



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Software

**Arquitectura para mitigar la complejidad inicial del
mantenimiento de software basada en Clean
Architecture**

TESIS

Para optar el Título Profesional de Ingeniero de Software

AUTORES

Anthony Elias RICSE PEREZ

Carlos Daniel TARMEÑO NORIEGA

ASESOR

Dra. Lenis Rossi WONG PORTILLO

Lima, Perú

2026



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Ricse, A. & Tarmeño, C. (2026). *Arquitectura para mitigar la complejidad inicial del mantenimiento de software basada en Clean Architecture*. [Tesis de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Software]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios

Datos de autor 1	
Nombres y apellidos	Anthony Elias Ricse Perez
Tipo de documento de identidad	DNI
Número de documento de identidad	72694445
URL de ORCID	https://orcid.org/0009-0005-9678-5053
Datos de autor 2	
Nombres y apellidos	Carlos Daniel Tarmeño Noriega
Tipo de documento de identidad	DNI
Número de documento de identidad	70583369
URL de ORCID	https://orcid.org/0009-0005-9768-3163
Datos de asesor	
Nombres y apellidos	Lenis Rossi Wong Portillo
Tipo de documento de identidad	DNI
Número de documento de identidad	10438282
URL de ORCID	https://orcid.org/0000-0002-5032-3233
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Hugo David Calderón Vilca
Tipo de documento	DNI
Número de documento de identidad	01317748
Miembro del jurado 1	
Nombres y apellidos	Rosa Menéndez Mueras
Tipo de documento	DNI
Número de documento de identidad	10246770
Datos de investigación	
Línea de investigación	C.0.3.22. Ingeniería de software

Grupo de investigación	No aplica
Agencia de financiamiento	Sin financiamiento
Ubicación geográfica de la investigación	Universidad Nacional Mayor de San Marcos Edificio: Facultad de Ingeniería de Sistemas e Informática País: Perú Departamento: Lima Provincia: Lima Distrito: Lima Dirección: Av. Carlos Germán Amézaga 375 Latitud: -12.053697 Longitud: -77.085774
Año o rango de años en que se realizó la investigación	Septiembre 2022 – Junio 2026
URL de disciplinas OCDE	Ingeniería de sistemas y comunicaciones https://purl.org/pe-repo/ocde/ford#2.02.04



UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERIA DE SISTEMAS E INFORMATICA
Escuela Profesional de Ingeniería de Software

Acta de Sustentación de Tesis

Siendo las once y treinta (11:30) horas del día seis (06) del mes mayo del año 2026, se reunieron en la sala presidido por el Dr. Hugo David Calderón Vilca (Presidente), Mg. Rosa Menéndez Mueras (Miembro) y la Dra. Lensa Rossi Wong Portillo (Asesora), para la sustentación de la tesis Intitulada: **"ARQUITECTURA PARA MITIGAR LA COMPLEJIDAD INICIAL DEL MANTENIMIENTO DE SOFTWARE BASADA EN CLEAN ARCHITECTURE"**, por el Bachiller **Anthony Elias, RICSE PEREZ**, para obtener el Título Profesional de Ingeniero de Software.

Acto seguido de la exposición de la Tesis, el Presidente invitó al Bachiller a responder las preguntas formuladas por los Miembros del Jurado.

El Bachiller, en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez las preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, el bachiller obtuvo la nota de (18:00). Dieciocho

A continuación, el Presidente del Jurado Dr. Hugo David calderón Vilca, declara al Bachiller Ingeniero de Software.

Siendo horas, se levantó la sesión.

Dr. Hugo David Calderón Vilca
Presidente

Mg. Rosa Menéndez Mueras
Miembro

Dra. Lenis Rossi Wong Portillo
Asesora



UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERIA DE SISTEMAS E INFORMATICA
Escuela Profesional de Ingeniería de Software

Acta de Sustentación de Tesis

Siendo las once y treinta (11:30) horas del día seis (06) del mes mayo del año 2026, se reunieron en la sala presidido por el Dr. Hugo David Calderón Vilca (Presidente), Mg. Rosa Menéndez Mueras (Miembro) y la Dra. Lensa Rossi Wong Portillo (Asesora), para la sustentación de la tesis Intitulada: **"ARQUITECTURA PARA MITIGAR LA COMPLEJIDAD INICIAL DEL MANTENIMIENTO DE SOFTWARE BASADA EN CLEAN ARCHITECTURE"**, por el Bachiller **Carlos Daniel, TARMEÑO NORIEGA**, para obtener el Título Profesional de Ingeniero de Software.

Acto seguido de la exposición de la Tesis, el Presidente invitó al Bachiller a responder las preguntas formuladas por los Miembros del Jurado.

El Bachiller, en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez las preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, el bachiller obtuvo la nota de (18.0). *Dieciocho*

A continuación, el Presidente del Jurado Dr. Hugo David calderón Vilca, declara al Bachiller Ingeniero de Software.

Siendo horas, se levantó la sesión.

Dr. Hugo David Calderón Vilca
Presidente

Mg. Rosa Menéndez Mueras
Miembro

Dra. Lenis Rossi Wong Portillo
Asesora



Anexo 1

CERTIFICADO DE SIMILITUD

Yo, LENIS ROSSI WONG PORTILLO, en mi condición de asesor acreditado con el Dictamen de la EP N° 000001-2025-EPISW-FISI/UNMSM, de la tesis de investigación, cuyo título es:

“ARQUITECTURA PARA MITIGAR LA COMPLEJIDAD INICIAL DEL MANTENIMIENTO DE SOFTWARE BASADA EN CLEAN ARCHITECTURE”,

Presentado por los bachilleres: Anthony Elias Ricse Perez y Carlos Daniel Tarmeño Noriega, para optar el Título Profesional de Ingeniero de Software, CERTIFICO que se ha cumplido con lo establecido en la Directiva de Originalidad y de Similitud de Trabajos Académicos, de Investigación y Producción Intelectual. Según la revisión, análisis y evaluación mediante la plataforma PIDS, el documento evaluado cuenta con el 06 % de similitud, nivel PERMITIDO para continuar con los trámites correspondientes y para su **publicación en el repositorio institucional**.

Se emite el presente certificado en cumplimiento de lo establecido en las normas vigentes, como uno de los requisitos para la obtención del Título Profesional de Ingeniero de Software.

Lima, 04 de agosto del 2026

Firma del Asesor:

Nombres y Apellidos del asesor: Dra. LENIS ROSSI WONG PORTILLO

DNI: 10438282

* Según corresponda: bachiller, magíster, doctor; título profesional o título de segunda especialidad

La vigencia del presente certificado corresponde a un período de noventa (90) días desde la fecha de su emisión; siempre que no haya observaciones en el texto durante el proceso de revisión por el jurado informante o surja alguna contingencia.

DEDICATORIA

Este trabajo está dedicado a mi familia,
compañeros y en especial a mis padres.

A mi familia, por ser siempre mi
mayor motivación y fortaleza.

AGRADECIMIENTOS

Expresamos nuestro agradecimiento a la profesora Lenis Wong, por su dedicación, que fueron fundamentales para alcanzar los objetivos trazados en este trabajo.

A nuestros padres, por su sacrificio, constante y preocupación que siempre nos brindaron para perseverar en nuestra formación profesional.

A todas aquellas personas que, de manera indirecta, contribuyeron con su apoyo para que podamos culminar este esfuerzo académico.

Anthony Elías Ricse Pérez
Carlos Daniel Tarmeño Noriega

**Arquitectura para mitigar la complejidad inicial del mantenimiento del Software
basada en Clean Architecture**

Orientador: Lenis Wong Portillo

UNMSM - LIMA
JUNIO 2026

Arquitectura para mitigar la complejidad inicial del mantenimiento de Software basada en Clean Architecture

RESUMEN

El aumento de la mantenibilidad al iniciar un sistema de Software basado en Clean Architecture se fundamentó en criterios que garantizan la sostenibilidad del sistema a lo largo del tiempo. Sin embargo, los sistemas iniciales suelen descuidar estos principios, generó como resultado baja mantenibilidad y una difícil analizabilidad del sistema en muchos casos, lo que impedía adquirir nuevos requisitos durante su ciclo de vida, estas características se manifestaron bajo problemas que se observan al inicio de un sistema basado en Clean Architecture. Según la literatura revisada, varios autores coincidieron que este tipo de problema pudo ser la causa de muchas arquitecturas en temas de esfuerzo y costo en aplicaciones ya en producción. Acorde a lo anterior, se propuso en esta investigación un sistema de 3 capas basado en Clean Architecture con el fin de disminuir la complejidad inicial de mantenimiento en un sistema tradicional de 4 capas basada en Clean Architecture. La metodología de la investigación se desarrolló en cuatro fases: (i) Evaluación arquitectónica, en la cual se analizaron las características y limitaciones de la arquitectura tradicional de 4 capas; (ii) Diseño de la arquitectura de 3 capas, donde se definió la nueva estructura propuesta y su comunicación interna; (iii) Implementación del sistema, que consistió en desarrollar un proyecto web utilizando un conjunto de datos de prueba diseñado para validar esta variante de desempeño de la arquitectura experimental de 3 capas frente al grupo de control de 4 capas; y (iv) Validación de la propuesta, donde se realizó un análisis de código estático. Para obtener los resultados, se aplicó un enfoque cuantitativo de pruebas por escenarios, es decir grupos de estudios y una comparación del análisis estático resultante. Se evaluaron cuatro estados evolutivos del código combinando dos módulos de abstracción clave, lo que permitió medir de forma incremental la complejidad estructural. A partir de un dataset controlado y usando herramientas como SonarQube y Dependency-Cruiser, se compararon empíricamente las arquitecturas de 3 y 4 capas en términos de dependencias, mantenibilidad y estructura. Los resultados determinaron que hubo una mejora general en la mantenibilidad del proyecto basado en la arquitectura de 3 capas. Según los criterios definidos por las normas ISO/IEC 25010 e ISO/IEC 25022, la modularidad aumentó en +3.85%, la reusabilidad en +1.97%, la analizabilidad en +28.08%, la modificabilidad en +19.78%, y la testabilidad en +71.71%. En conjunto, el proyecto mostró una mejora global

del +16.71% con respecto al sistema de 4 capas, confirmando que la reducción de complejidad arquitectónica al inicio del proyecto tiene un impacto positivo en la calidad y mantenibilidad del software.

Palabras Claves: Clean Architecture, arquitectura de software, mantenibilidad, métricas.

Architecture to mitigate the initial complexity of software maintenance based on Clean Architecture

ABSTRACT

The increase in maintainability when initiating a software system based on Clean Architecture was founded on criteria that ensured the system's long-term sustainability. However, initial systems often overlook these principles, resulting in low stability and difficult system analyzability, which in many cases prevented new requirements from being incorporated throughout the system's lifecycle. These characteristics manifested as problems commonly observed at the beginning of a system based on Clean Architecture. According to the reviewed literature, several authors agreed that this type of issue could be the cause of increased effort and cost in architectures of applications already in production. In line with this, the present research proposed a 3-layer system based on Clean Architecture with the aim of reducing the initial maintenance complexity found in a traditional 4-layer Clean Architecture system.

The research methodology was developed in four phases: (i) Architectural evaluation, in which the characteristics and limitations of the traditional 4-layer architecture were analyzed; (ii) Design of the 3-layer architecture, where the proposed structure and its internal communication were defined; (iii) System implementation, consisting of developing a web project using a test dataset designed to validate the performance of the experimental 3-layer architecture against the 4-layer control group; and (iv) Validation of the proposal, where a static code analysis was carried out.

To obtain the results, a scenario-based testing approach and a comparative static analysis were applied. Four evolutionary states of the codebase were evaluated by combining two key abstraction modules, allowing incremental measurement of structural complexity. Using a controlled dataset and tools such as SonarQube and Dependency-Cruiser, the 3-layer and 4-layer architectures were empirically compared in terms of dependencies, maintainability, and structural composition. The results showed a general improvement in the maintainability of the project based on the 3-layer architecture. According to the criteria defined by ISO/IEC

25010 and ISO/IEC 25022, modularity increased by +3.85%, reusability by +1.97%, analyzability by +28.08%, modifiability by +19.78%, and testability by +71.71%. Overall, the project demonstrated a global improvement of +16.71% compared to the 4-layer system, confirming that reducing architectural complexity at the beginning of the project has a positive impact on software quality and maintainability.

Keywords: Clean Architecture , software architecture, maintainability, metrics.

