

Rango	Interpretación de dependencias internas
(0-5)	Bajas dependencias internas
(6-11)	Dependencias internas moderadas
(11 a más)	Alta dependencia interna

Fuente: Los desarrolladores.

Profundidad de dependencias

Evaluará cuantos niveles de relaciones hay entre los microservicios, que tan lejos o profundo llega una dependencia entre otros servicios.

Tabla 7. Interpretación de profundidad de dependencias [14].

Rango	Interpretación de profundidad de dependencias
(0-2)	Baja profundidad de dependencias
(3-5)	Profundidad de dependencias moderadas
(6 o más)	Alta profundidad de dependencias

Fuente: Los desarrolladores.

Acoplamiento Estimado

Tomará en cuenta el nivel de conexión entre microservicios y otros.

Tabla 8. Interpretación de acoplamiento [14].

Rango	Interpretación de acoplamiento
(0-3)	Bajo acoplamiento
(4-6)	Acoplamiento moderado
(7 o más)	Alto acoplamiento

Fuente: Los desarrolladores.

Falta de Cohesión estimada.

Evalúa que tan poco relacionadas están las funciones dentro de un microservicio.

Tabla 9. Interpretación de falta de cohesión [14].

Rango	Interpretación falta de cohesión
(0.5 – 1.0)	Baja falta de cohesión
(1.1 – 2.0)	Falta de cohesión moderada
(2.1 o más)	Falta de cohesión alta

Fuente: Los desarrolladores.

Dependencia Estimada

Medirá que tanto depende un microservicio de otros, tanto interna como externa.

Tabla 10. Interpretación de dependencia estimada [14].

Rango	Interpretación dependencia estimada
(5 – 8)	Baja dependencia estimada
(9 – 12)	Dependencia estimada moderada
(13 o más)	Alta dependencia estimada

Fuente: Los desarrolladores.

3.2.8.1.2. Métricas de complejidad

Servirán para identificas las partes más complejas en la arquitectura.

Complejidad Ciclomática Estimada

Servirá para determinar la complejidad acorde al número de caminos posibles, es decir ayudará a evaluar la dificultad de la arquitectura en base a sus dependencias internas.

Tabla 11. Interpretación de complejidad ciclomática [14].

Rango	Interpretación complejidad ciclomática
(3 – 5)	Complejidad ciclomática baja
(6 – 8)	Complejidad ciclomática moderada
(9 o más)	Complejidad ciclomática alta

Fuente: Los desarrolladores.

Coeficiente de Acoplamiento Estimado

Es un indicador del grado de acoplamiento entre microservicios tomando en cuenta toda la arquitectura para evaluar su modularidad.

Tabla 12. Interpretación de coeficiente de acoplamiento [14].

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

Fuente: Los desarrolladores.

Complejidad Estructural Estimada

Determinará la complejidad de toda la arquitectura combinando el número de microservicios y sus dependencias internas.

Tabla 13. Interpretación de complejidad estructural [14].

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

Fuente: Los desarrolladores.

Puntos de Historias Estimados

Son el número total de funcionalidades que se deberán cumplir, para estimar el esfuerzo.

Complejidad Estimada

Hace referencia a la complejidad total de la arquitectura, tomando en cuenta todas las métricas de complejidad, para tener una medida global de la arquitectura.

Tabla 14. Interpretación de complejidad estimada [14].

Rango	Interpretación de complejidad estimada
(30 – 45)	Complejidad baja
(46 – 60)	Complejidad moderada
(61 o más)	Complejidad alta

Fuente: Los desarrolladores.

Índice Agregado Estimado

Resume la calidad general obtenida en la arquitectura.

Tabla 15. Interpretación de índice agregado [14].

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

Fuente: Los desarrolladores.

3.3. MOAM

El modelo denominado "MOAM" (Microservices Optimization Arquitectures Model), se basa en la implementación de algoritmos inteligentes que a partir de las entradas que son las historias de usuarios de un caso, genera el diagrama arquitectónico de los microservicios [15].

Posteriormente, utiliza una serie de métricas para calcular de manera automática la dependencia y complejidad de la arquitectura, finaliza con un reporte de los cálculos correspondientes y una serie de recomendaciones que fueron determinadas a través de umbrales definidos en los resultados de las diferentes métricas [15].

El modelo inteligente para especificas dependencia y complejidad en el diseño de aplicaciones de microservicios pretende brinda apoyo a los equipos de desarrollo, en la toma decisiones de carácter arquitectónico para prevenir futuros problemas de implementación, mantenibilidad, flexibilidad y escalabilidad [15]. MOAM ayuda a diseñar mejor las aplicaciones con microservicios, detectando desde el inicio si la arquitectura será muy complicada o difícil de mantener. Así, evita errores futuros y mejora la calidad del sistema.

3.4. DESARROLLO DE SOFTWARE

Según Noriega [16] Afirma que: "La ingeniería de Software es una forma (..) que aplica los principios de la ciencia de la computación y de la matemática para alcanzar soluciones con una mejor relación entre el coste y el beneficiario para el problema de software". Sin embargo, se puede mencionar que este autor nos da a comer un poco más acerca del proceso del mismo, en la cual Inicialmente, las limitaciones del hardware restringían el tamaño y la complejidad de los programas.

Sin embargo, a medida que avanzaba la potencia informática, el software evolucionó hacia sistemas más grandes y atractivos. Para gestionar esta creciente complejidad, surgieron diversas técnicas, como el desarrollo de lenguajes de programación especializados, el estudio más profundo de la ingeniería de software, la arquitectura de software y el uso de herramientas de ingeniería asistida por ordenador (CASE).

Los orígenes de la crisis del software se remontan a mediados del siglo pasado, cuando las limitaciones tecnológicas dificultaban la creación de programas robustos y escalables. Desde entonces, la industria ha buscado diversas soluciones para afrontar estos retos, como el desarrollo de nuevas metodologías, herramientas y lenguajes de programación. Sin embargo, la predicción precisa de costes y tiempos sigue siendo un objetivo difícil de alcanzar [16].

3.4.1. Modelos de desarrollo de software

Al pasar el tiempo se vio importante tratar de encontrar un mejor proceso metodológico accesible que mejore sobre todo la calidad para cada uno de los usuarios y su productividad siga creciendo, ya que varios modelos fueron ideados con el único objetivo de poder organizar un mejor proceso de desarrollo de Software, que sea constante, y que atenga alta validez del mismo [17].

Modelo tradicional en cascada: es uno de los modelos más sencillos que puede existir, ya que las fases encontradas dentro de esta cascada suelen ser ejecutadas de forma secuencial y preciso [17].

De la misma manera se puede mencionar que existe un Modelo en Fuente, se inspira en la modelo cascada, pero introduce un elemento iterativo, es decir, el proceso de desarrollo no es lineal, sino que implica ciclos de repetición en los que se revisan y perfeccionan las fases anteriores. Cada uno de estos son de suma importancia ya que depende de qué manera se representen que sea viable y eficaz [17].

3.4.2. Desarrollo interactivo e incremental

El desarrollo iterativo es un enfoque que promueve la creación de versiones iniciales de software para obtener retroalimentación temprana y realizar ajustes continuos. Esta metodología, al permitir una adaptación constante a las necesidades de los usuarios,

reduce significativamente el riesgo de desarrollar un producto que no cumpla con las expectativas [18].

3.4.3. Desarrollo de aplicaciones basadas en microservicios

Los microservicios representan una forma de organizar y estructurar el desarrollo de software, donde este se divide en componentes pequeños y autónomos que se conectan mediante APIs claramente definido. Cada uno de estos servicios es gestionado por equipos reducidos e independientes [19].

Sin embargo, se puede mencionar que las arquitecturas de microservicios hacen que las aplicaciones sean más fáciles de escalar y más rápidas de desarrollar. Esto permite la innovación y acelera el tiempo de comercialización de nuevas funciones [19].

3.4.3.1. Arquitectura monolítica en comparación con la arquitectura de microservicios.

A diferencia de las arquitecturas de microservicios, donde las aplicaciones se dividen en servicios independientes, las arquitecturas monolíticas presentan una estructura más rígida y menos flexible. En un monolito, todos los componentes están estrechamente acoplados, lo que dificulta la escalabilidad, la actualización y la adopción de nuevas tecnologías. Cualquier cambio en el sistema requiere una reimplementación completa, lo que puede generar tiempos de desarrollo más prolongados y un mayor riesgo de errores [20].

Sin embargo, este desarrollo de microservicio, es una aplicación que se crea componentes independientes que ejecutan un proceso de cada una de las aplicaciones como un servicio eficaz para los clientes, ya que estos productos o servicios se comunican a través de una interfaz bien complementada mediante una API muy ligeras.

3.4.4. Metodologías ágiles

Según [21], las metodologías ágiles se caracterizan por su flexibilidad, lo que permite ser adaptadas a las necesidades específicas de cada equipo o proyecto. En lugar de abordar el desarrollo como un todo, se divide en partes más pequeñas organizadas en la lista priorizada de funcionalidades. Cada una de estas partes se trabaja por separada en ciclos cortos que duran entre dos a seis semanas, durante todo este proceso la comunicación con el cliente es continua, incluso suele contar con un representante del cliente dentro del

equipo. Este enfoque impulsa la colaboración constante y la flexibilidad a diferentes cambios.

Algunas de las metodologías más comunes son:

Scrum: Se trata de un entorno de trabajo para facilitar la cooperación efectiva entre equipos en proyectos, mediante un conjunto de normas y la asignación de roles que establecen la estructura adecuada para su bien desempeño [21].

Programación Extrema (XP): Es una de las metodologías más conocida, ya que tiene como base 5 valores: Simplicidad, Comunicación, Retroalimentación, Respeto y Coraje [21].

3.4.5. Prácticas ágiles

Para [22], las practicas ágiles surgieron como una respuesta a los métodos tradicionales de desarrollo de software, con el objetivo de obtener resultados más rápidos sin comprometer la calidad. Estas prácticas ágiles ponen un fuerte énfasis en las personas involucradas en el proceso y en la entrega continua de pequeñas mejoras del producto a través de ciclos breves, ya que es clave para generar valor. Dado que los objetos de aprendizaje son productos de software que requiere un trabajo creativo e intelectual para obtener su diseño, además están destinados a un ámbito educativo que puede beneficiarse significativamente de las ventajas que ofrece la agilidad.

Las principales prácticas de la metodología ágil, un enfoque iterativo y colaborativo para el desarrollo de software. Estas prácticas se centran en la entrega frecuente de valor al cliente, la adaptación al cambio y la mejora continua. Las reuniones diarias breves facilitan la coordinación del equipo y la resolución de problemas. La priorización de tareas y los ciclos de desarrollo cortos permiten entregar funcionalidades de forma incremental. Además, el desarrollo ágil enfatiza la importancia de escribir pruebas unitarias para garantizar la calidad del código y la participación activa del cliente a través de historias de usuario.

Énfasis en la colaboración: La metodología ágil promueve la colaboración estrecha entre los miembros del equipo y el cliente, a través de prácticas como reuniones diarias y la creación de historias de usuario, permitiendo una mayor flexibilidad y adaptabilidad al cambio [23].

Énfasis en la calidad: El desarrollo ágil garantiza una alta calidad de software mediante la implementación de prácticas como el desarrollo basado en pruebas y la refactorización continua del código, lo que da como resultado productos más robustos y fáciles de mantener [24].

Énfasis en la velocidad: La metodología ágil permite una entrega de software más rápida y frecuente, gracias a la división del trabajo en ciclos cortos y la automatización de tareas repetitivas [25].

3.4.6. Herramientas de desarrollo

La ingeniería de software se ha beneficiado enormemente del desarrollo de herramientas especializadas que permiten a los desarrolladores crear aplicaciones de alta calidad de manera más eficiente. Estas herramientas van desde editores de código avanzados hasta plataformas de implementación en la nube y son esenciales para cualquier proyecto de desarrollo moderno [26].

Según ISIL [27] Afirma que: "Una herramienta de desarrollo de software es un programa, aplicación o conjunto de utilidades que los desarrolladores emplean para crear, depurar, probar y mantener programas informáticos". Sin embargo, esto se puede mencionar de tal manera que el dinámico panorama tecnológico actual, las empresas de software deben acelerar sus ciclos de desarrollo para mantenerse competitivas. Atrás quedaron los días de los lanzamientos anuales; ahora se requiere una entrega continua de software. Para satisfacer esta demanda, las organizaciones están adoptando prácticas innovadoras como el desarrollo ágil, DevOps y la arquitectura de microservicios, que permiten un desarrollo e implementación de aplicaciones más rápidos y flexibles.

Di Francesco y sus colaboradores [28] han llevado a cabo un análisis exhaustivo de la literatura científica sobre arquitectura de microservicios, identificando tendencias de investigación y evaluando el potencial de aplicación de este conocimiento en entornos industriales. Por su parte, Vural y su equipo [29] han llevado a cabo una revisión sistemática de la literatura para comprender la evolución de los microservicios y las áreas que requieren más investigación.

La arquitectura de microservicios ha surgido como una alternativa innovadora a los sistemas monolíticos tradicionales. Este enfoque modular, que divide una aplicación en servicios pequeños e independientes, ha sido objeto de mucha investigación en los

últimos años. Estudios recientes se han centrado en identificar los desafíos y oportunidades asociados con el desarrollo y la gestión de microservicios.

4. HIPOTESIS

El desarrollo de un prototipo de sistema de recomendación y descubrimiento de contenidos multimedia permitirá comprobar las métricas de dependencia y complejidad en su arquitectura.

5. MÉTODOS Y PROCEDIMIENTOS

5.1. TIPOS DE INVESTIGACIÓN

5.1.1. Investigación bibliográfica

Se busco y reviso información importante sobre temas como los microservicios, los sistemas de recomendación y como se organiza una arquitectura con microservicios, esta búsqueda fue muy útil ya que ayudo a entender mejor que es, como funciona y para qué sirve cada uno de estos conceptos. Gracias a eso, fue posible tener una base clara antes de empezar el proyecto, lo que permitió tomar buenas decisiones al momento de diseñar y construir el sistema.

5.1.2. Investigación exploratoria

La investigación exploratoria sirvió para conocer el tema a desarrollar, entender los conceptos y descubrir como otras personas han trabajado en proyectos parecidos, con esta investigación, se pudo tener una idea más clara del problema, encontrar posibles soluciones y definir bien el enfoque del trabajo.

5.2. MÉTODOS

5.2.1. Métodos de investigación bibliográfica

5.2.1.1. Método Analítico

Se utilizo para descomponer el sistema desarrollado en microservicios, analizando cada uno de ellos por separado en el fin de comprender sus funciones, su grado de dependencia y su complejidad.

5.2.1.2. Método Inductivo

Al desarrollar un prototipo de sistema de recomendación de contenido multimedia basado en microservicios se observó el comportamiento del prototipo, medimos métricas de calidad como dependencia y la complejidad utilizando la herramienta MOAM, con los resultados obtenidos identificamos patrones y formulamos conclusiones generales sobre la calidad de la arquitectura implementada.

5.2.1.3. Método Deductivo

Se utilizo a partir del marco teórico principales sobre la calidad de software, arquitectura de microservicios y metodologías ágiles, con ello desarrollamos el prototipo propuesto y posteriormente se realizó a la comprobación de su calidad avaluando métricas de dependencia y complejidad.

5.2.2. Método de investigación exploratoria

5.2.2.1. Identificar problema

Esta etapa permitió identifica el problema a partir de la falta de mecanismos para evaluar la calidad en sistemas de recomendación basados en microservicios, específicamente en cuanto a las métricas de dependencia y complejidad.

5.2.2.2. Establecer hipótesis

La hipótesis nos permitió anticipar que un sistema de recomendación de contenido multimedia desarrollado con una arquitectura de microservicios bien estructurada daría como resultado métricas aceptables de dependencia y complejidad.

5.2.2.3. Fundamentación de investigaciones posteriores

Este apartado permitió revisar estudios previos relacionados con sistemas de recomendación, arquitectura de microservicios y métricas de calidad como dependencia y complejidad.

5.3. TÉCNICAS DE INVESTIGACIÓN

5.3.1. Revisión bibliográfica

La revisión bibliografía se desarrolló mediante la selección de fuentes especializadas que aportan información actual y relevantes sobre arquitecturas de microservicios y si implementación en sistemas distribuidos. Para ello, se utilizaron criterios de pertinencia, actualidad y confiabilidad de las fuentes, priorizando literatura académica y científica publicada en revistas, conferencias, libros de referencia y repositorios digitales. Este proceso permitió organizar la información de manera sistemática para su posterior análisis, estableciendo así el marco conceptual y teórico del proyecto.

5.4. INSTRUMENTOS DE INVESTIGACIÓN

5.4.1. Ficha bibliográfica

La ficha bibliográfica facilita la organización y el manejo de la información obtenida durante la revisión de literatura. Cada ficha incluye datos relevantes como el auto, título de la obra, año de publicación, editorial y lugar de edición. Este instrumento permitió sistematizar la información, evitando omitir referencias importantes y asegurando la correcta citación de los documentos utilizados en la investigación.

5.5. TÉCNICAS ESPECÍFICAS

5.5.1. XP

En este proyecto, XP se utilizo para organizar el trabajo en iteraciones cortas, priorizando la comunicación constante con los usuarios y la retroalimentación continua. Sus prácticas, como la programación incremental, la integración continua y las pruebas unitarias, permitieron garantizar que el sistema de recomendación y descubrimiento de contenidos multimedia cumpliera con los requerimientos definidos, reduciendo el riesgo de errores y facilitando la adaptabilidad ante cambios.

5.5.2. Algoritmo de recomendación basado en contenido

Esta técnica se centró en él uso de un algoritmo de filtrado basado en contenido para la generación de recomendaciones personalizadas. Su funcionamiento consiste en analizar las características o tributos de los elementos multimedia disponibles y compararlos con

el perfil o preferencias del usuario. De esta manera, el sistema es capaz de sugerir contenidos similares a los que el usuario ya ha consumido o ha mostrado interés. En el desarrollo del prototipo, este algoritmo permitió ofrecer recomendaciones precisas, mejorando la experiencia del usuario y validando el enfoque propuesto para el sistema.

5.5.3. Evaluación de métricas

La evaluación de métricas se utilizó como técnica para comprobar la calidad de la arquitectura de microservicios implementada en el prototipo. Mediante el uso de la herramienta MOAM, se calcularon métricas de dependencias y complejidad. Esta técnica permitió obtener indicadores objetivos que fueron registrados en reportes automáticos y tablas de resultados, los cuales facilitaron el análisis y la interpretación de la calidad arquitectónica.

6. ANÁLISIS Y DISCUSIÓN DE LOS RESULTADOS

6.1. RESULTADOS DE LA METODOLOGÍA XP

6.1.1. Metodología XP.

Se utilizó XP porque es un marco de trabajo ágil orientado al desarrollo y mantenimiento de productos complejos.

Este enfoque es fundamentado en la teoría de control de procesos empíricos, la cual sostiene que el conocimiento se adquiere a través de la experiencia y la toma de decisiones basadas en lo conocido. Además, utiliza un método iterativo e incremental que permite mejorar la previsibilidad y gestionar mejor los riesgos.

6.1.1.1. Product backlog

El product backlog es una lista ordenada de todo lo que se necesita en el sistema, en este caso se ha utilizado 16 historias de usuarios de la cual se realizó el análisis de cada uno de ellos y partiendo de eso se obtendrá la prioridad que se clasifica en (Alta, media y baja), tomando en cuenta que los elementos más importantes se muestran al principio para que el equipo de trabajo sepa que hay que entregar primero y la estimación (Story time).

6.1.1.2. Técnicas de priorización Moscow

Es un método que se utilizó para clasificar y priorizar tareas de este proyecto tomando en cuenta que el método Moscow tiene 4 categorías que son:

- Must-Have (Debe tener)
- Should-Have (Debería tener)
- Could-Have (Podría tener)

Con referencia a estas categorías se ha identificado las tareas por realizar y asignarlas a cada categoría correspondiente basándose en su importancia y necesidad para tener un buen proyecto.

6.1.1.3. Técnicas de estimación

Se utilizo técnicas de estimación ya que cada actividad tendrá un tiempo determinado para poder realizarlas y así tener un tiempo más optimizado de desarrollo, en este caso se tomó en cuenta realizar cada historia de usuario por semanas.

Tabla 16. Historias de usuarios.

Prioridad	Historia de usuario	Descripción	Estimación
Alta	HU05	Como usuario, quiero registrarme y seleccionar mis gustos musicales.	1 semana
Alta	HU02	Como usuario, quiero recibir recomendaciones automáticas de música relacionada con mis intereses.	2 semanas
Alta	HU01	Como usuario, quiero buscar música utilizando un filtro para encontrar contenido específico.	2 semanas
Media	HU03	Como usuario, quiero explorar contenido nuevo basado en mis intereses musicales.	1 semana
Media	HU04	Como usuario, quiero descubrir nuevos géneros musicales	2 semanas

Prioridad	Historia de usuario	Descripción	Estimación
		basándome en etiquetas de contenido.	
Media	HU08	Como usuario, quiero ver mi historial de reproducciones para revisar el contenido que he escuchado.	2 semanas
Media	HU09	Como usuario, quiero agregar música o videos a mi lista de favoritos.	2 semanas
Baja	HU06	Como usuario, quiero que las etiquetas seleccionadas por mi estén disponibles para los demás usuarios.	1 semana
Baja	HU07	Como usuario, quiero actualizar mis datos ingresados en el registro	
Media	HU12	Como administrador, quiero ver todos los usuarios registrados	1 semana
Baja	HU10	Como usuario, quiero poder subir mis propios archivos de audio y video.	1 semana
Baja	HU11	Como usuario, quiero visualizar una lista de mis archivos multimedia (Audio y video).	1 semana
Alta	HU13	Como administrador, quiero subir contenido multimedia de audio y video con etiquetas detalladas para facilitar su búsqueda.	2 semanas
Baja	HU14	Como administrador, quiero eliminar contenido multimedia.	1 semana
Baja	HU15	Como administrador, quiero ver las estadísticas de los usuarios registrados	1 semana
Baja	HU16	Como administrador, quiero ver las estadísticas de la cantidad de	1 semana

Prioridad	Historia de usuario	Descripción	Estimación
		contenido multimedia asociado a la etiqueta "Mis Favoritos".	