ÍNDICE

Capítulo I: Introducción.....	20
1.1. Formulación del problema.....	20
1.1.1. Problema general.....	20
1.1.2. Problema específicos.....	20
1.2. Antecedentes del problema.....	20
1.3. Justificación.....	22
1.4. Objetivos.....	24
1.4.1. Objetivo General.....	24
1.4.2. Objetivos Específicos.....	24
1.5. Organización de la tesis.....	24
Capítulo II: Marco Teórico.....	26
2.1. Arquitectura Software.....	26
2.2. Tipos de Arquitecturas.....	26
2.2.1 Arquitectura Limpia.....	26
2.2.1.1 Capa de entidad.....	26
2.2.1.2 Capa de casos de Uso.....	27
2.2.1.3 Capa Adaptador.....	27
2.2.1.4 Capa de Presentación.....	27
2.2.2 Arquitectura Software Monolítica.....	27
2.2.3 Arquitectura basado en Microservicios.....	27
2.3. Principios Arquitectónicos.....	28
2.3.1. Single Responsibility.....	28
2.3.2. Open-Closed Principle.....	28
2.3.3. Interface Segregation Principle.....	28
2.3.4. Dependency Inversion Principle.....	28
2.4. Mantenibilidad.....	28
2.5. Tecnologías.....	29
2.5.1 React.js.....	29
2.5.2 Node.js.....	29
2.5.3 Base de Datos NoSQL.....	29
2.6. Aplicación Web.....	29
2.7. Patrones de diseño.....	30
2.7.1 Adapter.....	30
2.7.2 Strategy.....	30
2.7.3 Bridge.....	30
2.7.4 Composite.....	30
2.7.5 Builder.....	31
2.8. Normas.....	31
2.8.1 ISO/IEC 25010.....	31
2.8.2 ISO/IEC 25022.....	31
Capítulo III: Estado del Arte.....	32
3.1. Metodología de revisión de la literatura.....	32

3.2. Desarrollo de la revisión.....	33
3.3. Resultados de revisión.....	34
3.4. Revisión literaria.....	35
3.4.1. Arquitecturas para implementar un sistema Web.....	35
3.4.2. Framework para desarrollar un sistema web.....	46
3.4.3. Principios y patrones arquitectónicos.....	55
3.4.4. Métricas arquitectónicas.....	63
Capítulo IV: Metodología e Implementación.....	73
4.1. Evaluación arquitectónica.....	73
4.2. Diseño de la Arquitectura de 3 capas.....	74
4.2.1. Capas Principales.....	75
4.2.1.1. Capas Interfaz.....	76
4.2.1.1.1 Puerto de Accesos.....	77
4.2.1.2. Capas de uso.....	80
4.2.1.3. Capas de Entidad.....	81
4.2.2. Comunicación entre capas.....	82
4.2.3. Selección de la tecnología.....	83
4.3. Implementación del sistema.....	84
4.3.1. Diseño del Sistema.....	85
4.3.1.1. Database TMDb.....	85
4.3.1.2. Preprocesamiento de los datos.....	88
4.3.1.2.1. Diseño de entrada.....	88
4.3.1.2.2. Transformación de los datos de entrada.....	89
4.3.1.3. Diseño del sistema basado en 3 capas.....	90
4.3.1.3.1. Diseño de la capa interfaz.....	91
4.3.1.3.2. Diseño de la capa de uso o aplicación.....	92
4.3.1.3.3. Diseño de la capa entidad.....	93
4.3.1.4. Vista del Sistema.....	94
4.3.1.4.1. Vista modular.....	94
4.3.1.4.2. Vista de casos de uso.....	95
4.3.1.4.3. Vista de componentes.....	95
4.3.1.5. Requerimientos.....	96
4.3.1.5.1. Alcance.....	96
4.3.1.5.2. Actores del sistema.....	96
4.3.1.5.3. Historia De usuario.....	96
4.3.1.5.4. Product Backlog.....	110
4.3.2. Implementación de la arquitectura.....	110
4.3.2.1. Implementación de la capa interfaz.....	112
4.3.2.2. Implementación de la capa de uso o aplicación.....	118
4.3.2.3. Implementación de la capa entidad.....	120
4.3.2.4. Funcionalidades del sistema.....	122
4.3.2.4.1. Sprint 1.....	122
4.3.2.4.2. Sprint 2.....	128

4.2.3.5. Evaluación de funcionalidades del sistema.....	131
4.4. Validación de la propuesta.....	132
4.4.1. Métricas de mantenibilidad.....	133
4.4.1.1. Métrica de la norma ISO/IEC 25010.....	133
4.4.1.2. Métricas de la norma la ISO/IEC 25022.....	134
4.4.2. Planteamiento de los de casos de estudio.....	138
4.4.2.1. Dataset.....	138
4.4.2.1. Estructura de los registros.....	139
4.4.2.2. Obtención y filtrado de datos.....	139
4.4.2.3. Transformación de los datos de entrada.....	140
4.4.2.4. Casos de estudios.....	141
4.4.2.4.1. Grupo experimental.....	142
4.4.2.4.2. Grupo de control.....	146
4.4.2.5. Limitación.....	150
4.4.3. Implementación de las Métricas.....	151
4.4.3.1. Capacidad de condensación MT1M1.....	154
4.4.3.2. Acoplamiento MT1M2.....	155
4.4.3.3. Ejecución de Reutilización MT2M1.....	156
4.4.3.4. Capacidad de auditoría de seguimiento MT3M1.....	158
4.4.3.5. Diagnóstico de funciones MT3M2.....	159
4.4.3.6. Complejidad Ciclomática MT4M1.....	160
4.4.3.7. Profundidad de la herencia MT4M2.....	161
4.4.3.8. Complejidad de Modificación MT4M4.....	164
4.4.3.9. Complejidad Funcional del test MT5M1.....	165
4.4.4. Análisis de la Arquitectura de 3 capas basada en Clean Architecture.....	165
4.4.4.1. Capacidad de condensación - MT1M1.....	166
4.4.4.2. Acoplamiento - MT1M2.....	168
4.4.4.3. Ejecución de reutilización - MT2M1.....	169
4.4.4.4. Capacidad de seguimiento de auditoría - MT3M1.....	171
4.4.4.5. Diagnóstico de funciones- MT3M2.....	172
4.4.4.6. Complejidad Ciclomática - MT4M1.....	174
4.4.4.7. Profundidad de herencia - MT4M2.....	180
4.4.4.8. Complejidad de modificación - MT4M4.....	183
4.4.4.9. Complejidad funcional del test - MT5M1.....	185
4.4.5. Análisis de la Arquitectura de 4 capas basada en Clean Architecture.....	188
4.4.5.1. Capacidad de condensación - MT1M1.....	190
4.4.5.2. Acoplamiento- MT1M2.....	191
4.4.5.3. Ejecución de reutilización - MT2M1.....	192
4.4.5.4. Capacidad de seguimiento de auditoría - MT3M1.....	194
4.4.5.5. Diagnóstico de funciones- MT3M2.....	196
4.4.5.6. Complejidad ciclomática - MT4M1.....	198
4.4.5.7. Profundidad de herencia - MT4M2.....	204
4.4.5.8. Complejidad de modificación - MT4M4.....	207

4.4.5.9. Complejidad funcional del test - MT5M1.....	209
Capítulo V: Resultados y Discusión.....	214
5.1. Resultados del Grupo experimental.....	214
5.2. Resultados del Grupo de Control.....	215
5.3. Normalización de los resultados.....	216
5.4. Resultados por sub-características.....	218
5.4.1 Modularidad.....	218
5.4.2. Reusabilidad.....	220
5.4.3. Analizabilidad.....	221
5.4.4. Modificabilidad.....	222
5.4.5. Testabilidad.....	223
5.5. Resultados del Sistema.....	224
5.6. Discusión.....	228
Capítulo VI: Conclusiones.....	229
6.1. Conclusiones.....	229
6.1.1. Conclusión general.....	229
6.1.2. Conclusiones específicas.....	229
6.2. Trabajos futuros.....	230
Referencias bibliográficas.....	231
Anexos.....	238
Anexo A.....	238

ÍNDICE DE TABLAS

Tabla 1 Criterios de selección de artículos.....	33
Tabla 2 Estudios potencialmente elegibles y estudios seleccionados.....	34
Tabla 3 Métodos usados para la implementación del sistema.....	85
Tabla 4 Diseño de la transformación de los datos de entrada.....	89
Tabla 5 Historia de usuario número 1.....	96
Tabla 6 Historia de usuario número 2.....	98
Tabla 7 Historia de usuario número 3.....	100
Tabla 8 Historia de usuario número 4.....	103
Tabla 9 Historia de usuario número 5.....	105
Tabla 10 Historia de usuario número 6.....	108
Tabla 11 Product Backlog.....	110
Tabla 12 Funcionalidades del sistema.....	131
Tabla 13 Mantenibilidad y subcaracterísticas.....	134
Tabla 14 Análisis de mantenibilidad del proyecto.....	134
Tabla 15 Métricas de mantenibilidad.....	134
Tabla 16 Análisis de las herramientas de código estático.....	141
Tabla 17 Descripción de los casos de estudio.....	142
Tabla 18 Transformación de los datos de entrada para su posterior análisis.....	152
Tabla 19 Desglose de los parámetros de entrada para dependency cruiser.....	154
Tabla 20 Datos del grupo experimental.....	166
Tabla 21 Datos preprocesados del grupo experimental.....	166
Tabla 22 Datos del grupo control.....	189
Tabla 23 Datos preprocesados del grupo control.....	189
Tabla 24 Resultados del grupo experimental.....	214
Tabla 25 Resultados del grupo de control.....	215
Tabla 26 Normalización de las métricas.....	216
Tabla 27 Resultado normalizado del grupo experimental.....	217
Tabla 28 Resultado en normalizado del grupo de control.....	217
Tabla 29 Resultados del criterio de modularidad.....	219
Tabla 30 Resultados del criterio de reusabilidad.....	220
Tabla 31 Resultados del criterio de analizabilidad.....	221
Tabla 32 Resultados del criterio de Modificabilidad.....	223
Tabla 33 Resultados del criterio de Testabilidad.....	224
Tabla 34 Descripción de los resultados de los criterios de mantenibilidad.....	226
Tabla 35 Resultados en rendimiento promedio.....	227

ÍNDICE DE FIGURAS

Figura 1 Número de descarga de Google Play Store (Statista, 2022).....	24
Figura 2 Base de datos utilizadas.....	34
Figura 3 Arquitectura Limpia (Cecil, R., 2017).....	36
Figura 4 Modelo lógico de la arquitectura orientada (Rojas et al., 2021).....	37
Figura 5 Propuesta SAPL (Kerdoudi et al., 2022).....	39
Figura 6 Página de resultados de ScanMedicine (Sadek et al., 2022).....	40
Figura 7 Arquitectura monolítica (Milić, M., & Makajić-Nikolić, D. 2022).....	42
Figura 8 Enfoque propuesto para el desarrollo de sistemas turísticos (Naimi, et al., 2025).....	43
Figura 9 Arquitectura Multi Layered (Tesoriero et al., 2022).....	44
Figura 10 Clean Architecture basado para servicios internos (Aguilar & Figueira, 2020).....	46
Figura 11 Número de repositorios creados en GitHub por framework (Swacha et al., 2023).....	47
Figura 12 Sistema web desarrollado con Laravel y Vue.js (Avilés Matute, S., et al, 2020).....	48
Figura 13 Diferencia entre el VDOM y el Real DOM (Rathinam, S., 2022).....	49
Figura 14 Sistema creado con la librería de React JS. (Saundariya, K., et. al. ,2021).....	50
Figura 15 Arquitectura del Sistema de Lenguaje (Lei, Z., et al., 2022).....	52
Figura 16 Performance de los diferentes frameworks (Ollila, R., Mäkitalo, N., & Mikkonen, T., 2022). 53	
Figura 17 Gráfico de tiempos de ejecución al actualizar los componentes en un árbol de X componentes (Robayo, J., et al., 2024).....	54
Figura 18 Gráfico de comparativa de frameworks, costo versus tiempo de implementación (Rashid, R., et al., 2024).....	55
Figura 19 Instancias smells (García et al., 2021).....	56
Figura 20 Reporte de Patrones adoptados que fueron reportados por los entrevistados (Vale et al. 2022).....	57
Figura 21 Diseño de patrones de software (Piyawardhana et al., 2023).....	58
Figura 22 Patrón de diseño genérico basado en IEC 61499 para monitorear interacciones de componentes a nivel de aplicación (Jhunjhunwala et al., 2021).....	59
Figura 23 Evaluación de las métricas de Precisión, Recall y F1-Score para cada patrón de diseño (Houichime & El Amrani, 2025).....	60
Figura 24 Decisiones arquitectónicas en el desarrollo de software cuántico (Aktar et al., 2025).....	61
Figura 25 Porcentaje de proyectos en Python con anti-patrones detectados con posibles dependencias analizadas (Liu et al., 2024).....	62
Figura 26 Estructura e implementación del patrón de refactorización: Regla de 3 símbolos exportados (Yanakiev, I., et.al., 2025).....	63
Figura 27 Resultados de las métricas de la aplicación del PQEM en la segunda iteración en HeartCare (Falco, Scott & Robiolo, 2020).....	64
Figura 28 Ecuaciones de métricas de mantenibilidad según la norma ISO/IEC 25022:2016 (como se citó en Mena & Santorum, 2021).....	66
Figura 29 Métricas de calidad (Quiña-Mera et al., 2022).....	67
Figura 30 Calidad media de los modelos reparados (Cai et al., 2025).....	68
Figura 31 Smells en las métricas de modularidad y testabilidad (Jolak et al., 2025).....	69
Figura 32 Resultados de la medición de mantenimiento en el aplicativo del estudiante y profesor de myITS (Rahmi Dewi, Ngaliah & Rochimah, 2020).....	70

Figura 33 Performance del sistema (Baul Canlas et al., 2021).....	71
Figura 34 Evaluación sobre el grado de cumplimiento de la ISO 25010 en el sistema (Paliuanan et al., 2022).....	72
Figura 35 Diagrama de la propuesta arquitectónica.....	73
Figura 36 Clean Architecture.(Cecil, R , atl , 2017).....	75
Figura 37 Propuesta de diseño de Clean Architecture.....	76
Figura 38 Escenario genérico de interacción de componentes que debe ser monitoreado (Jhunhunwala et al., 2021).....	78
Figura 39 Patrón de diseño genérico basado en IEC 61499 para el monitoreo de interacción de componentes a nivel de aplicación (Jhunhunwala et al., 2021).....	78
Figura 40 Aplicación del patrón bridge en la Capa de Interfaz con React para Abstraer Servicios de Datos.....	79
Figura 41 Aplicación de la capa de uso, donde se define la funcionalidad de crear un servicio según los pasos definidos.....	81
Figura 42 Representación de entidad y atributos que constituyen en la capa entidad.....	82
Figura 43 Comunicación de capas de manera unidireccional.....	83
Figura 44 Selección de la tecnología del diseño Arquitectónico.....	84
Figura 45 Metodología de la implementación de la arquitectura de 3 capas.....	85
Figura 46 Método de creación de sesión.....	86
Figura 47 Método de creación de token.....	86
Figura 48 Método para obtener detalle de la cuenta.....	87
Figura 49 Método para obtener el estado de la cuenta.....	87
Figura 50 Método para obtener una película.....	88
Figura 51 Diagrama de Paginación.....	89
Figura 52 Diagrama general de diseño del sistema usando Clean Architecture de 3 capas.....	91
Figura 53 Diagrama de la capa interfaz para la aplicación del sistema de películas.....	92
Figura 54 Diagrama de la capa de uso para la aplicación del sistema de películas.....	93
Figura 55 Diagrama de la capa entidad para la aplicación del sistema de películas.....	94
Figura 56 Diagrama de vista modular.....	95
Figura 57 Diagrama de casos de uso.....	95
Figura 58 Diagrama de vista de componentes.....	96
Figura 59 Diagrama del repositorio.....	111
Figura 60 Diagrama de implementación del sistema.....	111
Figura 61 Captura de Pantalla de Visual Studio Code (Visual Studio Code).....	112
Figura 63 Diagrama de paquete del presenter (UI).....	112
Figura 64 Diagrama de paquetes de components.....	113
Figura 65 Estructura de un button.....	113
Figura 66 Estructura de la página Home del proyecto.....	114
Figura 67 Estructura de un Template.....	115
Figura 68 Diagrama de paquetes de los puertos.....	115
Figura 69 Módulo del Puerto privado o interno.....	116
Figura 70 Módulo del Puerto público o externo.....	117
Figura 71 Alerta de versión de node.....	117
Figura 72 Diagrama de paquete de la capa de uso.....	118