Fuente: Los desarrolladores.

6.2. RESULTADOS DE LA EVALUACIÓN DE METRICAS DE DEPENDENCIA Y COMPLEJIDAD.

6.2.1. Arquitectura de microservicios

Se utilizo la herramienta MOAM para poder obtener la arquitectura correspondiente, mediante las historias de usuario mencionadas anteriormente, pero traducidas al inglés.

Mostró los siguientes resultados que se presenta en la Figura 2.

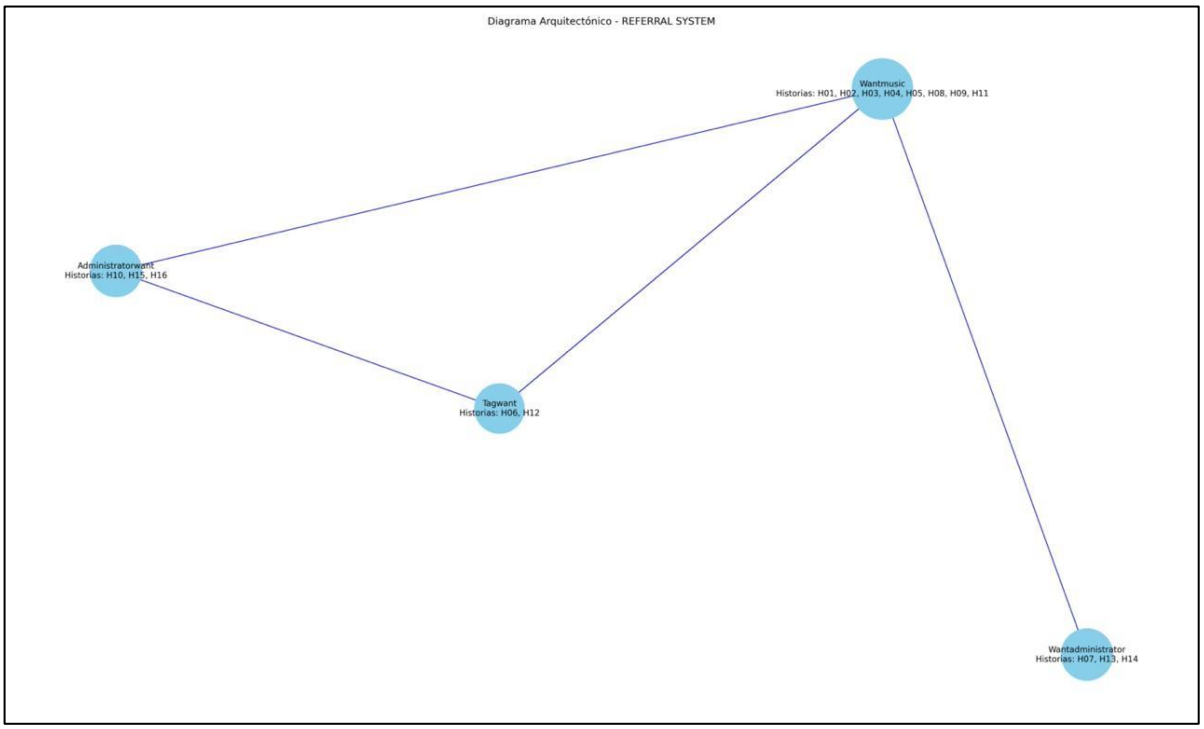


Figura 2. Arquitectura diseñada por la herramienta MOAM [15].

6.2.2. Clasificación de historias

Después de obtener la arquitectura, se clasifico las historias de usuarios con sus microservicios correspondientes, como se muestra en las siguientes Tablas

Tabla 17. Microservicio Wantmusic.

Microservicio	Historias de usuarios	Descripción
Wantmusic	HU1	Como usuario, quiero buscar música utilizando un filtro para encontrar contenido específico.
	HU2	Como usuario, quiero recibir recomendaciones automáticas de música relacionada con mis intereses.
	HU3	Como usuario, quiero explorar contenido nuevo basado en mis intereses musicales.
	HU4	Como usuario, quiero descubrir nuevos géneros musicales basándome en etiquetas de contenido.
	HU5	Como usuario, quiero registrarme y seleccionar mis gustos musicales.
	HU8	Como usuario, quiero ver mi historial de reproducciones para revisar el contenido que he escuchado.
	HU9	Como usuario, quiero agregar música o videos a mi lista de favoritos.

	HU11	Como usuario, quiero visualizar una lista de mis archivos multimedia (Audio y video).

Fuente: Los desarrolladores.

Tabla 18. Microservicio Tagwant.

Microservicio	Historias de usuarios	Descripción
Tagwant	HU6	Como usuario, quiero que las etiquetas seleccionadas por mi estén disponibles para los demás usuarios.
	HU12	Como administrador, quiero ver todos los usuarios registrados

Fuente: Los desarrolladores.

Tabla 19. Microservicio Administratorwant

Microservicio	Historias de usuarios	Descripción
Adiministratorwant	HU10	Como usuario, quiero poder subir mis propios archivos de audio y video.
	HU15	Como administrador, quiero ver las estadísticas de los usuarios registrados.
	HU16	Como administrador, quiero ver las estadísticas de la cantidad de contenido multimedia asociado a la etiqueta "Mis Favoritos".

Fuente: Los desarrolladores.

Tabla 20. Microservicio Wantadministrator.

Microservicio	Historia de usuarios	Descripción
Wantadministrator	HU7	Como usuario, quiero actualizar mis datos ingresados en el registro
	HU13	Como administrador, quiero subir contenido multimedia de audio y video con etiquetas detalladas para facilitar su búsqueda.
	HU14	Como administrador, quiero eliminar contenido multimedia.

Fuente: Los desarrolladores.

Junto a la clasificación anterior se generó un archivo, en el cual detalla toda la estructura en formato JSON, tomando en cuenta varios puntos importantes para la evaluación de calidad de dependencia y complejidad, el archivo tiene la siguiente estructura mostrará en la Figura 3.

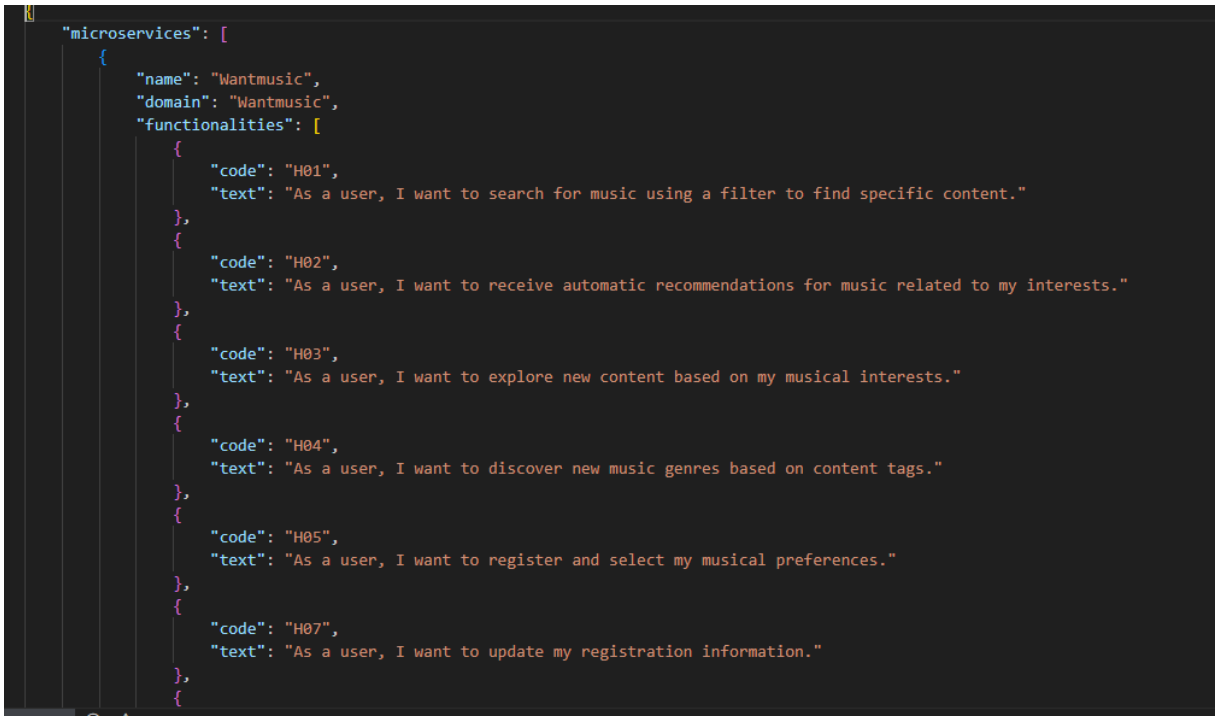


Figura 3. Formato JSON generada por la herramienta MOAM [15].

En esta Figura se especifica el nombre del servicio, su dominio y las historias de usuarios correspondientes, lo más importante es el número de dependencias tanto internas como externas que esto servirá para el cálculo de las métricas.

6.2.3. Métricas evaluadas en la herramienta MOAM.

Una vez obtenida la arquitectura se evalúa mediante el segundo componente de la herramienta MOAM (Aplicar el Modelo de Evaluación), en el cual se ingresa el formato JSON y empieza a realizar la evaluación, el resultado se muestra en la Figura 4.

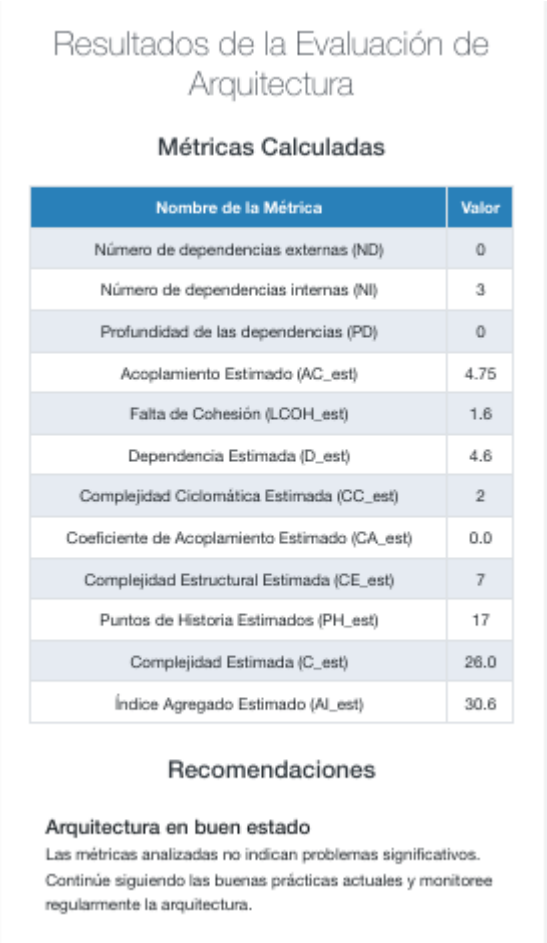


Figura 4. Resultados de la Evaluación de Arquitectura [15].

Todos estos resultados son basados en la arquitectura diseñada en la herramienta MOAM para su posterior comprobación mediante el desarrollo del prototipo, comprobando la eficacia de las métricas de calidad de dependencia y complejidad.

6.2.4. DESPLIEGUE

Para el despliegue se tomará en cuenta los siguientes pasos para poder tener un despliegue adecuando.

1. Abrir putty y colocar la IP: 94.130.77.86

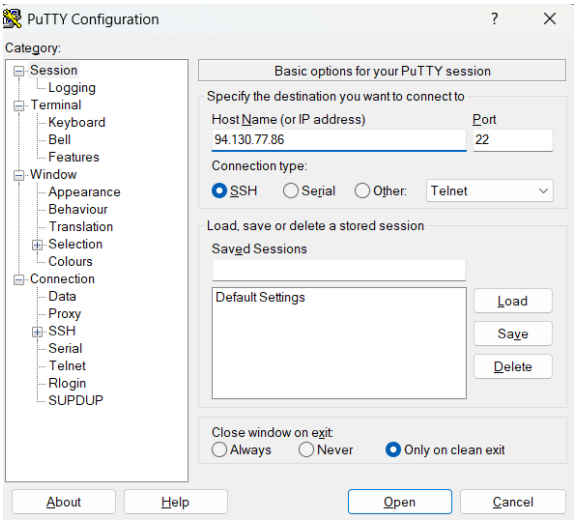


Figura 5. Confuguración Putty.

2. Loguearse
3. **User:** root

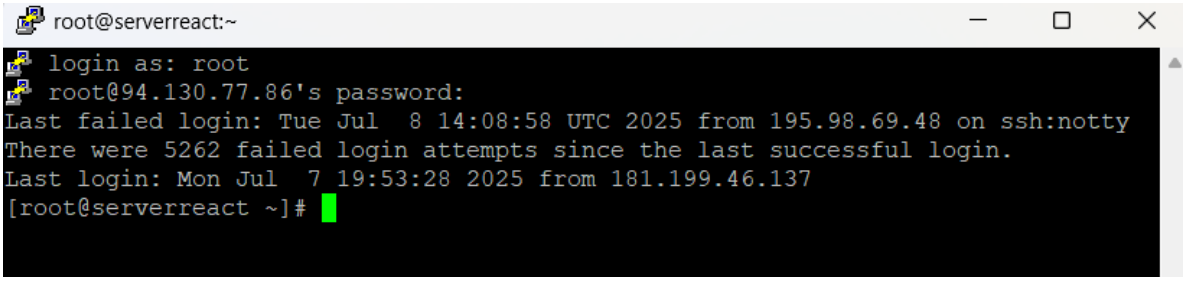


Figura 6. Logeo.

4. Actualizar con el comando: sudo dnf update –y

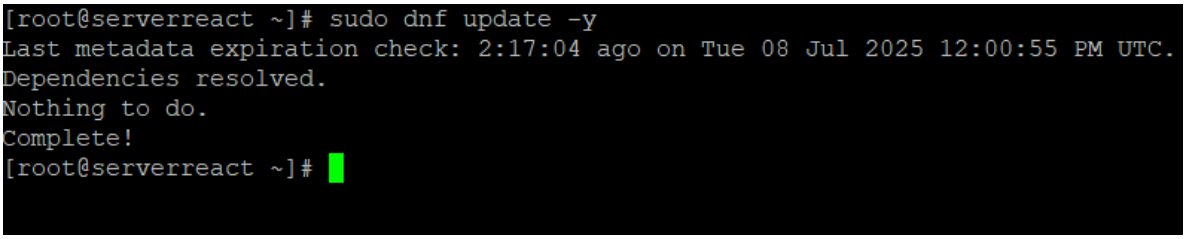


Figura 7. Actualización.

5. Instalar herramientas:
6. sudo dnf install -y epel-release
7. sudo dnf install -y wget curl git vim nano htop

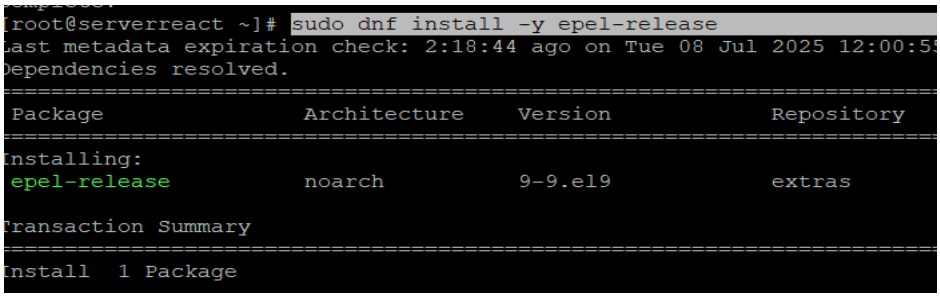


Figura 8. Instalación de herramientas.

8. Instalar paquetes del servidor: **sudo dnf config-manager --set-enabled powertools**
9. Instalar herramientas de desarrollo:
10. sudo dnf groupinstall -y "Development Tools"
11. Instalar python y crear entorno virtual:
12. sudo dnf install -y python3 python3-pip python3-devel
13. No es necesario crear venv.

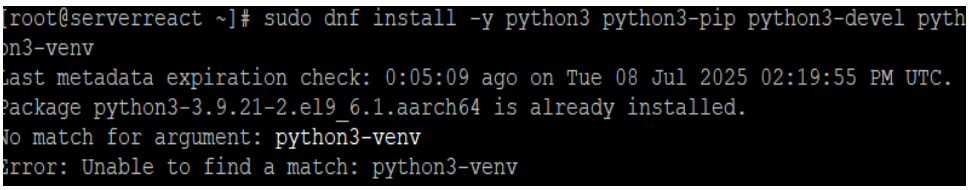


Figura 9. Herramienta de desarrollo.

14. Instalar nodejs:
15. sudo dnf install -y nodejs npm
16. Acceder a la carpeta:
17. cd /opt/miproyecto
18. Clonar el repositorio: https://github.com/GaryJ14/TesisIsma.git
19. Acceder a la ruta base del backend.

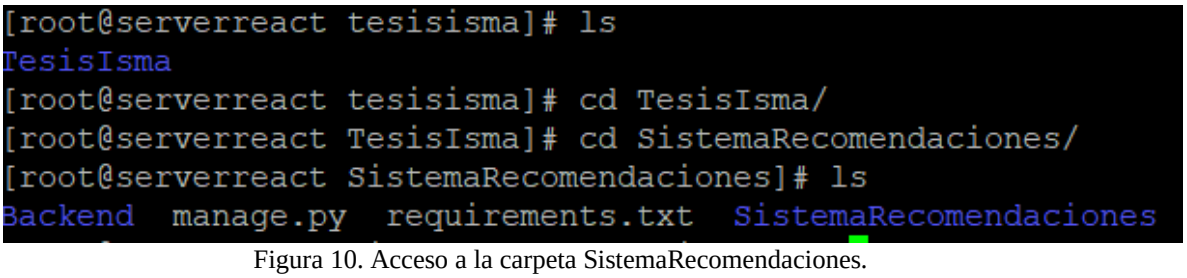


Figura 10. Acceso a la carpeta SistemaRecomendaciones.

20. Crear entorno virtual en la ruta base indicada en el paso anterior:
21. python3 -m venv env
22. Activar entorno virtual:

23. source env/bin/activate
24. Actualizar el pip:
25. pip install --upgrade pip
26. Instalar el requirements:
27. pip install -r requirements.txt
28. Desactivar el entorno virtual:
29. desactivate

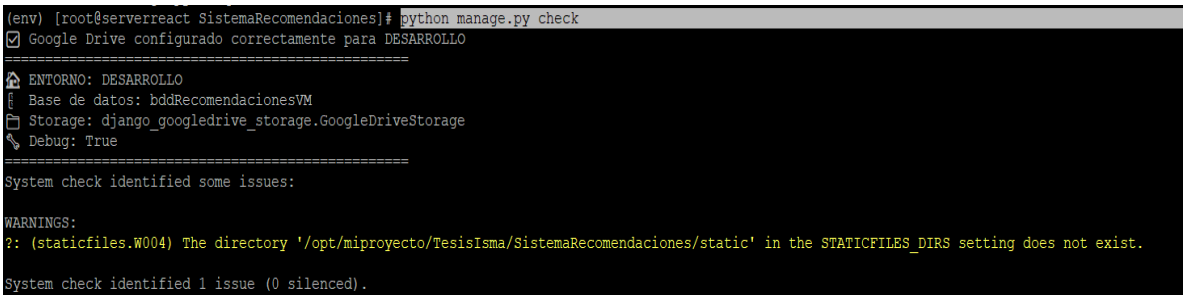


Figura 11. Entorno de Desarrollo.