Figura 73 Estructura del repositorio de Usuario visitante.....	118
Figura 74 Estructura del repositorio de películas.....	119
Figura 75 Estructura del servicio de películas.....	120
Figura 76 Estructura de la entidad Usuario.....	121
Figura 77 Estructura de la entidad Película.....	122
Figura 78 Inicio de sesión del sistema.....	123
Figura 79 Cerrar sesión a través del sidebar.....	124
Figura 80 Landing del sistema.....	125
Figura 81 Visualizar carrusel de películas.....	125
Figura 82 Agregar lista de favoritos.....	126
Figura 83 Página de películas favoritas.....	126
Figura 84 Eliminar las películas de mis listas.....	127
Figura 85 Calificación de la película.....	127
Figura 86 Actualizar calificación a una película.....	128
Figura 87 Eliminar la calificación de las películas de mis listas.....	128
Figura 88 Agregar las películas de mis listas.....	129
Figura 90 Modo oscuro.....	130
Figura 91 Detalle de una película china.....	131
Figura 92 Proceso de Validación.....	133
Figura 93 Características del repositorio.....	139
Figura 94 Diagrama de preprocesamiento de los datos estáticos.....	140
Figura 95 Característica del repositorio luego del filtro.....	140
Figura 96 Diagrama de la transformación de los datos estáticos.....	141
Figura 97 Diagrama del caso 1 del grupo experimental.....	143
Figura 98 Diagrama del caso 2 del grupo experimental.....	144
Figura 99 Diagrama del caso 3 del grupo experimental.....	145
Figura 100 Diagrama del caso 4 del grupo experimental.....	146
Figura 101 Diagrama del caso 1 del grupo control.....	147
Figura 102 Diagrama del caso 2 del grupo control.....	148
Figura 103 Diagrama del caso 3 del grupo control.....	149
Figura 104 Diagrama del caso 4 del grupo control.....	150
Figura 105 Proceso de análisis usando SonarQube.....	151
Figura 106 Analizador de código estático SonarQube.....	153
Figura 107 Script de la métrica MT1M1.....	155
Figura 108 Script de la métrica MT1M2.....	156
Figura 109 Script de la métrica MT2M1.....	158
Figura 110 Script de la métrica MT3M1.....	159
Figura 112 Script de la métrica MT4M1.....	161
Figura 114 Segunda parte del Script de la métrica MT4M2 de análisis.....	163
Figura 118 Análisis de mantenibilidad del grupo experimental.....	166
Figura 119 Capacidad de condensación del grupo experimental caso 1.....	166
Figura 120 Capacidad de condensación del grupo experimental caso 2.....	167
Figura 121 Capacidad de condensación del grupo experimental caso 3.....	167

Figura 122 Capacidad de condensación del grupo experimental caso 4.....	168
Figura 123 Acoplamiento del grupo experimental caso 1.....	168
Figura 124 Acoplamiento del grupo experimental caso 2.....	168
Figura 125 Acoplamiento del grupo experimental caso 3.....	169
Figura 126 Acoplamiento del grupo experimental caso 4.....	169
Figura 127 Ejecución de reutilización del grupo experimental caso 1.....	170
Figura 128 Ejecución de reutilización del grupo experimental caso 2.....	170
Figura 129 Ejecución de reutilización del grupo experimental caso 3.....	170
Figura 130 Ejecución de reutilización del grupo experimental caso 4.....	171
Figura 131 Capacidad de seguimiento de auditoría del grupo experimental caso 1.....	171
Figura 132 Capacidad de seguimiento de auditoría del grupo experimental caso 2.....	172
Figura 133 Capacidad de seguimiento de auditoría del grupo experimental caso 3.....	172
Figura 134 Capacidad de seguimiento de auditoría del grupo experimental caso 4.....	172
Figura 135 Diagnóstico de funciones del grupo experimental caso 1.....	173
Figura 136 Diagnóstico de funciones del grupo experimental caso 2.....	173
Figura 137 Diagnóstico de funciones del grupo experimental caso 3.....	174
Figura 138 Diagnóstico de funciones del grupo experimental caso 4.....	174
Figura 139 Complejidad Ciclomática desde el SonarQube caso 1.....	175
Figura 140 Número total de funciones desde el SonarQube caso 1.....	176
Figura 141 Número total de funciones desde el SonarQube del grupo experimental caso 1.....	176
Figura 142 Complejidad Ciclomática desde el SonarQube caso 2.....	177
Figura 143 Número total de funciones desde el SonarQube caso 2.....	177
Figura 144 Número total de funciones desde el SonarQube del grupo experimental caso 2.....	177
Figura 145 Complejidad Ciclomática desde el SonarQube caso 3.....	178
Figura 146 Número total de funciones desde el SonarQube caso 3.....	178
Figura 147 Número total de funciones desde el SonarQube del grupo experimental.....	179
Figura 148 Complejidad Ciclomática desde el SonarQube.....	180
Figura 149 Número total de funciones desde el SonarQube caso 4.....	180
Figura 150 Número total de funciones desde el SonarQube del grupo experimental caso 4.....	180
Figura 151 Profundidad de herencia en el grupo de experimental caso 1.....	181
Figura 152 Archivo json de herencia en el grupo de experimental caso 1.....	181
Figura 153 Profundidad de herencia en el grupo de experimental caso 2.....	182
Figura 154 Archivo json de herencia en el grupo de experimental caso 2.....	182
Figura 155 Profundidad de herencia en el grupo de experimental caso 3.....	183
Figura 156 Archivo json de herencia en el grupo de experimental caso 3.....	183
Figura 157 Profundidad de herencia en el grupo de experimental caso 4.....	183
Figura 158 Archivo json de herencia en el grupo de experimental caso 4.....	183
Figura 159 Deuda técnica del grupo experimental caso 1.....	184
Figura 160 Deuda técnica del grupo experimental caso 2.....	184
Figura 161 Deuda técnica del grupo experimental caso 3.....	185
Figura 162 Deuda técnica del grupo experimental.....	185
Figura 163 Cobertura del grupo experimental caso 1.....	186
Figura 164 Coverage resultante del grupo experimental caso 1.....	186

Figura 165 Cobertura del grupo experimental caso 2.....	187
Figura 166 Coverage resultante del grupo experimental caso 2.....	187
Figura 167 Cobertura del grupo experimental caso 3.....	187
Figura 168 Coverage resultante del grupo experimental caso 3.....	187
Figura 169 Cobertura del grupo experimental caso 4.....	188
Figura 170 Coverage resultante del grupo experimental caso 4.....	188
Figura 171 Análisis de mantenibilidad del grupo de control.....	189
Figura 172 Capacidad de condensación del grupo de control caso 1.....	190
Figura 173 Capacidad de condensación del grupo de control caso 2.....	190
Figura 174 Capacidad de condensación del grupo de control caso 3.....	191
Figura 175 Capacidad de condensación del grupo de control caso 4.....	191
Figura 176 Acoplamiento del grupo de control caso 1.....	192
Figura 177 Acoplamiento del grupo de control caso 2.....	192
Figura 178 Acoplamiento del grupo de control caso 3.....	192
Figura 179 Acoplamiento del grupo de control caso 4.....	192
Figura 180 Ejecución de reutilización del grupo de control caso 1.....	193
Figura 181 Ejecución de reutilización del grupo de control caso 2.....	193
Figura 182 Ejecución de reutilización del grupo de control caso 3.....	194
Figura 183 Ejecución de reutilización del grupo de control caso 4.....	194
Figura 184 Capacidad de seguimiento de auditoría del grupo control caso 1.....	195
Figura 185 Capacidad de seguimiento de auditoría del grupo control caso 2.....	195
Figura 186 Capacidad de seguimiento de auditoría del grupo control caso 3.....	195
Figura 187 Capacidad de seguimiento de auditoría del grupo control caso 4.....	196
Figura 188 Diagnóstico de funciones del grupo de control caso 1.....	196
Figura 189 Diagnóstico de funciones del grupo de control caso 2.....	197
Figura 190 Diagnóstico de funciones del grupo de control caso 3.....	197
Figura 191 Diagnóstico de funciones del grupo de control caso 4.....	198
Figura 192 Complejidad Ciclomática desde el SonarQube del grupo de control caso 1.....	199
Figura 193 Número total de funciones desde el SonarQube del grupo de control caso 1.....	199
Figura 194 Complejidad ciclomática del grupo de control caso 1.....	199
Figura 195 Complejidad Ciclomática desde el SonarQube del grupo de control caso 2.....	200
Figura 196 Número total de funciones desde el SonarQube del grupo de control caso 2.....	201
Figura 197 Complejidad ciclomática del grupo de control caso 2.....	201
Figura 198 Complejidad Ciclomática desde el SonarQube del grupo de control caso 3.....	202
Figura 199 Número total de funciones desde el SonarQube del grupo de control caso 3.....	202
Figura 200 Complejidad ciclomática del grupo de control caso 3.....	202
Figura 201 Complejidad Ciclomática desde el SonarQube del grupo de control caso 4.....	203
Figura 202 Número total de funciones desde el SonarQube del grupo de control caso 4.....	203
Figura 203 Complejidad ciclomática del grupo de control caso 4.....	204
Figura 204 Profundidad de herencia en el grupo de control caso 1.....	204
Figura 205 Archivo json de herencia en el grupo de control caso 1.....	204
Figura 206 Profundidad de herencia en el grupo de control caso 2.....	205
Figura 207 Archivo json de herencia en el grupo de control caso 2.....	205

Figura 208 Profundidad de herencia en el grupo de control caso 3.....	206
Figura 209 Archivo json de herencia en el grupo de control caso 3.....	206
Figura 210 Profundidad de herencia en el grupo de control caso 4.....	207
Figura 211 Archivo json de herencia en el grupo de control caso 4.....	207
Figura 212 Complejidad de modificación del grupo de control caso 1.....	208
Figura 213 Complejidad de modificación del grupo de control caso 2.....	208
Figura 214 Complejidad de modificación del grupo de control caso 3.....	209
Figura 215 Complejidad de modificación del grupo de control caso 4.....	209
Figura 216 Diagrama Cobertura del grupo de control caso 1.....	210
Figura 217 Coverage del grupo de control caso 1.....	210
Figura 218 Diagrama Cobertura del grupo de control caso 2.....	211
Figura 219 Coverage del grupo de control caso 2.....	211
Figura 220 Diagrama Cobertura del grupo de control caso 3.....	212
Figura 221 Coverage del grupo de control caso 3.....	212
Figura 222 Diagrama Cobertura del grupo de control caso 4.....	213
Figura 223 Coverage del grupo de control caso 4.....	213
Figura 224 Diagrama comparativo de la modularidad.....	219
Figura 225 Diagrama comparativo para las métricas de reusabilidad.....	220
Figura 226 Diagrama comparativo para las métricas de Analizabilidad.....	221
Figura 227 Diagrama comparativo para las métricas de Modificabilidad.....	222
Figura 228 Diagrama comparativo para las métricas de Testabilidad.....	224
Figura 229 Diagrama comparativo de las métricas de mantenibilidad del sistema.....	224
Figura 230 Diagrama de línea de las métricas de mantenibilidad del sistema.....	225
Figura 240 Diagrama de radar de las métricas de mantenibilidad del sistema.....	226
Figura 241 Diagrama del porcentaje de mejora según criterio de mantenibilidad y sistema.....	228

Capítulo I: Introducción

1.1. Formulación del problema

1.1.1. Problema general

¿Cómo diseñar e implementar una Arquitectura para mitigar la complejidad inicial del mantenimiento de software basada en Clean Architecture?

1.1.2. Problema específicos

PE1: ¿Cuáles son los principios fundamentales y las características arquitectónicas más relevantes de la Clean Architecture que garantiza la mantenibilidad del software?

PE2: ¿Cómo se puede simplificar el diseño de la Clean Architecture para mitigar la complejidad inicial de la mantenibilidad?

PE3: ¿Cómo se realiza la implementación de un sistema que sirva de apoyo para corroborar la reducción de la complejidad inicial basada en Clean Architecture?

PE4: ¿De qué manera se valida y compara las métricas para evaluar el diseño arquitectónico propuesto en términos de mantenibilidad?

1.2. Antecedentes del problema

En los últimos años, el creciente desarrollo de aplicaciones web y móviles impulsó el auge del comercio electrónico. Este nuevo modelo de negocio ofrecía ventajas como la rapidez en procesos de venta, pero a la vez elevó la exigencia tecnológica para empresas, lo cual generó en empresas la necesidad de desarrollar sistemas más eficientes y de menor costo para seguir siendo competitivas.