30. Instalar postgres del repositorio:
31. sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-9-x86_64/pgdg-redhat-repo-latest.noarch.rpm
32. sudo dnf repolist | grep pgdg
33. sudo dnf clean all
34. sudo dnf install -y postgresql16-server postgresql16-contrib postgresql16-devel --nogpgcheck
35. Incializar postgres:
36. sudo /usr/pgsql-16/bin/postgresql-16-setup initdb
37. sudo ls -la /var/lib/pgsql/16/data/
38. Editar configuraciones de postegres y colocar **trust**:
39. sudo nano /var/lib/pgsql/16/data/pg_hba.conf
40. Editar las líneas de local y port:
41. sudo nano /var/lib/pgsql/16/data/postgresql.conf
42. Verificar que esta levantado el servidor:
43. sudo systemctl enable postgresql-16
44. sudo systemctl start postgresql-16
45. sudo systemctl status postgresql-16

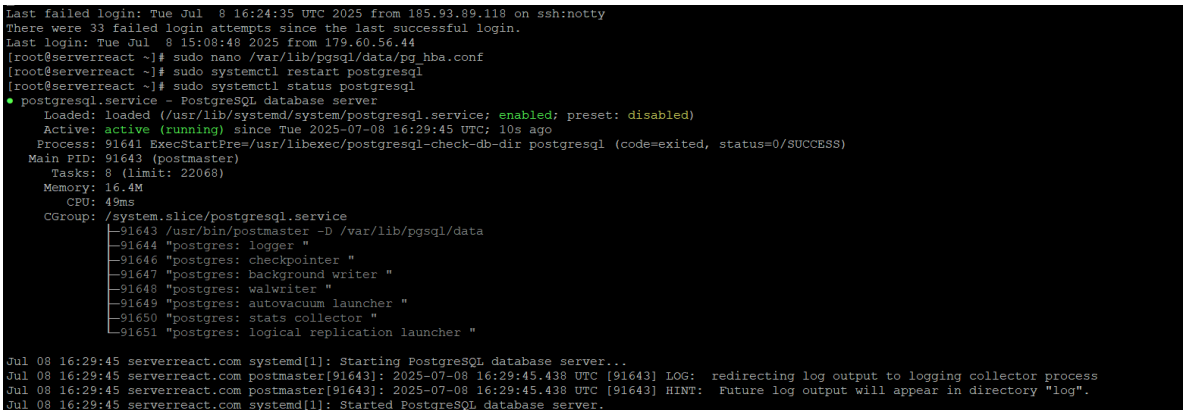


Figura 12. Verificación de levantamiento del servidor.

46. Agregar al path:
47. echo 'export PATH=/usr/pgsql-16/bin:$PATH' >> ~/.bashrc
48. source ~/.bashrc
49. Abrir postgres:
50. sudo -u postgres /usr/pgsql-16/bin/psql
51. Alterar el user de postgres:
52. ALTER USER postgres PASSWORD 'postgres123';
53. CREATE USER tesisuser WITH PASSWORD 'tesispass123';
54. CREATE DATABASE bddrecomendacionesvm_prod OWNER tesisuser;
55. GRANT ALL PRIVILEGES ON DATABASE bddrecomendacionesvm_prod TO tesisuser;
56. \l
57. \q
58. Ejecutar:
59. python manage.py makemigrations
60. python manage.py migrate

```
Operations to perform:
  Apply all migrations: ContenidoEti_infrastructure, Usuario_infrastructure, admin, auth, contenidoEliminado_infrastructure, conte
re, estUsuario_infrastructure, etiqueta_infrastructure, favorito_infrastructure, historialBusq_infrastructure, historialRep_infras
Running migrations:
  Applying etiqueta_infrastructure.0001_initial... OK
  Applying contenido_infrastructure.0001_initial... OK
  Applying ContenidoEti_infrastructure.0001_initial... OK
  Applying contenttypes.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0001_initial... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying Usuario_infrastructure.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
```

Figura 13. Migración de datos.

61. En el archivo **settings.py**:

62. Agregar las dos rutas después del comentario Frontend

```
else:
    # Para producción - ajusta según necesites
    CORS_ALLOWED_ORIGINS = [
        "http://localhost:3000",
        "http://127.0.0.1:3000",
        # Añade aquí las URLs de tu frontend cuando las tengas
        "http://94.130.77.86:3000",
        "http://94.130.77.86:8000"
    ]
    CORS_ALLOW_ALL_ORIGINS = True

CORS_ALLOW_CREDENTIALS = True
```

Figura 14. Rutas.

63. Agregar las siguientes líneas después de "CORS_ALLOW_CREDENTIALS"

```
CORS_ALLOW_CREDENTIALS = True

CORS_ALLOW_METHODS = [
    'DELETE',
    'GET',
    'OPTIONS',
    'PATCH',
    'POST',
    'PUT',
]
CORS_ALLOWED_HEADERS = [
    'accept',
    'accept-encoding',
    'authorization',
    'content-type',
    'dnt',
    'origin',
    'user-agent',
    'x-csrftoken',
    'x-requested-with',
]
```

Figura 15. CORS_ALLOW_CREDEMTIALES.

64. Agregar la ip pública:

```
Hosts permitidos
f ENVIRONMENT == 'development':
   ALLOWED_HOSTS = ['localhost', '127.0.0.1', '0.0.0.0']
lse:
   # Para producción en tu servidor AlmaLinux
   ALLOWED_HOSTS = [
       'localhost',
       '127.0.0.1',
       '0.0.0.0',
       '94.130.77.86'
       # 'TU.IP.PUBLICA.AQUI',
   ]
```

Figura 16. Ip pública.

DESPLEGAR EN SERVER

65. Instalar gunicorn
66. pip install gunicorn
67. gunicorn --bind 0.0.0.0:8000 SistemaRecomendaciones.wsgi
68. Instalar nginx
69. sudo dnf install -y nginx --nogpgcheck
70. sudo systemctl enable nginx
71. sudo systemctl start nginx
72. sudo systemctl status nginx
73. Verificar que el nginx este corriendo:
74. sudo systemctl status nginx
75. Configuraciones:
76. sudo nano /etc/nginx/conf.d/tesisisma.conf
77. Ejecutar:
78. sudo nginx –t
79. sudo systemctl restart nginx
80. sudo systemctl status nginx

CONFIGURACIÓN PARA REACT

81. Abrir config.js
82. nano src/config.js
83. Agregar la ip del servidor:

```
// src/config.js
const BACKEND_URL = process.env.NODE_ENV === 'production'
  ? 'http://94.130.77.86:8000'  // Reemplaza TU_IP_SERVIDOR por la IP real
  : 'http://localhost:8000';

export { BACKEND_URL };

// También puedes exportar como default si lo prefieres
export default BACKEND_URL;
```

Figura 17. Agregación de ip al servidor.

84. Cambiar **localhost:8000** por la ip del server **94.130.77.86:8000**, esto para todo el api del frontend que estén como localhost.

```
import axios from 'axios';

const API_URL = 'http://94.130.77.86:8000/api';

const api = axios.create({
  baseURL: API_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Registro
export const register = async (payload) => {
```

Figura 18. Cambio de ip.

85. Levantar el servidor:
86. npm run build
87. npm start
88. Detener servidor:
89. Ctrl + c
90. Cambiar el archivo **config.js** agregando la ip publica del servidor

```
export const BACKEND_URL = 'http://94.130.77.86';
```

Figura 19. Ip pública al servidor.

91. Construir el proyecto:
92. npm run build

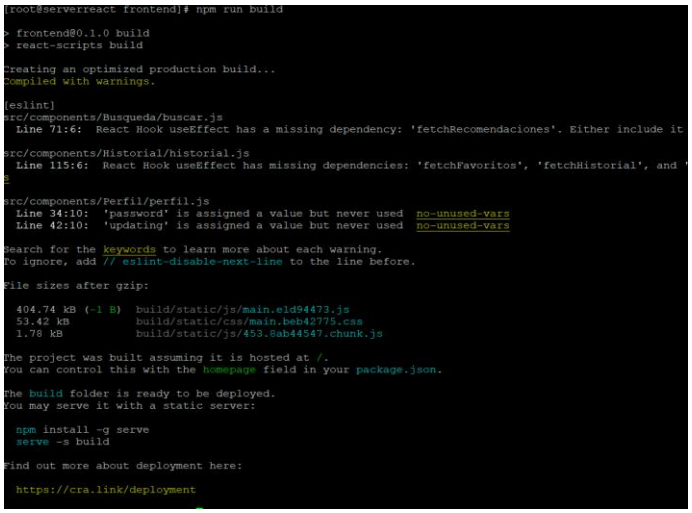


Figura 20. Construcción del proyecto.

93. Eliminar el puerto 8000 de las rutas:
94. cd /opt/miproyecto/TesisIsma/frontend
95. find src/ -name "*.js" -exec sed -i 's/http:\/\/94\.130\.77\.86:8000/http:\/\/94.130.77.86/g' {} \;
96. grep -r "94.130.77.86:8000" src/ || echo "✅ Todas las referencias :8000 fueron eliminadas"
97. Construir proyecto:
98. sudo systemctl restart nginx
99. npm run build
100. Definir tamaño para los archivos:
101. sudo nano /etc/nginx/conf.d/tesisisma.conf

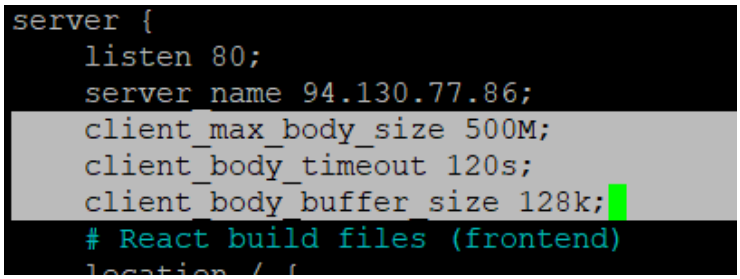


Figura 21. Definir tamaño para los archivos.

102. Restaurar el nginx:
103. sudo systemctl restart nginx
104. Restaurar django:
105. sudo systemctl restart tesisisma-django

6.3. COMPROBACIÓN DE LA HIPOTESIS

Una vez evaluadas las métricas del sistema, es posible verificar la validez de la hipótesis planteada. Por ello, se analizan los resultados obtenidos mediante el cálculo, comparación y revisión de los indicadores definidos. Esta comparación permite determinar si el sistema desarrollado cumple con los objetivos propuestos en cuanto a calidad, eficiencia y comportamiento, comprobando así su diseño basado en una arquitectura de microservicios.

Tabla 21 Comprobación de Hipótesis.

Métricas calculadas	Valor	Rangos	Análisis resultado
Numero de dependencias externas (ND)	0	(0 – 2) BAJO (3-5) MODERADO (6 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.
Numero de dependencias internas (NI)	3	(0 – 5) BAJO (6 - 10) MODERADO (11 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.
Profundidad de las dependencias (PD)	0	(0 – 2) BAJO (3 - 5) MODERADO (6 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.
Acoplamiento Estimado (AC_est)	4.75	(0 – 3) BAJO (4 - 6) MODERADO (7 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango MODERADO.
Falta de Cohesión (LCOH_est)	1.6	(0.5 – 1.0) BAJO (1.1 – 2.0) MODERADO (2.1 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango MODERADO.
Dependencia Estimadas (D_est)	4.6	(5 - 8) BAJO (9 - 12) MODERADO (13 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.