Se ha realizado un estudio sobre la eficiencia y eficacia de un sistema, demostró que existen sistemas de gran envergadura que tienen un mayor costo que sobrepasó muchas veces el presupuesto de la empresa lo que trae consigo una disminución de los beneficios para las MYPEs como también medianas y grandes empresas cuyo objetivo era buscar mejorar su liquidez (Crespo-Martínez, E., et al., 2023).

Según otros autores, en un inicio, la complejidad de mantenimiento de un sistema de software era mínima. No obstante, con el fin de adaptarse a los nuevos requerimientos muchas arquitecturas crecieron desmesuradamente para cumplir con los requerimientos nuevos, debido a ello solían ser sistemas poco organizados y con mucho *code smell* esto suele ser nefasto para el mantenimiento de estos sistemas, lo cual representaba el 66% del costo total del sistema (Zurita, E., et al., 2022).

Para el desarrollo de sistemas de software, la arquitectura debe orientarse no solo a reducir la complejidad de mantenimiento, sino también garantice escalabilidad, resiliencia y sostenibilidad organizacional. (Gupta et al., 2021) proponen un marco de principios arquitectónicos que vinculaba decisiones técnicas con objetivos estratégicos, la cual demostró que una estructura flexible y alineada con indicadores de rendimiento puede minimizar costos operativos, evitar retrasos y reducir el impacto ambiental en sistemas destinados a usuarios finales.

Una aplicación perjudicó a una MYPE debido a su elevado costo y su complicado uso, requirió una capacitación adicional para el personal. El software debió estar facultado a poder adaptarse a las necesidades de la empresa, permitiendo su modificación y capacidad de ofrecer una mayor flexibilidad (Crespo-Martínez, E. et al., 2023).

Una arquitectura debió ser construida de la mejor manera posible, ya que representaba la satisfacción de los requisitos de calidad que impactarían con el tiempo la ejecución de la seguridad y la experiencia del usuario. Por lo que, fue recomendable optar por patrones de software y buenas prácticas de desarrollo con el fin de sistematizar el código y no impactar directamente en las responsabilidades del sistema, lo cual podría haber afectado su mantenibilidad (Aguilar, P., & Figueira, L. B., 2020).

La importancia de una arquitectura radicó en los atributos de calidad que ésta brindaba. Por ello, muchas arquitecturas pudieron tener variantes con la finalidad de ofrecer soluciones a problemas diferentes, como lo fueron los requisitos que surgieron durante el desarrollo diario. Al tomar una decisión de diseño se logró un "atributo-respuesta", lo que generó una táctica arquitectónica demandada para satisfacer los modelos propuestos bajo las decisiones tanto en la confiabilidad como en el rendimiento. Estos bloques de construcción fueron derivados de los patrones arquitectónicos. No obstante, persistió el desconocimiento por parte del diseño para lograr buenos sistemas de software y estructuras mantenibles (Márquez, G., Astudillo, H., & Kazman, R., 2023).

Debido al auge tecnológico, existió una gran acogida por el desarrollo de la tecnología, lo cual permitió la creación de 108.5 mil millones de aplicaciones en 2020. Sin embargo, estos números pudieron resultar engañosos al analizar desde el punto de vista de la mantenibilidad, ya que el 14% de estas aplicaciones presentó déficit. Esto se debió a que la norma ISO/IEC 25010, la cual evaluaba la analizabilidad, cambiabilidad y estabilidad frente a eventos inesperados, no fue cumplida por las métricas de estas aplicaciones. Mientras los sistemas crecieron, los proyectos fueron menos mantenibles y el acoplamiento aumentó (Francisco Arias-Orezano et al., 2021).

Clean Architecture se presentó como una solución eficaz para enfrentar los desafíos de mantenibilidad y escalabilidad en entornos ágiles. Su enfoque permitió estructurar el sistema en capas independientes pero bien definidas, donde cada una asumió una responsabilidad específica. Esta separación facilitó la adaptación a requisitos cambiantes sin comprometer la cohesión del sistema, y promovió una planificación arquitectónica desde las etapas iniciales del desarrollo (Arango y Loaiza, 2021)

1.3. Justificación

La calidad de un sistema de software depende, en su mayoría, de las decisiones arquitectónicas tomadas inicialmente. La literatura indica que arquitecturas complejas en fases tempranas incrementa el esfuerzo de mantenimiento y dificultan atributos de calidad como modularidad, testabilidad y analizabilidad. En el caso de Clean Architecture tradicional, el uso de múltiples capas y abstracciones puede generar una sobrecarga técnica cuando no es requerido por el proyecto, lo cual hace que la sostenibilidad se vea afectada. Autores como (Francisco Arias-Orezano et al., 2021) señaló que la sostenibilidad tecnológica de un sistema depende de evitar complejidades innecesarias desde la concepción del mismo, enfatizó que una arquitectura debe ajustarse al contexto, necesidades y tamaño del proyecto, así lo demostró en su propuesta de desarrollo móvil con aplicación de Clean Architecture y norma ISO/IEC 25010, mejoró significativamente su mantenibilidad aparte se notó un aumento en criterios como analizabilidad, estabilidad, testabilidad y cambiabilidad. Con los resultados se evidenció que la adopción de forma consciente de patrones contribuye con la sostenibilidad de los sistemas. En ese sentido, existe una base sólida para afirmar que las decisiones sobre arquitectura, hechas de forma temprana, son determinantes para la mejora en calidad y mantenimiento del software.

De acuerdo con (Klotins, E., Unterkalmsteiner, M., & Gorschek, T., 2023), las start-ups son una fuente de innovación al aprovechar tecnología de vanguardia para crear y lanzar nuevos productos de software; sin embargo, entre el 75% y 99% fracasaron en sus primeros años, tras el análisis de 88 casos, se identificó como causas recurrentes la incertidumbre del producto, la baja calidad del software y la ruptura de los equipos, lo que demostró que los fracasos no siempre provienen de problemas de mercado, sino también de deficiencias en la ingeniería y en la implementación del software. A pesar de ello, la creciente demanda con un incremento de 54.9 millones de descargas entre 2016 y 2022, equivalente a una mejora del 49.95% respecto al 2016 confirma la necesidad de seguir desarrollando las nuevas soluciones así como lo indica la Figura 1, entonces, en este sentido el uso una arquitectura, patrones y principios favorece la calidad y la sostenibilidad del software, así como reconocer tempranamente aquellas decisiones arquitectónicas que pueden conducir al fracaso de un proyecto.

Basado en (Jolak et al., 2025), en los últimos años se ha evidenciado que la calidad del código no solo depende de su uso, sino del producto en sí mismo. Por esta razón, los atributos de calidad exigidos por la norma son importantes, ya que su ausencia puede elevar el riesgo de propagación de errores. Estos hallazgos justifican el uso de métricas de software, ya que permiten identificar y cuantificar las decisiones del diseño.

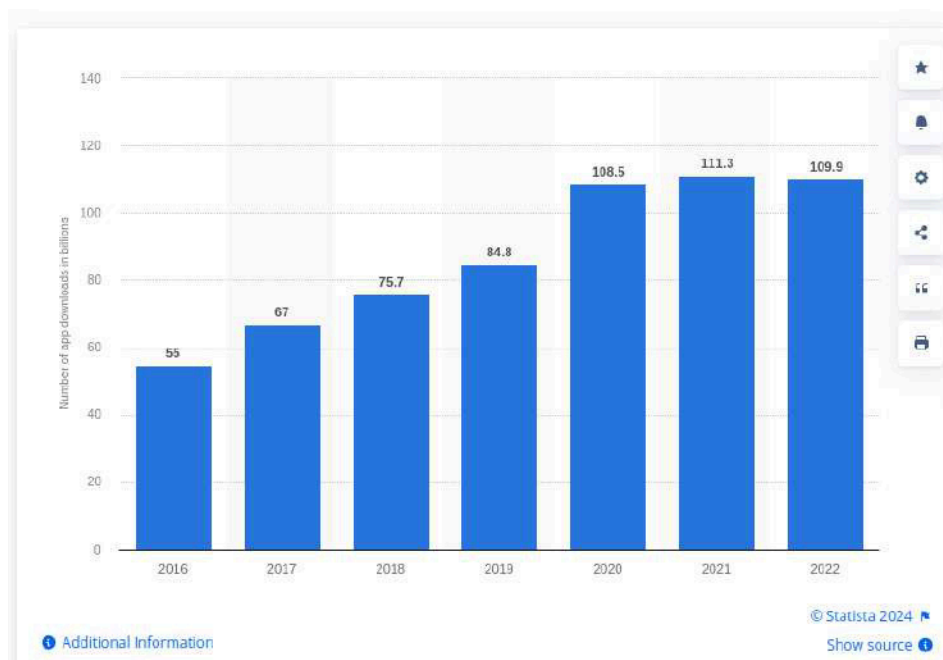


Figura 1 Número de descarga de Google Play Store (Statista, 2022)

1.4. Objetivos

1.4.1. Objetivo General

El objetivo de la presente tesis es diseñar una arquitectura para mitigar la complejidad inicial de mantenimiento del software basada en Clean Architecture con el fin de mejorar la mantenibilidad de los sistemas ante el cambio que se exige en la actualidad.

1.4.2. Objetivos Específicos

OE1: Analizar y revisar diferentes tipos de arquitectura para implementar un sistema.

OE2: Diseñar una Arquitectura para un sistema que disminuya la complejidad del mantenimiento basado en Clean Architecture.

OE3: Desarrollar un sistema que sirva como apoyo para corroborar la reducción de la complejidad inicial mantenimiento basado en Clean Architecture.

OE4: Validar la mantenibilidad del diseño arquitectónico considerando el caso de estudio de un proyecto.

1.5. Organización de la tesis

El presente trabajo se organizó en 7 capítulos:

En el Capítulo I, se abordó el planteamiento del problema, así como la descripción de los antecedentes, los cuales ayudaron a consolidar una propuesta de solución.

En el Capítulo II, se presentó el marco teórico, con una explicación de los conceptos fundamentales que abarcó la investigación.

En el Capítulo III, se analizó el estado del arte, el cual incluyó propuestas realizadas por otros autores a través de artículos científicos. Esto brindó conceptos clave para el diseño e implementación de una arquitectura y el uso de tecnologías basadas en Clean Architecture.

En el Capítulo IV, se explicó la metodología de investigación, donde se detalló el proceso de desarrollo de la propuesta: evaluación, diseño, implementación y validación.

En el Capítulo V, se describieron los resultados y se discutieron los datos hallados

En el Capítulo VI, se expusieron las conclusiones generales y específicas del proceso de investigación, así como posibles líneas de trabajo futuro.

Capítulo II: Marco Teórico

2.1. Arquitectura Software

La Arquitectura Software es el arte de establecer límites y separar los elementos de una aplicación. La separación benefició al proceso de construcción del software, ya que facilitó la estructuración y organización de forma más eficiente. Para lograr un mejor resultado, fue fundamental que la arquitectura evite la contaminación del núcleo de la lógica de negocio, de modo que no se introdujo dependencias innecesarias y se minimizó los recursos requeridos. La lógica de negocio no debía depender de utilidades, bases de datos o librerías externas, ya que estas no deben influir de manera significativa en la arquitectura del sistema (Cecil, R., 2017).

2.2. Tipos de Arquitecturas

2.2.1 Arquitectura Limpia

La arquitectura limpia se centra en la separación de responsabilidades tomando como referencia la organización por niveles, de elementos más externos al núcleo del sistema. El principio es buscar la independencia del código interno de la lógica de los niveles externos. Así también, el transporte de datos debe permanecer separado entre las distintas capas para evitar acoplamientos innecesarios. La arquitectura limpia busca estructurar y organizar el proyecto de forma sistemática y comprensible, lo cual facilita su escalabilidad y mantenimiento a largo plazo (Martinez, J. C., et al., 2021).

2.2.1.1 Capa de entidad

En esta capa se encapsula las reglas de negocio del proyecto y define su estructura de datos. Debido a su propósito, debe tener la menor probabilidad de ser modificada. En caso se realicen cambios en otras partes de la aplicación, estos cambios no deberían afectar la capa de entidad (Cecil, R., 2017).

2.2.1.2 Capa de casos de Uso

Esta capa encapsula la lógica de negocio y coordina la interacción entre las entidades de un sistema. Su función principal es definir las acciones que la aplicación debe ejecutar (Cecil, R., 2017).

2.2.1.3 Capa Adaptador

En esta capa se integran elementos externos como la interfaz de usuario y servicios externos. Su propósito es transformar los datos externos en un formato que sea comprensible para las capas internas de la arquitectura (Cecil, R., 2017).

2.2.1.4 Capa de Presentación

Esta capa contiene librerías y *frameworks*, herramientas usadas en la construcción del software. Se considera la capa más variable dentro de la arquitectura, ya que está sujeta a cambios debido a las constantes actualizaciones y cambios tecnológicos (Cecil, R., 2017).

2.2.2 Arquitectura Software Monolítica

Esta arquitectura mantiene unificados los módulos, lo cual conlleva a la pérdida de independencia entre ellos, aunque cada uno sigue teniendo sus propias responsabilidades. Esta estructura implica una ejecución compilada reunida que se representa como única entidad. Debido a la integración, los cambios en los módulos sugieren un reensamblado en la arquitectura (Milić, M., & Makajić-Nikolić, D., 2022).

2.2.3 Arquitectura basado en Microservicios

El diseño de esta arquitectura consta en agrupar las funcionalidades en pequeños grupos de servicios distribuidos entre sus módulos, funcionando como un proceso cohesivo e independiente. Cada servicio puede comunicarse a través de un protocolo común y pueden ser contruidos con diferentes lenguajes, tecnologías y bases de datos (Milić, M., & Makajić-Nikolić, D., 2022).

2.3. Principios Arquitectónicos

2.3.1. Single Responsibility

La responsabilidad única establece que cada módulo de un software tiene una responsabilidad sobre una parte funcional, y esta responsabilidad debe estar encapsulada en su totalidad por la clase. Este principio se basa en el corolario de Conway (Cecil, R., 2017).

2.3.2. Open-Closed Principle

Según Bertrand Meyer, el principio de abierto-cerrado establece que los sistemas de software deben ser resilientes a los cambios y a la adición de nuevos códigos. Lo cual implica que los módulos deben estar abiertos a la extensión, hace referencia a permitir agregar nuevas funcionalidades sin modificar el comportamiento de los módulos existentes (Cecil, R., 2017).

2.3.3. Interface Segregation Principle

Este principio recomienda evitar depender de módulos que no son partícipes fuertemente en el software, por lo que los clientes que usan este sistema solo deben acceder a métodos que realmente les sirvan logrando un sistema muy bien desacoplado respecto a otros, dando efectos como modificación y refactorización (Cecil, R., 2017).

2.3.4. Dependency Inversion Principle

El principio de inversión de dependencia sugiere que el diseño de software debe estar desacoplado por módulos específicos dentro del sistema. El desarrollo debe implementar políticas de alto nivel para que el sistema no dependa de los detalles de bajo nivel, como el código con el que interactúa. Esta interacción abstracta permite reducir el sistema como también adicionar patrones de codificación adicionales (Cecil, R., 2017).

2.4. Mantenibilidad

La mantenibilidad en el software contempla criterios como analizabilidad, cambiabilidad, estabilidad y la capacidad de ser probado según las normas ISO/IEC 25010. Esto representa la eficiencia con la que los sistemas de software pueden evolucionar a través de medidas correctivas o perfectivas (Francisco Arias-Orezano, et al, 2021).

2.5. Tecnologías

2.5.1 React.js

Es una biblioteca desarrollada en JavaScript para desarrollar código abierto con procesos mucho más rápido debido a la integración del Virtual DOM que realiza renderizados mucho más rápido lo que genera el agrado entre los programadores. Además tiene un excelente soporte por parte de Facebook lo cual contribuyó con su rápido desarrollo y adopción en el ámbito del frontend (Saundariya, K., et al., 2021).

2.5.2 Node.js

Node Js es una herramienta que sirve para el desarrollo de backend en plataformas de tipo de servidor con un flujo de datos de tipo asíncrono. Los trabajos de servidor son mucho más rápidos debido a su cantidad de componentes, el cual genera el código de alta escalabilidad y rendimiento entre las aplicaciones más modernas (Saundariya, K., et al., 2021).

2.5.3 Base de Datos NoSQL

Esta base de datos brinda mejores resultados con datos no estructurados mejorando a comparación a base de datos tradicionales, ya que están libres de esquemas y con una gran distribución de datos, ofreciendo a los sistemas una mayor escalabilidad y confiabilidad apropiada para datos heterogéneos y problemas contemporáneos (Eyada, M. et al., 2020).

2.6. Aplicación Web

Las aplicaciones web son sistemas que se ejecutan directamente en navegadores, sin necesidad de una instalación local, permitiendo a los usuarios interactuar con las funcionalidades alojadas en un servidor web. Estas aplicaciones constituyen una pieza fundamental en el ecosistema digital, debido a la integración de herramientas que ayudan en la optimización de la experiencia del usuario y la eficiencia del sistema (Patiño-Vásquez & Llerena-Izquierdo, 2021).

2.7. Patrones de diseño

2.7.1 Adapter

El patrón *adapter* (adaptador) tiene como propósito convertir la interfaz de una clase en otra que el cliente espera, permitiendo que clases con interfaces incompatibles colaboren sin necesidad de modificar su código original. Este patrón actúa como un puente entre dos interfaces, encapsulando la lógica de adaptación en una clase separada (Gamma et al., 2003).

2.7.2 Strategy

Patrón de diseño *strategy* (estrategia) que permite definir una familia de algoritmos, encapsular cada uno y hacerlo intercambiables. Esto habilita al algoritmo para que sea variable independientemente del cliente que lo esté usando. Un caso útil, es para la selección de múltiples comportamientos sin usar condiciones extensas (Gamma et al., 2003).

2.7.3 Bridge

El patrón de diseño *bridge* (puente) desacopla una abstracción de su implementación, permitiendo que ambas evolucionen de forma independiente. Es útil cuando se desea evitar una dependencia rígida entre la interfaz e implementación, especialmente en sistemas que se deben adaptar a múltiples plataformas o dispositivos (Gamma et al., 2003).

2.7.4 Composite

El patrón de diseño *composite* (composición) permite componer estructuras jerárquicas de tipo árbol, tratando de forma uniforme objetos individuales y compuestos. Es útil para representar relaciones parte-todo (Gamma et al., 2003).

2.7.5 Builder

El patrón *builder* (constructor), se utiliza para separar la construcción de un objeto complejo de su representación, de modo que la separación permite generar diferentes representaciones. Presenta utilidad cuando se requiere la creación de un objeto con múltiples pasos o configuraciones, y donde se requiera mantener una flexibilidad y claridad (Gamma et al., 2003).

2.8. Normas

2.8.1 ISO/IEC 25010

La norma ISO/IEC 25010 define criterios para evaluar la calidad del software, útiles en decisiones arquitectónicas durante el desarrollo de software. Permite orientar el diseño del software hacia soluciones escalables, modulares y alineadas con los requisitos del usuario, siguiendo con las características de mantenibilidad, eficiencia de desempeño y adecuación funcional (ISO, 2011).

2.8.2 ISO/IEC 25022

La norma ISO/IEC 25022, define métricas específicas para evaluar la calidad en uso del software, esto indica saber cómo el producto satisface las necesidades del usuario en condiciones reales de operación. Se incluyen indicadores como efectividad, eficiencia, protección contra riesgos, cobertura contextual y satisfacción, que son útiles para validar el impacto de un sistema después de su despliegue (ISO, 2016).

Capítulo III: Estado del Arte

En el presente capítulo, se presentó una visión general de toda la literatura revisada durante el proceso de investigación. Se aplicó la metodología para desarrollarla en 3 fases: planificación de la revisión, desarrollo de la revisión y resultados de la revisión. Además, se detalló la estructura taxonómica aplicada para analizar los resultados obtenidos por la investigación realizada.

3.1. Metodología de revisión de la literatura

La metodología de revisión literaria consistió en realizar una planificación de revisión de la literatura planteando preguntas de investigación, para posteriormente desarrollar la revisión de artículos y analizar los resultados analizados por los artículos seleccionados.

En esta revisión literaria, se hizo el empleo de la metodología planteada por Wong et al. (2017), la cual se determinó en 3 fases: (i) Planificación de la Revisión: donde se elaboraron las preguntas de investigación y se definió el protocolo de revisión; (ii) Desarrollo de la Revisión: donde se eligieron los estudios primarios, según los criterios de selección o exclusión suscritos; y (iii) Los Resultados de la Revisión: donde se analizaron los estudios seleccionados, presentando un análisis detallado y estadística de los mismos.

- Pregunta 1 ¿Qué arquitecturas permiten implementar un sistema de software?
- Pregunta 2 ¿Qué frameworks permiten desarrollar un sistema de software?
- Pregunta 3 ¿Qué principios y patrones permiten construir una arquitectura de Software?
- Pregunta 4 ¿Qué métricas son necesarias para que un proyecto sea mantenible y escalable con el tiempo?

La investigación abarcó el periodo de enero del 2020 a agosto del 2025. Para la búsqueda de información se utilizó una estrategia basada en palabras clave vinculadas al tema de estudio, tales como “Clean Architecture”, “Framework web”, “Design software architecture”, “Development”, “Architecture”, “Web”, “Software”, “Metrics” y “Maintainability”. Estas palabras se combinaron mediante el uso de operadores booleanos (AND, OR, NOT) y filtros de exclusión, con el fin de precisar los resultados y evitar temas no relacionados, como Inteligencia Artificial, Machine Learning, Realidad Virtual o Large Language Models.

Asimismo, se emplearon filtros avanzados en los buscadores académicos para optimizar la relevancia de los estudios recuperados.

Finalmente, se aplicaron los criterios de selección y exclusión mostrados en la Tabla 1 que garantizan la pertinencia de los artículos analizados.

Tabla 1 *Criterios de selección de artículos*

Criterio de Selección	Criterio de Exclusión
Publicaciones en español e inglés	Duplicados o versiones preliminares de publicaciones finales
Publicaciones desde el 2020 hasta la actualidad	Casos o estudios que no poseen evaluación de mantenibilidad o complejidad
Artículos científicos o conferencias	Trabajos que solo presenten herramientas sin discusión arquitectónica o de mantenibilidad
Publicaciones de acceso libre	Publicaciones que no tengan relación a la arquitectura de software

3.2. Desarrollo de la revisión

En el proceso de búsqueda de la información para esta investigación, se llevó a cabo una revisión exhaustiva de estudios seleccionados, aplicando los criterios específicos ya mencionados en el punto anterior. Durante esta fase de desarrollo de la investigación, se evaluó el contenido de cada estudio para determinar su relevancia para el tema de investigación. Como resultado, se descartó la mayoría de los estudios que no se enfocaron en el ámbito de desarrollo de software ni en tecnologías web, sino que se abordaron otras áreas. Este proceso de selección riguroso permitió identificar únicamente los estudios que presentaron información actualizada y acorde a la evaluación de arquitectura de software, frameworks web. Con base en estos estudios, se llevó a cabo una investigación completa para definir la mejor arquitectura que permitiera implementar un sistema web, y el mejor framework para desarrollar un sistema web que permitiera que el proyecto fuera escalable y flexible. El proceso desarrollado se encontró en la Figura 2.

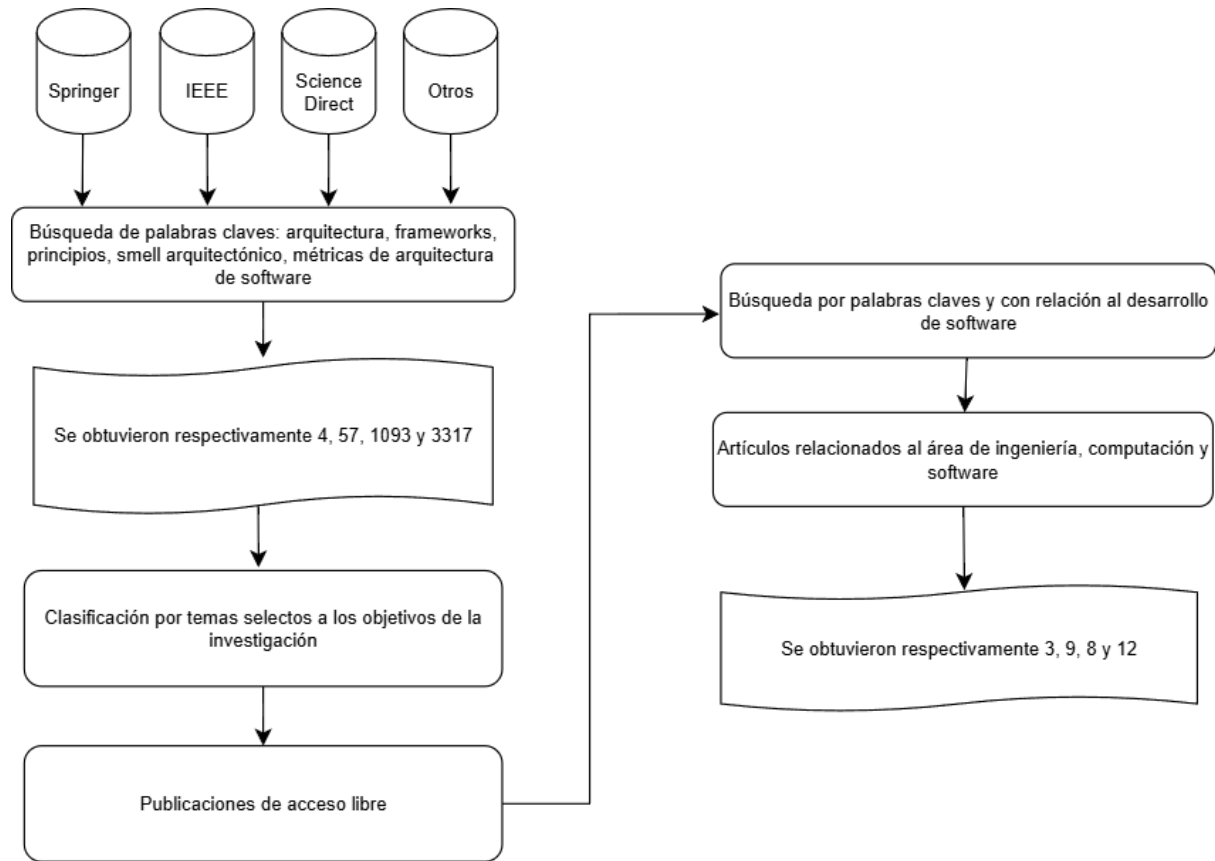


Figura 2 Base de datos utilizadas

3.3. Resultados de revisión

Las estrategias iniciales de búsqueda identificaron diferentes cantidades de referencias por cada búsqueda mencionada anteriormente. Como resultado, se obtuvo un total de 135,952 estudios. En primera instancia, se seleccionaron 4,471 investigaciones que cumplieran con los criterios establecidos, los cuales fueron objeto de sucesivos cribados de acuerdo a los objetivos del tema de investigación, resultando en 32 publicaciones finales. En la Tabla 2, fue posible ver la cantidad de estudios seleccionados por cada tipo de fuente.