Métricas calculadas	Valor	Rangos	Análisis resultado
Complejidad Ciclomática Estimada (CC_est)	2	(3 - 5) BAJO (6 - 8) MODERADO (9 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.
Coeficiente de Acoplamiento Estimada (CC_est)	0.0	(0 – 1) BAJO (1.1 – 2.0) MODERADO (2.1 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.
Complejidad Estructuras Estimada (CE_est)	7	(10 -15) BAJO (16 - 20) MODERADO (21 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.
Puntos de Historias Estimada (CE_est)	17	Hace referencias a las funcionalidades que debe cumplir el sistema.	El valor proyectado en la aplicación MOAM muestra las funcionalidades necesarias que debe cumplir el sistema.
Complejidad Estimada (C_est)	26.0	(30 - 45) BAJO (46 - 60) MODERADO (61 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.
Índice Agregado Estimado (AI_est)	30.6	(40 -55) BAJO (56 - 70) MODERADO (71 o más) ALTO	El valor proyectado en la aplicación MOAM muestra que está dentro del rango BAJO.

Fuente: Los desarrolladores.

Luego de realizar la comprobación correspondiente, se verifica que las métricas calculadas por la herramienta MOAM se encuentran dentro de los rangos BAJOS y MODERADOS. Esto indica que el sistema mantiene una complejidad contralada y una dependencia aceptable entre microservicios, además la misma herramienta sugiere recomendaciones de mejora en la arquitectura como se muestra en la figura 22.

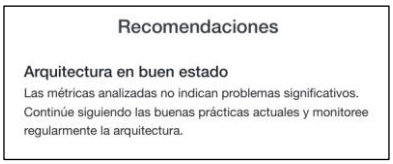


Figura 22 Recomendación MOAM [15].

7. CONCLUSIONES

- La investigación bibliográfica realizada proporciono una sólida fundamentación teórica, permitiendo comprender en profundidad el objeto y campo de estudio, lo cual fue fundamental para orientar adecuadamente el desarrollo del proyecto y asegurar la pertinencia de las decisiones metodológicas y técnicas.
- La implementación del sistema de recomendación y descubrimiento de contenido multimedia, basada en una arquitectura de microservicios y aplicada mediante practicas ágiles, demostró ser efectiva para lograr un desarrollo rápido, flexible y escalable, facilitando el mantenimiento y la integración continua del prototipo.
- La evaluación de las métricas de calidad de dependencia y complejidad utilizando la herramienta MOAM confirmo que el prototipo cumple con los parámetros aceptables, evidenciando que la arquitectura deseñada es eficiente y sostenible, lo que valida la correcta aplicación del modelo de optimización para microservicios en el proyecto.

8. RECOMENDACIONES

- Diseñar e implementar sistemas basados en microservicios, ya que este enfoque permite mantener las métricas de calidad, dependencia y complejidad en niveles óptimos, facilitando además la extensión y mejora continua de la arquitectura en futuros proyectos relacionado con sistemas distribuidos.
- Estructurar las aplicaciones en capas bien definidas. Lo que favorece la claridad del código, facilita la realización de pruebas y mejora la escalabilidad.

Proporcionando una base sólida para el mantenimiento y la evolución de sistemas que utilicen arquitecturas similares.

- Utilizar mecanismos de evaluación continua como la herramienta MOAM, la cual ha demostrado ser eficaz para medir métricas clave en microservicios. Para complementar o ampliar el análisis de mejor manera también se puede considerar otras herramientas como SonarQube, que evalúa calidad de código y detecta problemas de mantenimiento, o Prometheus con métricas personalizadas para monitoreo en tiempo real. Estas herramientas contribuyen en un desarrollo más controlado, eficiente y con mayor capacidad de mejora continua.

9. REFERENCIAS

[1] J. M. O. Candel, Tecnologías para arquitecturas basadas en microservicios: Patrones y soluciones para aplicaciones desplegadas en contenedores, 2020.

[2] J. H. H. Xia, «DESARROLLO DE UN SISTEMA DE RECOMENDACIÓN PARA UNA EMPRESA DE SERVICIOS ONLINE,» 13 de junio de 2023.

[3] L. Á. L. C. Reiman Alfonso Azcuy, «Módulo de recomendación de patrones de diseño para EGPat,» *Scielo,* 01 Junio 2021.

[4] A. Maitaigahassea, J. L. K. E. Fendji y M. Atemkeng, «Offline Content-based Recommendation System for Wikimedia Commons Contents,» *ScienceDirect,* 2025.

[5] E. M. Daniel López, «Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web,» 17 de Julio 2017.

[6] G. R. Alejandro, G. S. Agustín, M. Pualuk, N. Roy y R. Zalazar, «¿Son los microservicios la mejor opción? Una evaluación de su eficacia y eficiencia frente a los monolitos,» 2024.

[7] Á. I. Rodríguez, J. I. Padilla y H. A. Parra., «Arquitectura basada en micro-servicios para aplicaciones web,» 2019.

[8] V. J. M. Sekulic, «Traspasando fronteras: la sinergia de la,» La Laguna, 2023.

[9] Microsoft, «¿Qué es la arquitectura de software?,» *Microsoft,* 2021.

[10] D. M. Rodriguez, «Salesforce API Framework para Patrones de Diseño de Microservicios,» 2023.

[11] X. L. K. C.-V. José A. Valdivia, «Quality attributes in patterns related to microservice architecture: a Systematic Literature Review,» 2019.

[12] W. J. C. Jhon Castaño-Henríquez, «Métricas en la evaluación de la calidad del software: Una revisión conceptual,» Diciembre 2021.

[13] A. U. A.-M. O. Michel-Daniel Cojocaru, «Attributes Assessing the Quality of Microservices Automatically,» 2019.

[14] V. C. Tapia y C. M. Gaona, «Research Opportunities in Microservices Quality Assessment: A Systematic Literature Review,» 2023.

[15] V. d. C. T. Cerda, «Modelo inteligente para especificar dependencia y complejidad en el diseño de aplicaciones basadas en microservicios.,» Santiago de Cali, 27 de febrero 2025 [not published].

[16] R. Noriega, El proceso de Desarrollo de Sotware, 2015.

[17] B. B. F. L. C. M. B. Angel Miguel Hernández Oliva, «Modelos de desarrollo de software. ¿Cuál elegir ?,» *Gestión de Proyectos Informáticos,* 30 03 2018.

[18] E. S.-F. D. Porras-Alfaro, «El modelo iterativo e incremental para el desarrollo de la aplicación de realidad aumentada Amón_RA,» Tecnología en marcha, 2020.

[19] L. Gonzales, «Microservicios,» 2018. [En línea]. Available: https://aws.amazon.com/es/microservices/.

[20] J. Llumitasig, «Politicas de microservicios,» 2022. [En línea]. Available: https://azure.microsoft.com/es-es/solutions/modern-application-development.

[21] J. D. F. M. J. M. V. Andrés Navarro Cadavid, «Revisión de metodologías ágiles para el desarrollo de software,» 2013.

[22] M. d. l. M. G. Valeria Iliana Bertossi, «Prácticas ágiles en el desarrollo de objetos de aprendizaje: estado del arte,» *Revista Iberoamericana de Tecnología en Educación y Educación en Tecnología,* Marzo 2022.

[23] A. Millizta, «Enfasis de la colaboración.,» 2023. [En línea]. Available: https://fastercapital.com/es/palabra-clave/%C3%A9nfasis-colaboraci%C3%B3n.html.

[24] G. Álvarez, «Enfasis de calidad.,» 2016. [En línea]. Available: https://es.scribd.com/presentation/312387572/Enfasis-en-La-Calidad..

[25] L. Garcia, «Enfasis de velocidad en sistemas.,» 2024. [En línea]. Available: https://es.khanacademy.org/science/physics/two-dimensional-motion/two-dimensional-projectile-mot/a/what-are-velocity-components.

[26] J. Posligua, «Herraminetas de software,» 2021. [En línea].

[27] ISIL, «Herraminetas de desarrollo,» 2024. [En línea].

[28] F. Lima, «Microservicios.,» 2019. [En línea].

[29] A. Zurita, «Desarrollo de aplicaciones basadas en microservicios.,» 2017. [En línea].

[30] U. T. d. Pereira, «INTRODUCCIÓN A LA CALIDAD DE SOFTWARE,» 2008.

[31] F. H. Vera–Rivera, «Puntos de complejidad cognitiva: una métrica para evaluar el diseño de aplicaciones basadas en microservicios,» 2023.

[32] R. S. Pressman, Ingeniería del software un enfoque práctico séptima edición, 2010.

[33] F. H. V. R. MSc, MODELO INTELIGENTE DE ESPECIFICACIÓN DE LA GRANULARIDAD DE APLICACIONES BASADAS EN MICROSERVICIOS., 2021.

[34] P. J. Rendón, «MICROSERVICIOS,» 2023.

[35] E. Maida, Desarrollo de Software, Cátedra Seminario de Sistemas , 2015.

[36] J. Pacienzia, Tipode de Software, 2019.

[37] O. Zimmermann, «Principios de los microservicios,» 2017. [En línea]. Available: https://link.springer.com/article/10.1007/s00450-016-0337-0.

[38] Newman, «Desafios de investigación en el desarrollo de aplicaciones basadas en microservicios.,» 2018. [En línea].

[39] M. d. l. M. G. Valeria Iliana Bertossi, «Prácticas ágiles en el desarrollo de objetos de aprendizaje: estado del arte,» *Revista Iberoamericana de Tecnología en Educación y Educación en Tecnología,* Marzo 2022.

[40] G. C. C. M. A. A. E. Tapia Cerda Verónica, «Evaluando Dependencia y Complejidad en el Diseño de Microservicios: Un Enfoque Integral,» *Revista Cientifica ProQuest,* 2024.

ANEXOS

ANEXO A: DEFINICIÓN DE ROLES DEL EQUIPO

Se menciona a los diferentes involucrados en este proyecto de investigación los cuales se muestran en la Tabla 21.

Tabla 22. Roles del Equipo

Rol	Responsable
Programador	Gary Jami Elizabeth Quinatoa
Cliente	Dra. Verónica del Consuelo Tapia Cerda, PhD
Tester	Gary Jami Elizabeth Quinatoa
Coach	Dra. Verónica del Consuelo Tapia Cerda, PhD
Tracter	Dra. Verónica del Consuelo Tapia Cerda, PhD

Fuente: Los desarrolladores.

ANEXO B: MODELO DE LA BASE DE DATOS

Para el diseño de la base de datos se utilizó Dbeaver ya que es una herramienta útil para la visualización de la estructura de la base de datos ya que genera un diagrama entidad relación de manera fácil y automática. Como se muestra en la Figura 22.