Tabla 2 Estudios potencialmente elegibles y estudios seleccionados

Fuente	Estudios potencialmente elegibles	Estudios seleccionados
Springer	4	3
IEEE	57	9
ScienceDirect	1093	10

Otros	3317	10
Total	3317	32

3.4. Revisión literaria

3.4.1. Arquitecturas para implementar un sistema Web

Según el estudio realizado por (Arias-Orezano et al., 2021), se analizó el impacto de la implementación del Clean Architecture y la norma ISO/IEC 25010, que incluyó una muestra de 2037 clases y 168,217 líneas de código, el objetivo fue comprender criterios fundamentales para una buena arquitectura de software, tales como la estabilidad, testabilidad, cambiabilidad y analizabilidad, abarcando el desarrollo móvil y web. La metodología descrita es prescriptiva, ya que buscó ofrecer una solución enfocada en la mejora de la mantenibilidad del software, y cuantitativa, debido al uso de métricas específicas. Para el estudio se desarrolló una aplicación móvil bajo el nombre de Educar Teacher, basada en Clean Architecture. Se empleó Java, compatible con dispositivos Android a partir de la versión 4.0. Los servicios REST fueron construidos en C#, con la posibilidad de operar en entornos sin conexión a internet. Además, se aplicaron los principios SOLID, lo cual permitió una implementación estratégica de patrones de diseño.

Posteriormente, se evaluó las métricas de criterios de mantenibilidad usando normas ISO/IEC 25010, bajo el análisis de las métricas se encontró, a través del Diagrama de Kiviata para analizar la complejidad, que Clean Architecture alcanzó los 24.58 dentro del modelo a diferencia de otras como MVC que tenía 26.2. En cuanto a la métrica CL_CMOOF (cantidad de comentarios por función), la arquitectura limpia mostró un 6.06%, frente al 9.83% del sistema CRM Distribution, lo cual reflejó una menor dependencia de comentarios innecesarios y una preferencia por comentarios documentales. Así mismo, la métrica CL_DATA_PUBLIC (atributos públicos por clase), la arquitectura limpia presentó un promedio de 6.22, mientras que el CRM obtuvo 1.36, lo que sugirió que Clean Architecture.

En base a la norma ISO/IEC 25010, los resultados obtenidos de comparar Clean Architecture con CRM Distribution, fueron los siguientes: analizabilidad (49.26 vs. 52.55), estabilidad (20.9 vs. 9.24), testabilidad (49.39 vs. 49.04) y cambiabilidad (40.96 vs. 40.58). Los datos reflejaron mejoras del 7% en analizabilidad, 56% en estabilidad, 0.7% en testabilidad y 0.9% en cambiabilidad al utilizar Clean Architecture en combinación con

ISO/IEC 25010, frente a un sistema tradicional como CRM Distribution, basado en el patrón MVC. La arquitectura propuesta se mostró en la Figura 3.

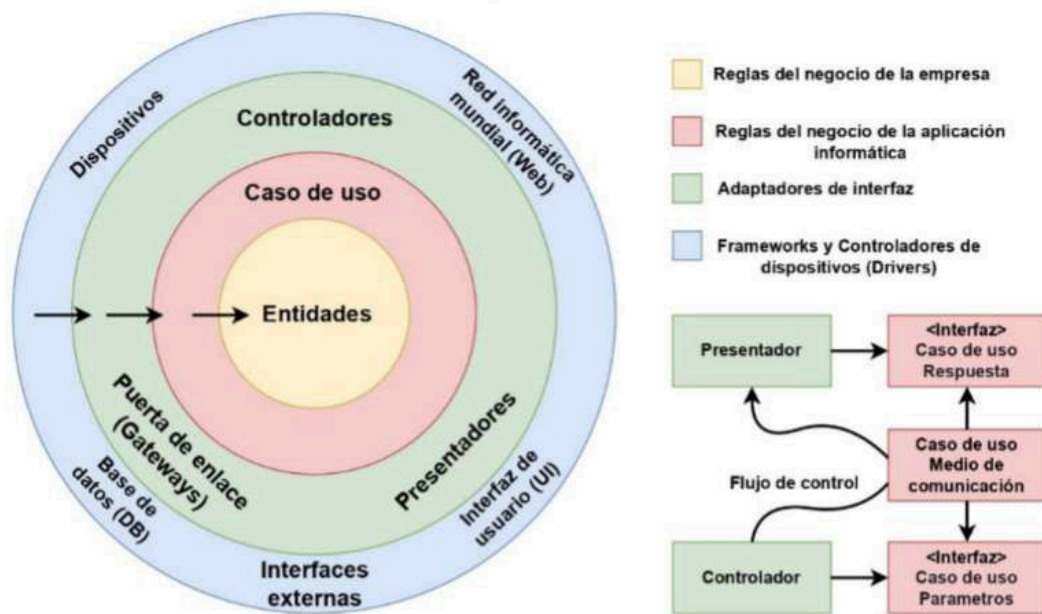


Figura 3 Arquitectura Limpia (Cecil, R., 2017)

Según lo planteado por los autores (Rojas et al., 2021), las pequeñas, medianas y microempresas representaron la mayor parte del sector empresarial y tuvieron el principal reto de ofrecer servicios de calidad, los cuales solían estar cubiertos por recursos tecnológicos, financieros y humanos limitados. La adopción de arquitecturas orientadas a servicios (SOA), aunque tuvo el beneficio de interoperabilidad y flexibilidad, solía tener altos costos en personal especializado e infraestructura, lo cual la hizo difícil de implementar en dicho tipo de organización que se encontraba en crecimiento. El principal objetivo del trabajo fue diseñar, administrar e implementar un modelo de SOA, que se encontrara adaptado a las necesidades de la empresa de desarrollo de software Codideep EIRL, para optimizar recursos e infraestructura, reducir costos, mantenimiento, mejorar el rendimiento y aumentar la agilidad del negocio.

Se presentó el desarrollo de una propuesta arquitectónica basada en SOA, a partir de un modelo que abstraigo el negocio en 3 servicios centrales: servicio de chat, servicio de negocio y servicio de compañía (la Figura 4 detalló el diagrama). Se definió un modelo lógico y físico que incluyó componentes como clientes consumidores, objetos de servicio, capas de negocio, validaciones, acceso a datos y base de datos. Las tecnologías usadas para la implementación fueron Node.js y MongoDB para el servicio de Chat, Laravel (PHP) con MySQL para

procesos del negocio, y .NET Core con SQL Server para la gestión de información corporativa. Así también, se realizaron adaptaciones a la arquitectura orientada a servicios (SOA) tradicional, de las que destacan la sustitución de la capa de contrato de servicios por Service Objects más flexibles, la integración combinada de REST y sockets con el fin de optimizar los tiempos de respuesta, y la descentralización del almacenamiento de datos mediante el uso de diferentes motores de base de datos según el tipo y la criticidad de la información.

La arquitectura propuesta por los autores permitió mejorar la agilidad empresarial y optimizar el uso de recursos, reduciendo costos frente a una arquitectura monolítica y SOA estándar. La adaptación de eliminar la capa de contrato y el uso de los Service Objects apoyó en la disminución de carga de datos y la mejora de rendimiento; la combinación de REST con Sockets redujo significativamente los tiempos de respuesta; y la descentralización del almacenamiento evitó cuellos de botella, permitiendo asignar recursos según la demanda de cada servicio. Finalmente, la arquitectura SOA adaptada y moderada demostró ser viable y rentable para microempresas, ya que ofreció flexibilidad y escalabilidad sin llegar a los altos costos típicos en implementaciones SOA convencionales.

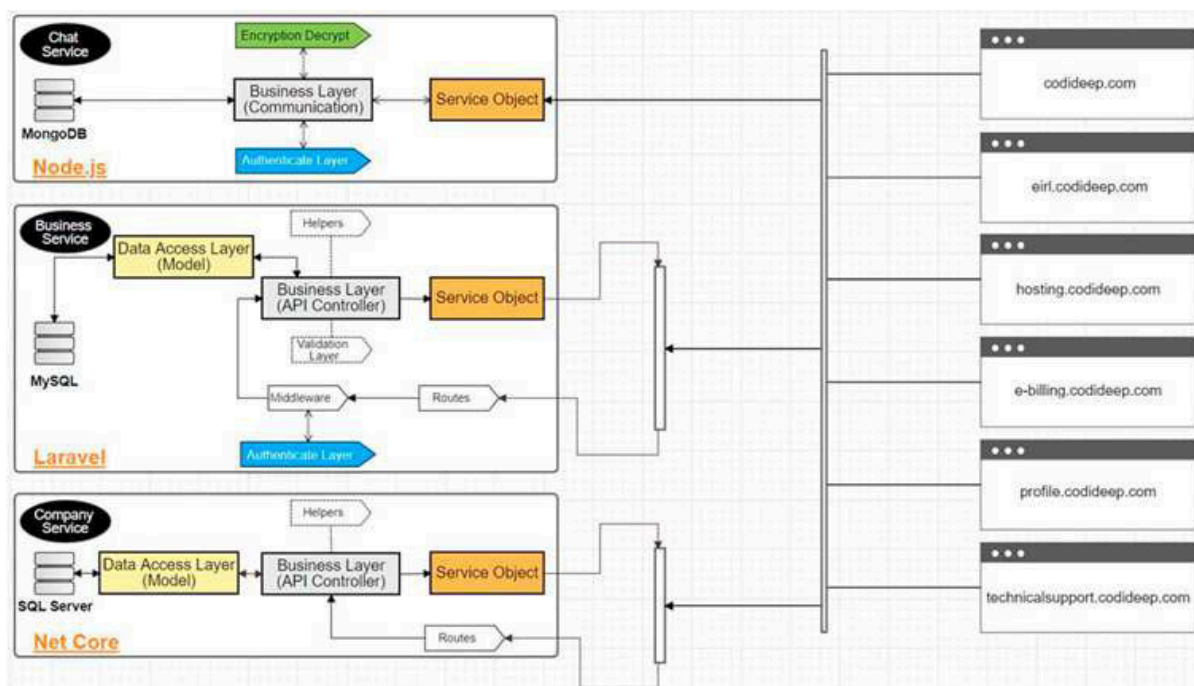


Figura 4 Modelo lógico de la arquitectura orientada (Rojas et al., 2021)

(Kerdoudi et al., 2022) propusieron una visión donde se consideró una arquitectura basada en Software Product Line (SPL), de la cual se pudo derivar múltiples variantes de arquitectura

de software, cada una con su propia función y ciclo de vida, que se encontraban enfocadas a las necesidades evolutivas del sistema que depende del entorno específico del software. Para ello, se utilizó el modelo OSGi (Open Service Gateway Initiative), que permitió generar diversas arquitecturas de software desde diferentes puntos de vista. El diseño de los metamodelos permitió a los desarrolladores entender progresivamente las variantes del sistema. El primer paso dentro del proceso ascendente fue aplicar técnicas de ingeniería inversa, lo que implicó el análisis del código y los archivos de configuración. Las variantes identificadas se emplearon como modelo para la construcción de SAPL (Software Architecture Product Line), aplicando el activador del marco BUT4Reuse, el cual permitió manejar artefactos arquitectónicos y detectar bloques reutilizables a partir de un conjunto de variantes. El enfoque propuesto se detalló en la Figura 5.

Para ello, se utilizó un algoritmo de elementos interdependientes usando la equivalencia de clases, el cual se evaluó por dos procesos: uno de abajo hacia arriba, que permitió reconstruir el SPL, y otro de ingeniería avanzada, que facilitó la derivación de nuevas variantes. Las variantes fueron evaluadas con métricas de rendimiento, así también, con métricas de cambios arquitectónicos como A2A y MojoFM. El trabajo desarrollado por los autores fue relevante en proyectos de gran escala y nivel de implementación. Con apoyo en herramientas gráficas de SAPL para definir restricciones y funciones, aunque el enfoque requirió investigación, los resultados mostraron que fueron aplicables en conjuntos de datos que presentaron alta variabilidad en productos de software.

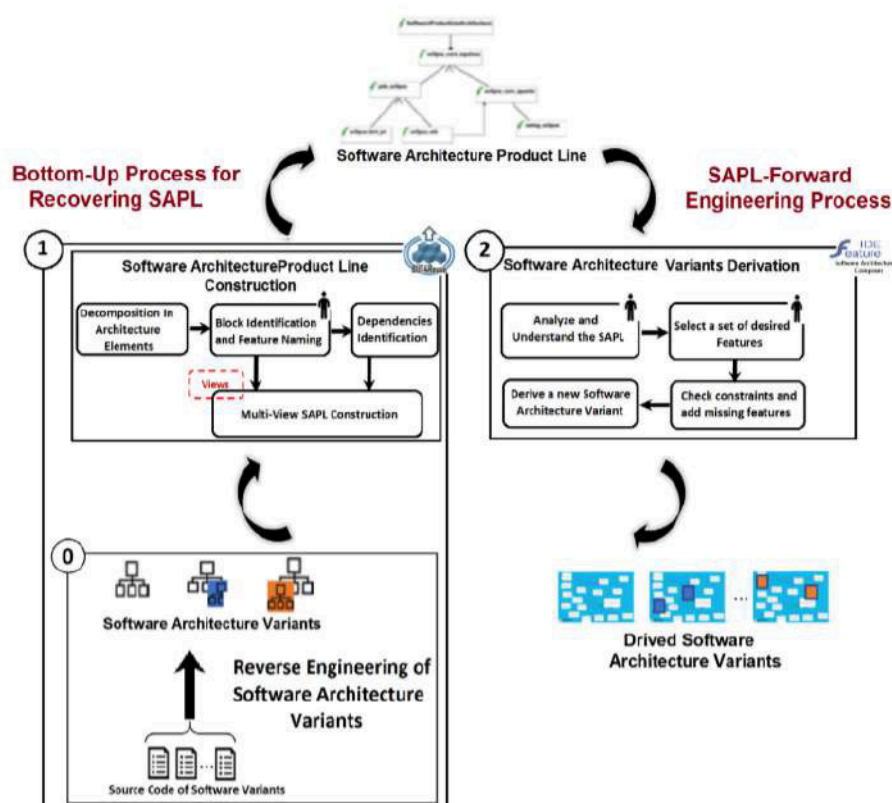


Figura 5 Propuesta SAPL (Kerdoudi et al., 2022)

La investigación realizada por (Sadek et al., 2022) indicó que el acceso a información sobre tecnologías sanitarias, como dispositivos médicos y ensayos clínicos, se vio limitado por la fragmentación de fuentes, la falta de integración y retrasos en la actualización de datos. Este fue un problema que dificulta la labor de profesionales de salud, investigadores y pacientes que necesitaron datos fiables y acertados para la toma de decisiones. El objetivo fue diseñar e implementar un sistema de búsqueda (ScanMedicine) que centralizó y optimizó el acceso a datos y que combinara arquitecturas de microservicios y serverless para lograr flexibilidad, escalabilidad y reducción de costos. Respecto a la arquitectura, se adaptó un estilo arquitectónico por capas, en la que cada capa ejecutó hardware independiente y usó técnicas distintas. La capa de núcleo usó microservicios para procesos que tomaron mucho tiempo y manejaron datos, para casos de web crawling, indexación y lenguaje natural; la capa de integración coordinó servicios y aseguró la escalabilidad y tolerancia de fallos; y la capa de experiencia expuso las funcionalidades a usuarios mediante el uso de funciones serverless, optimizando tiempos de respuesta y costos para tareas pequeñas y sin estado. La

implementación de la propuesta se realizó sobre AWS, integrando tecnologías como Docker, RabbitMQ, Spring Cloud, MongoDB, Elasticsearch y API Gateway.