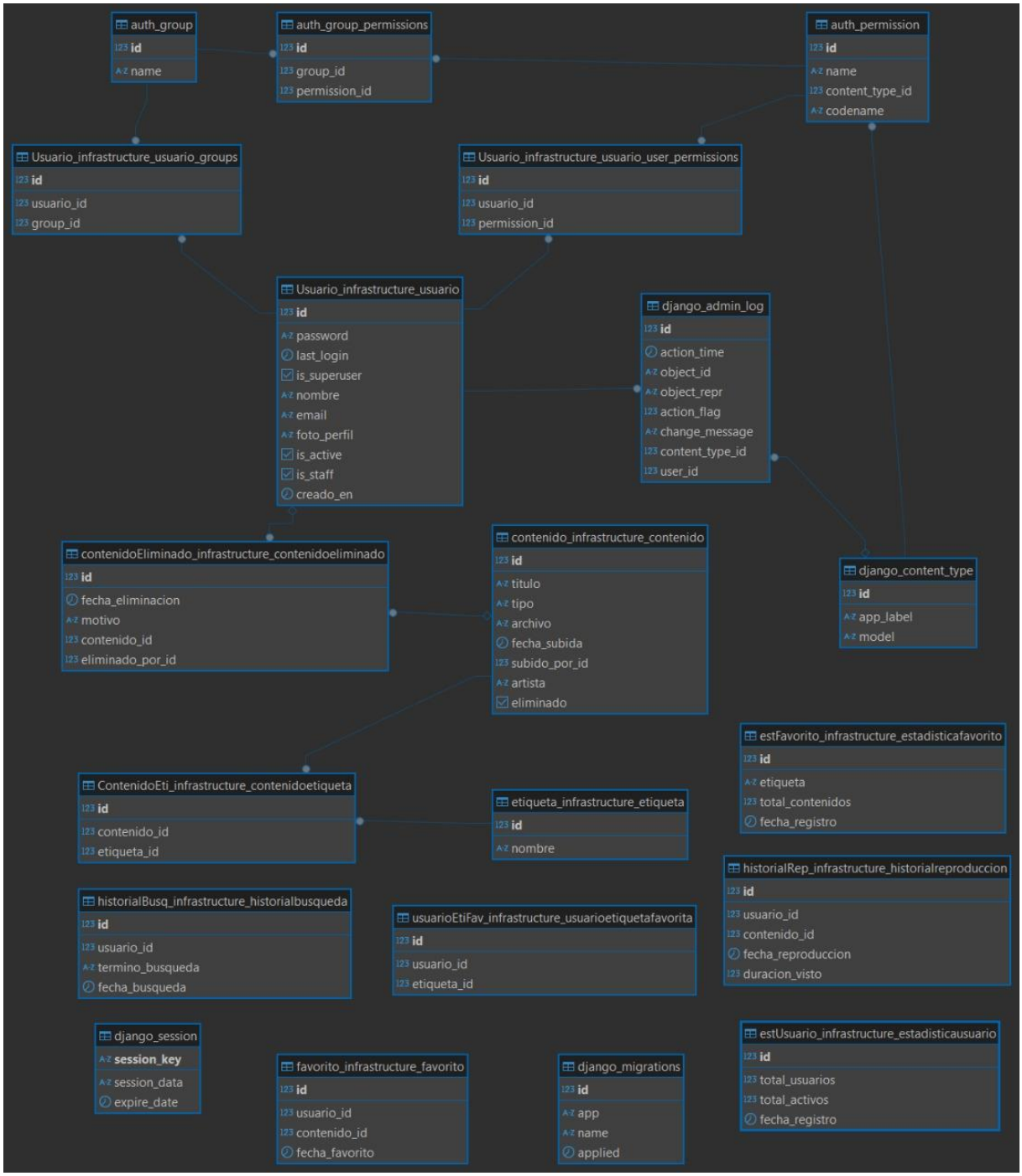


Figura 23. Modelo Bade de Datos.

ANEXO C: CAPTURAS DEL PROTOTIPO

A continuación, se muestra las interfaces desarrolladas.

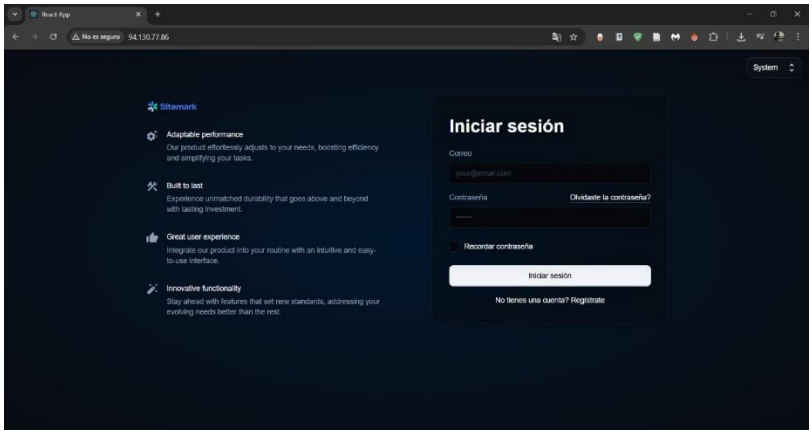


Figura 24. Interfaz de inicio de sesión.

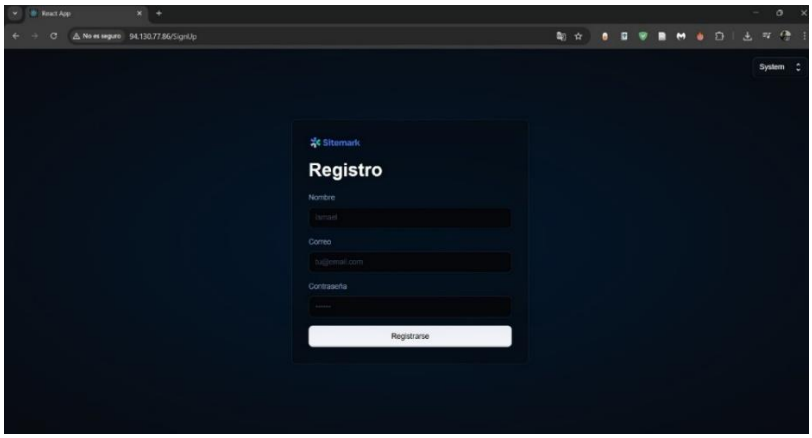


Figura 25. Interfaz Registro.

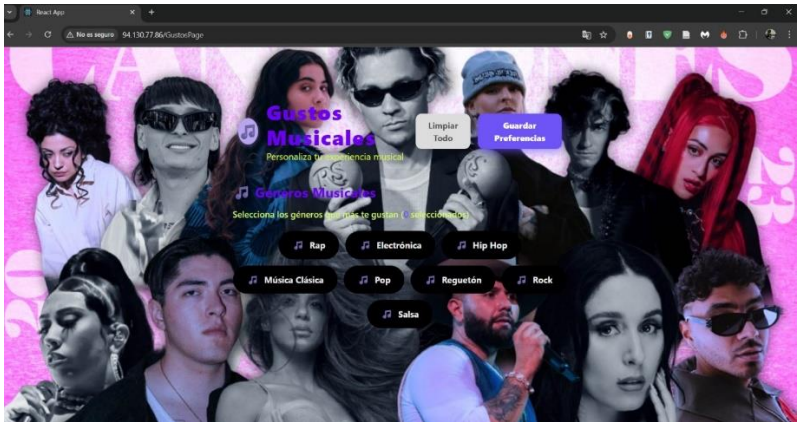


Figura 26. Interfaz gustos musicales.

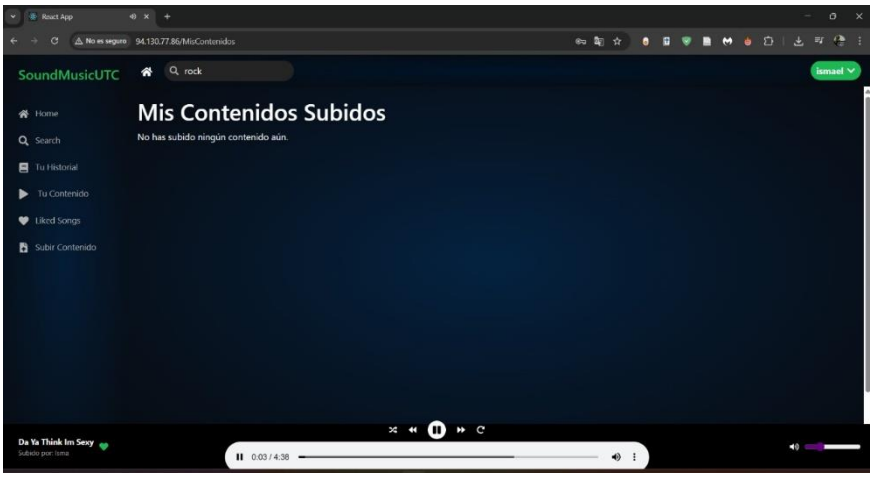


Figura 27. Interfaz de contenidos subidos por los usuarios.

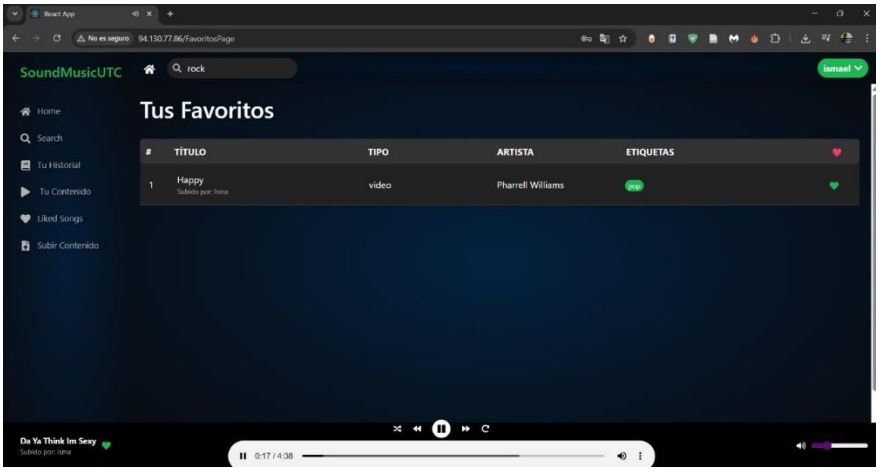


Figura 28. Interfaz tus Favoritos.

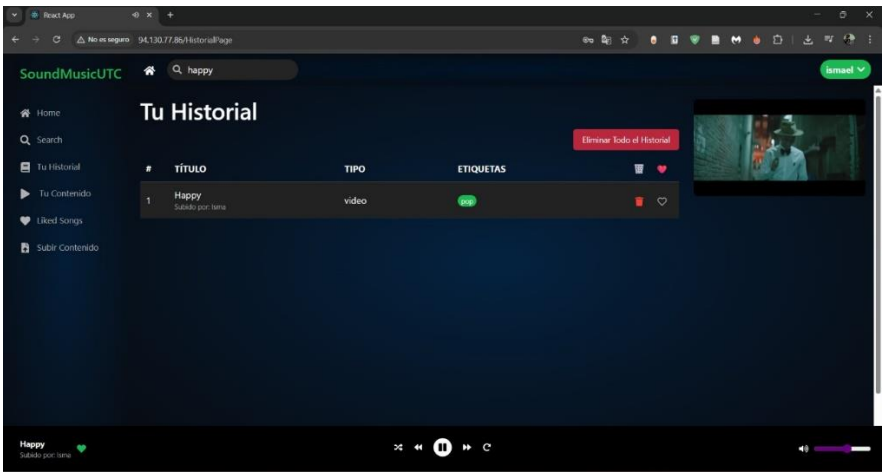


Figura 29. Interfaz Historial.

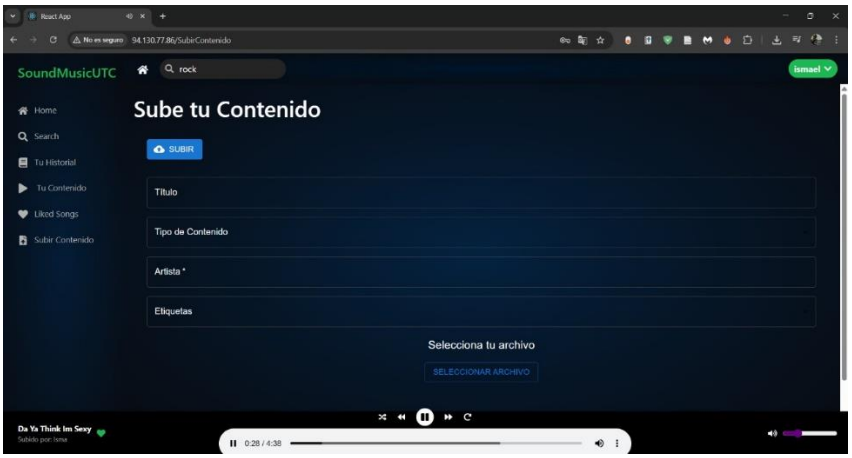


Figura 30. Interfaz subir contenido.

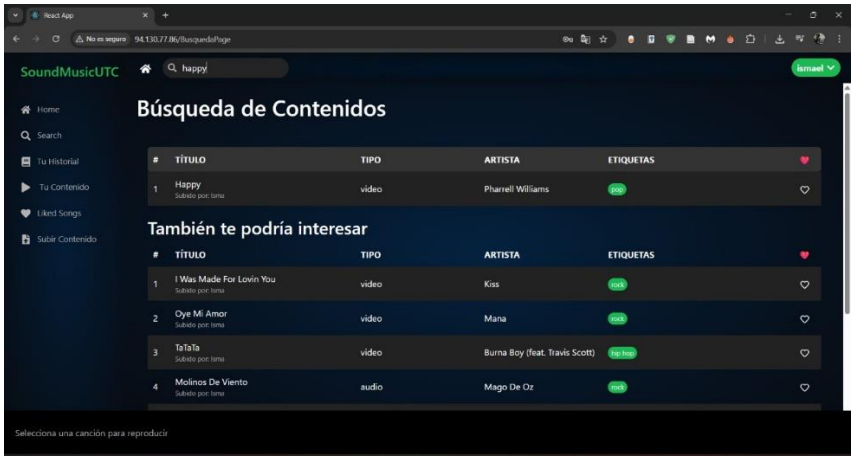


Figura 31. Búsqueda de contenido.

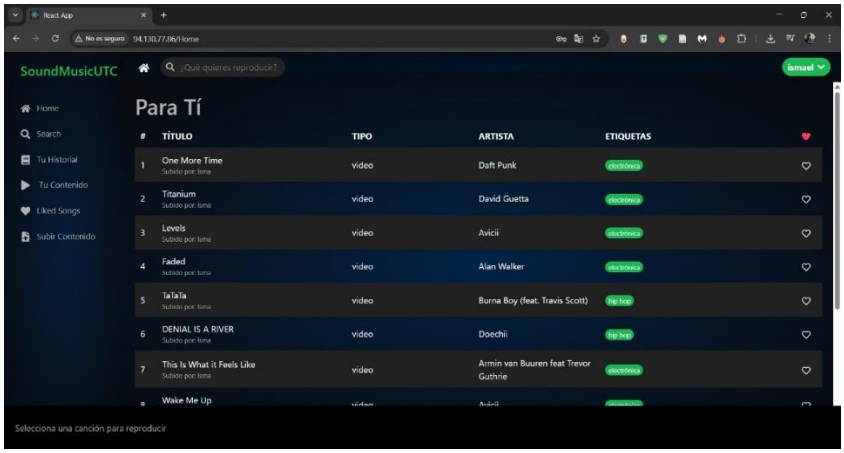


Figura 32. Interfaz para ti.

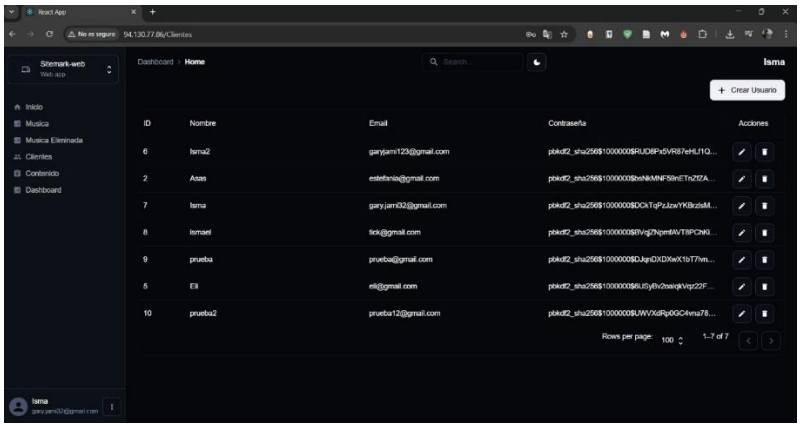


Figura 33. Interfaz listado de clientes.

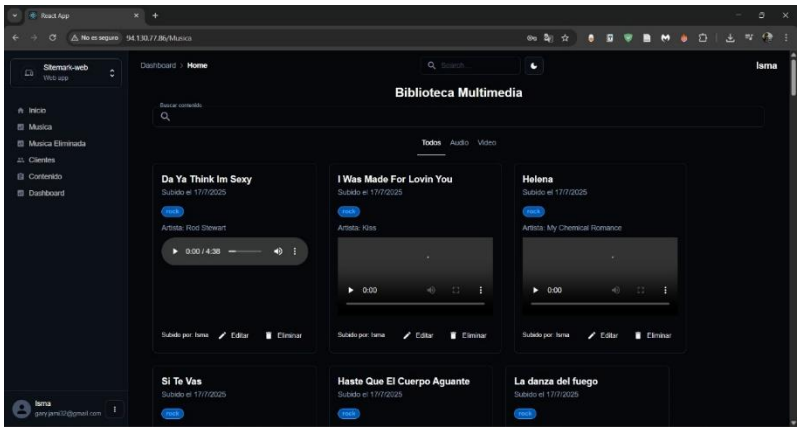


Figura 34. Interfaz Biblioteca Multimedia.

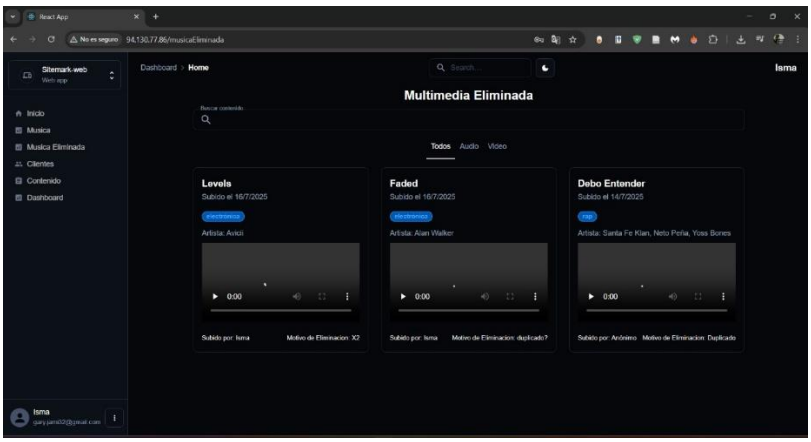


Figura 35. Interfaz Multimedia Eliminada.

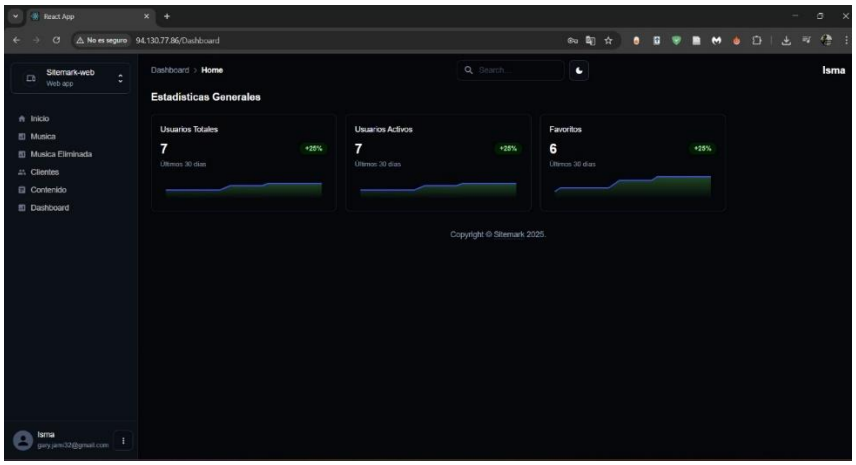


Figura 36. Interfaces estadísticas.

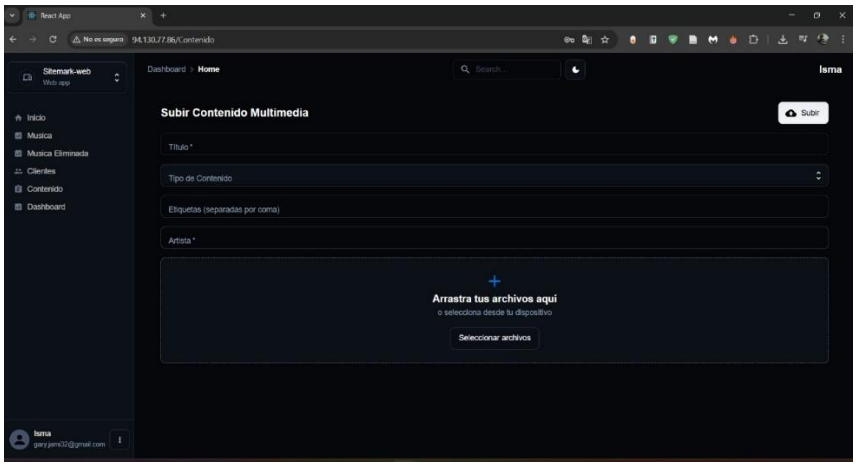


Figura 37. Interfaz subir contenido (Administrador).