Como resultados, los autores crearon un sistema que permitió búsquedas rápidas y filtradas, como se observó en la Figura 6, que incluyeron visualización interactiva de resultados y acceso unificado a datos de ensayos clínicos y dispositivos médicos. El uso combinado de microservicios y serverless superó limitaciones propias de cada enfoque por separado, dando como resultado un diseño extensible, modular, compatible con tecnologías comerciales y de código abierto. Los autores indicaron que la arquitectura híbrida fue adecuada para entornos con demandas variables y requisitos de alta disponibilidad, y plantearon mejoras futuras para la incorporación de analíticas de big data y minería de texto para extraer nuevo conocimiento.

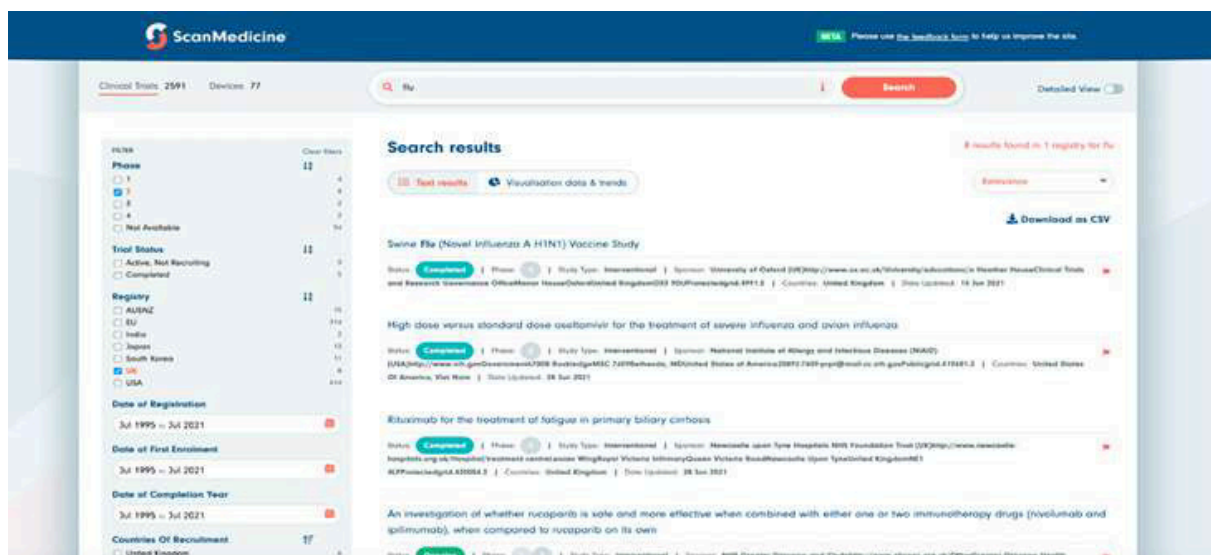


Figura 6 Página de resultados de ScanMedicine (Sadek et al., 2022)

En el estudio realizado por (Milić et al., 2022), se realizó una comparación entre una arquitectura monolítica y una de microservicios, ambas implementadas con Jakarta EE. El objetivo principal de la investigación consistió en el desarrollo de un modelo cuantitativo para optimizar la selección de la arquitectura de software basándose en atributos de calidad. Los autores realizaron una evaluación de ambas arquitecturas mediante un conjunto de métricas cuantitativas y cualitativas que incluyeron la seguridad, la complejidad del código, el tiempo promedio de compilación, la disponibilidad del sistema (se midió a través de *health checks*), la cobertura de las pruebas y la modularidad. Los resultados empíricos demostraron que la arquitectura de microservicios ofrecía una mayor independencia y modularidad pero podía conducir a una menor cobertura de pruebas y a la violación del principio “No repetirse”

(DRY). A diferencia de la arquitectura monolítica, que era más simple de probar, poseía un alto acoplamiento entre sus componentes, en la Figura 7, se mostró la arquitectura monolítica evaluada.

Respecto a las amenazas de validación o limitaciones, se debieron por el uso de tecnologías específicas usadas como Jakarta EE y Spring, y se advirtió que los resultados podrían ser diferentes en sistemas con mayor complejidad, otros dominios de software u otros frameworks. En adición, los autores señalaron que solo hubo un análisis de un subconjunto de atributos de calidad del software, y que una evaluación más amplia, incluyendo sus interdependencias y más métricas, era requerida para obtener resultados que se pueda generalizar. Por otra parte, el estudio concluyó que no existía una arquitectura universalmente superior; la elección dependía de los atributos de calidad que se priorizaron en un proyecto. La principal contribución fue el modelo matemático MIGP, el cual fue una herramienta cuantitativa que permitía a los equipos tomar decisiones informadas. La investigación sugirió que la arquitectura de microservicios era preferible para la disponibilidad, despliegue y el manejo de la complejidad a medida que el sistema crece, aunque presentaba desafíos en las pruebas de integración. El modelo presentado proveyó un enfoque de ingeniería de software guiado por calidad, permitiendo a los equipos justificar su elección arquitectónica con datos.

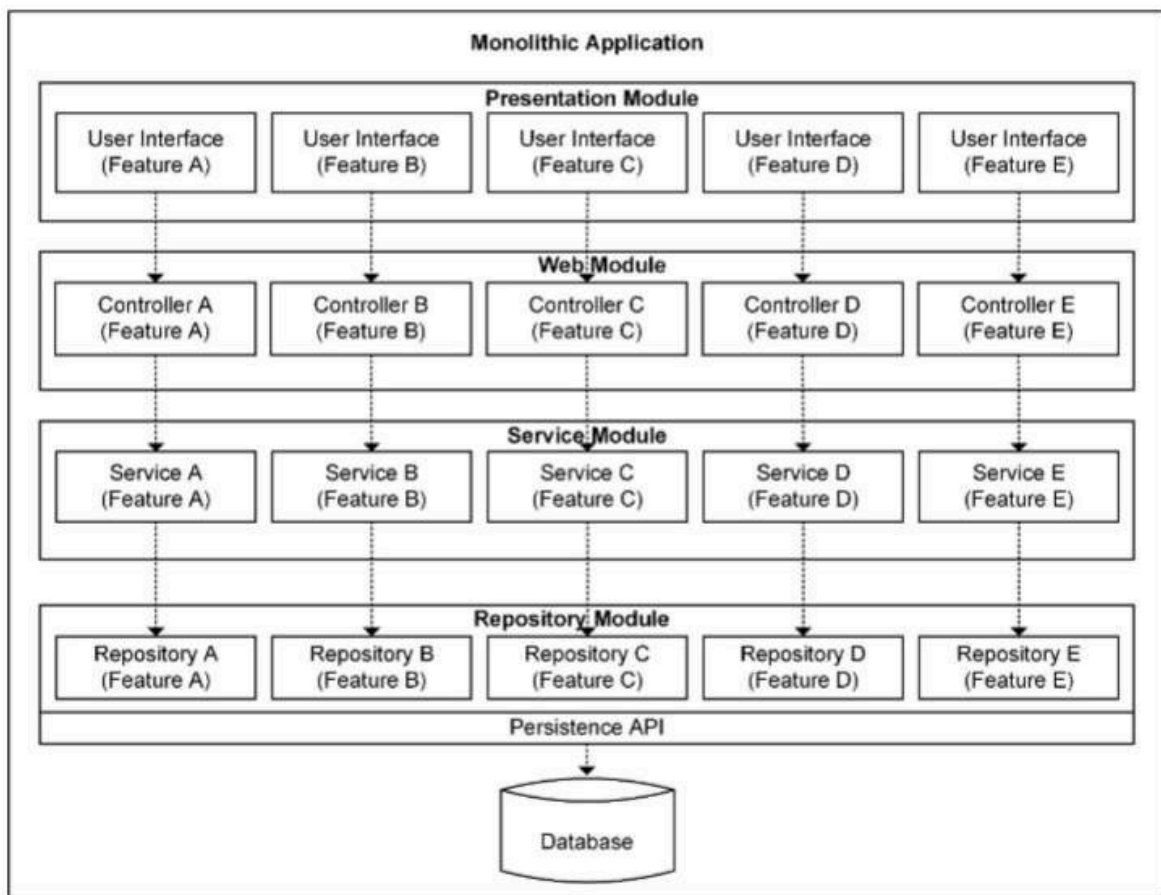


Figura 7 Arquitectura monolítica (Milić, M., & Makajić-Nikolić, D. 2022)

De acuerdo con lo expuesto por (Naimi et al., 2025) existió un obstáculo para las aplicaciones turísticas, debido a la diversidad de plataformas y dispositivos, la complejidad de integrar múltiples fuentes de datos y servicio; así también, el rápido cambio tecnológico y comportamiento de usuarios, todo ello bajo restricciones de tiempo, recursos, requisitos de escalabilidad y mantenibilidad. El objetivo fue proponer una metodología basada en Model-Driven Architecture (MDA), apoyada en el marco de dominio 6As que permitiera acelerar el desarrollo de aplicaciones turísticas, reduciendo el esfuerzo manual y que garantizara la calidad, escalabilidad y adaptabilidad por medio de transformaciones automáticas de modelo a código. El marco conceptual 6As, indicó *Attractions*, *Accessibility*, *Amenities*, *Activities*, *Ancillary Services* y *Available Packages* como modelos de dominio, lo cual permitió capturar de forma sistemática los puntos claves del ecosistema turístico.

La metodología adoptada siguió las tres capas de MDA: *Computation Independent Model* (CIM), *Platform Independent Model* (PIM) y *Platform Specific Model* (PSM), como se mostró en la Figura 8. La capa CIM se abordó con la construcción en UML con paquetes

alineados al 6As, y se refinó manualmente para obtener el PIM, posteriormente, se parametrizó con perfiles UML y plantillas para transformación (PDMs). El caso de estudio implementó un backend en .NET siguiendo principios de Clean Architecture, con separación de capas (Core, Infrastructure, Presentation) y generación automática de entidades, repositorios y servicios mediante transformaciones M2T (model-to-text) en XSLT. En la validación, se realizó la comparación del desarrollo manual con el automatizado, donde se mostró una reducción drástica del tiempo total (pasó de un promedio de 7h 28m a 4m 42s) y una alta fidelidad estructural del código generado (89–97% de coincidencia con el manual). El enfoque propuesto, demostró que la combinación de MDA, un modelo de dominio robusto y un catálogo de PDMs permitió el desarrollo de aplicaciones turísticas escalables y mantenibles en tiempos muy reducidos.

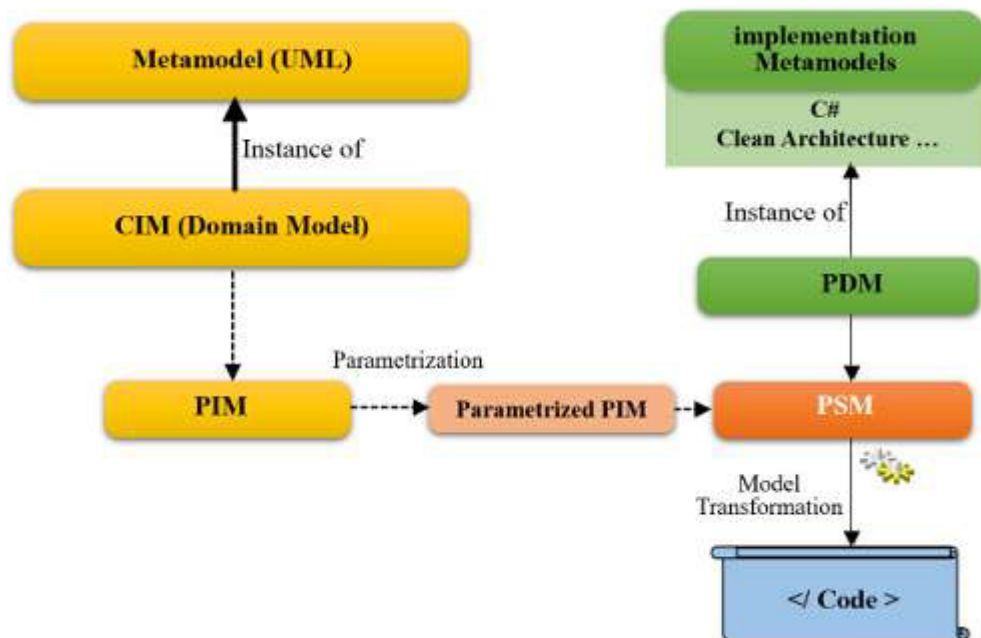


Figura 8 Enfoque propuesto para el desarrollo de sistemas turísticos (Naimi, et al., 2025)

El desarrollo web tenía retos como la obsolescencia tecnológica y la necesidad de interoperabilidad. Debido a esto, (Tesoriero et al., 2022) propusieron una arquitectura de transformación de modelos para la generación automática de aplicaciones web multicapa, en el que buscaron resolver la falta de soluciones que genero de forma integral las capas de persistencia, servicio y presentación; y asegurar el cumplimiento de estándares e interoperabilidad en las fases de desarrollo. El objetivo principal que presenta el estudio es

automatizar la creación de aplicaciones web completas a partir de un único modelo de clase UML independiente de la plataforma (PIM). Para lograrlo, los autores implementaron una metodología basada en la Arquitectura Dirigida por Modelos (MDA) de OMG, utilizando transformaciones M2M (ATL) y M2T (ACCELEO); las transformaciones permitieron generar el código fuente para las capas de la aplicación y casos de prueba, en la Figura 9 se mostró el modelo de la transformación.

Los resultados demostraron la capacidad de la arquitectura para generar aplicaciones web funcionales de tres capas (Persistencia con MySQL y Propel, Servicios RESTful con JSON y PHP, y Presentación con Polymer, HTML y JavaScript) a partir del PIM. La aproximación generada, a diferencia de otras propuestas, cubrió la totalidad de la arquitectura de la aplicación. A pesar de que la propuesta era robusta, se identificaron limitaciones como el soporte restringido a MySQL, servicios REST con JSON y Polymer, lo que abrió futuras líneas de investigación para incorporar diversas tecnologías en las distintas capas de la aplicación.

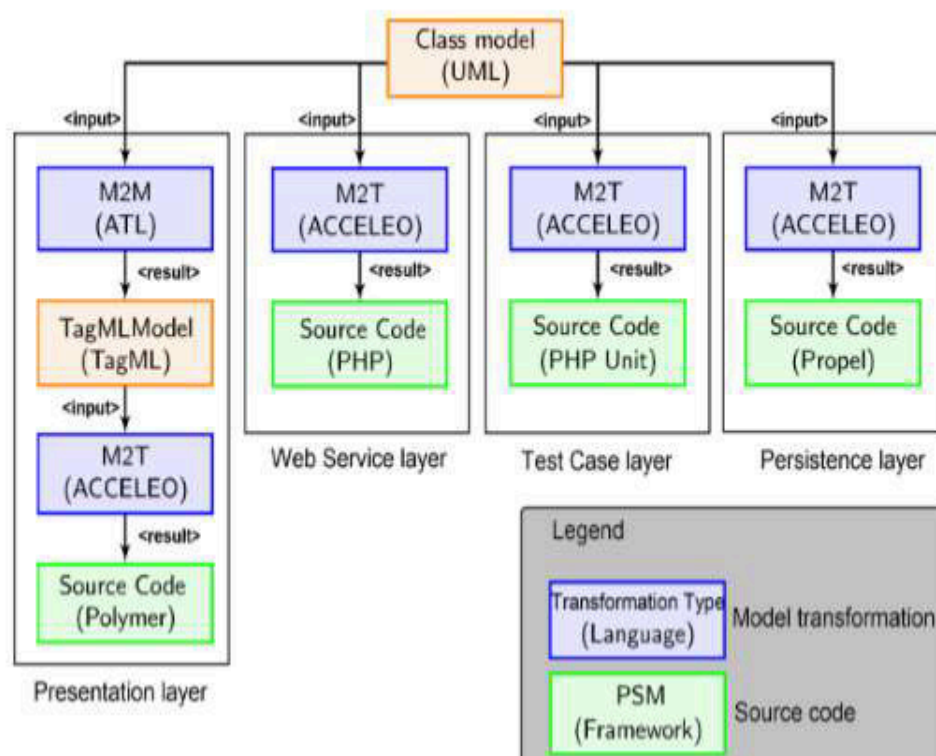


Figura 9 Arquitectura Multi Layered (Tesoriero et al., 2022)

(Aguilar y Figueira, 2020) señalaron que garantizó la mantenibilidad del software, fue necesario adoptar patrones de arquitectura que favoreció la comprensibilidad del sistema. En

su estudio, desarrollaron Kamui, un sistema en Python que actuó como envoltorio sobre KSQL, implementado siguiendo los principios SOLID, en especial el de Responsabilidad Única (SRP). Esto fue de modo que cada clase se ejecutó como instancia invocable y organizó en torno a casos de uso propios de la Clean Architecture, como se indicó en la Figura 10. La estructura aplicada permitió aplicar de forma práctica el enfoque de mantenibilidad y demostró una gran flexibilidad durante el proyecto; en la migración de una interfaz HTML con CSS hacia una Single Page Application (SPA), no requirió modificar la capa de casos de uso, gracias a que estaba desacoplada de otras entidades. Aunque la lógica de negocio era limitada, el sistema integraba múltiples proveedores de datos, algunos con interacciones complejas, cuya gestión se simplificó en un entorno escalable. Los autores llegaron a la conclusión de que, incluso en las aplicaciones con reglas de negocio poco robustas, la adopción de la Clean Architecture podía aportar beneficios significativos, especialmente en escenarios con diversidad de fuentes de datos.

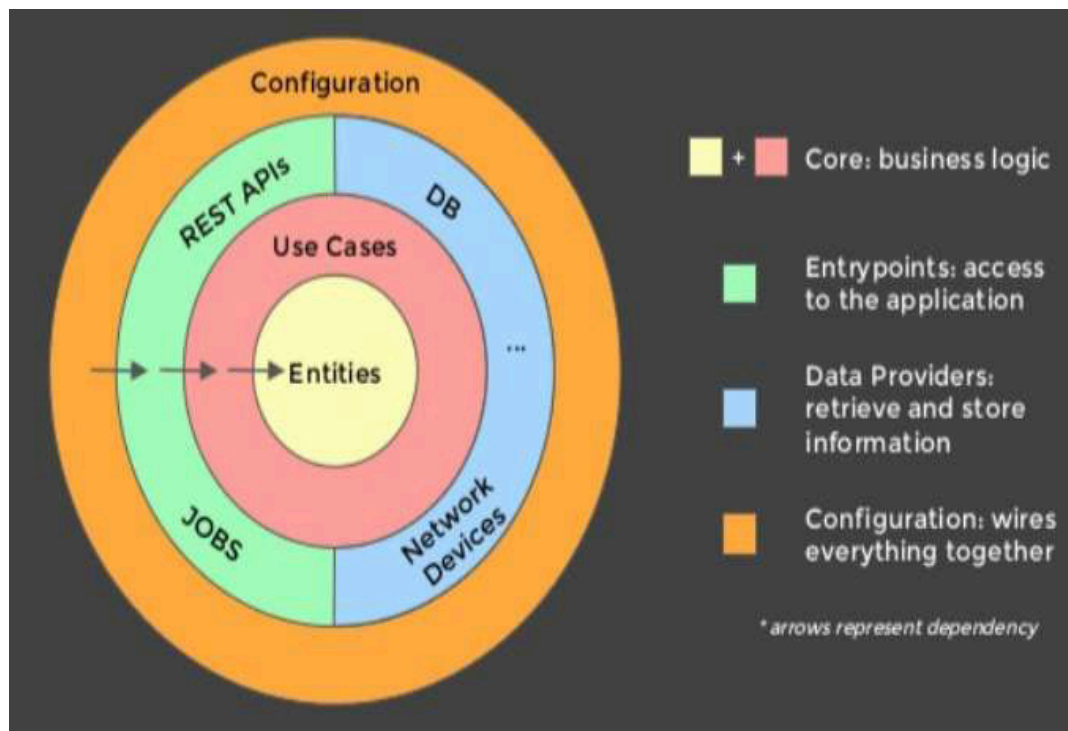


Figura 10 Clean Architecture basado para servicios internos (Aguilar & Figueira, 2020)

3.4.2. Framework para desarrollar un sistema web

De acuerdo con los autores (Swacha et al., 2023), existió una dificultad para desarrolladores e investigadores para medir la popularidad y evolución de los frameworks usados para el desarrollo web a lo largo del tiempo. Conocer esa información conllevó a una mejor toma de decisiones al iniciar nuevos proyectos. Por lo cual, tener acceso a las métricas fue indispensable para la toma de decisiones más informadas, lo que podía influir en aspectos como la disponibilidad de recursos, el soporte de la comunidad, la facilidad de encontrar ayuda y la relevancia a largo plazo de la tecnología elegida. El objetivo del estudio fue analizar la popularidad de los frameworks web más usados en los últimos 15 años usando datos de diferentes fuentes, como GitHub y Stack Overflow, para ver si se relacionaban entre sí. Así también, se buscó comparar esta popularidad con la de sus respectivos lenguajes de programación. En la Figura 11, se indicó en un diagrama la comparativa de repositorios creados con diferentes tecnologías durante el 2008 - 2022.

La metodología consistió en un estudio comparativo y exploratorio que usó un análisis visual y estadístico con datos desde 2008 hasta 2022 de GitHub (repositorios), Stack Overflow (preguntas) y el índice TIOBE (lenguajes de programación), con ayuda de programas en Python para automatizar el proceso. Para analizar los datos, se usó la correlación cruzada y la prueba de causalidad de Granger para entender la relación entre las fuentes y si una podía predecir a la otra. Por otro lado, también se realizó un análisis comparativo de las características de los frameworks a partir de otros estudios.

Los resultados mostraron que la popularidad de los frameworks fue un fenómeno dinámico y que no siempre se manifestó de manera consistente en todas las plataformas. Los datos de las plataformas evaluadas a menudo reflejaron tendencias opuestas, pero se encontró una correlación significativa para la mayoría de frameworks evaluados. La prueba de causalidad de Granger demostró que, en gran parte de los casos, la popularidad en una plataforma podía predecir la de la otra. Se concluyó que el uso de múltiples fuentes fue crucial para obtener una imagen completa de la popularidad de los frameworks y que la correlación entre plataformas, aunque existía, no fue uniforme y dependía del framework específico.

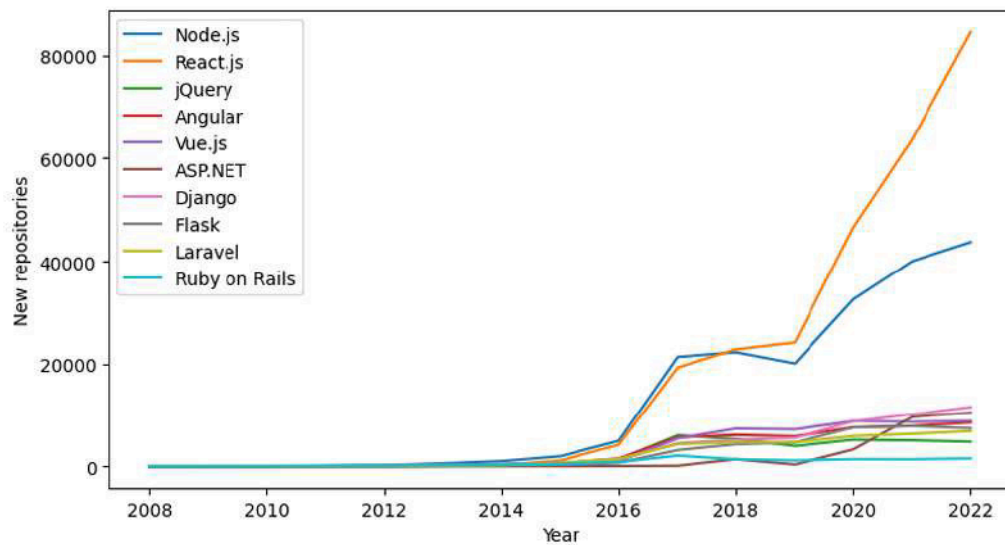


Figura 11 Número de repositorios creados en GitHub por framework (Swacha et al., 2023)

El desarrollo de software moderno demandó el uso de grandes volúmenes de datos y tecnologías web modernas. En base a este contexto, el estudio de (Avilés Matute et al., 2020) presentó un sistema web diseñado para mejorar la gestión por procesos en la compañía de seguridad UNICEPRI (el cual se mostró en la Figura 12). Su propuesta se centró en la integración de frameworks como Laravel para el backend y Vue.js para el frontend, utilizando MariaDB como gestor de base de datos. El problema principal que abordó el estudio fue la ineficiencia de los sistemas existentes en la compañía UNICEPRI para la gestión de procesos críticos como el personal de seguridad y el inventario, que operaban con un sistema básico desarrollado en Visual Basic. El objetivo de los autores fue desarrollar una aplicación web que optimice la gestión de estos procesos, mejorando la transferencia de datos y la experiencia del usuario.

La metodología adoptada fue SCRUM, una metodología ágil que permitió un desarrollo incremental y una constante interacción con el cliente. Las fases incluyeron la planificación, desarrollo y finalización, donde la planificación usó el método de T-shirt y el desarrollo, la arquitectura Modelo-Vista-Controlador (MVC). La evaluación de calidad se realizó mediante la métrica de eficiencia en tiempo de respuesta de la norma ISO/IEC 25010, analizada con una prueba estadística t-pareada. Los resultados demostraron una eficiencia significativa del sistema, que optimizó la gestión de los procesos administrativos de UNICEPRI. La implementación de la arquitectura MVC y la reutilización de componentes mejoraron la

eficiencia de los recursos y la recuperación de datos. La integración de Laravel y Vue.js resultó ser una solución eficaz y estandarizada para el desarrollo de sistemas de gestión por procesos, mostrando un modelo exitoso para empresas que buscaban eficiencia y automatización, aunque se reconoció la necesidad de explorar la integración de otras tecnologías en futuros trabajos.

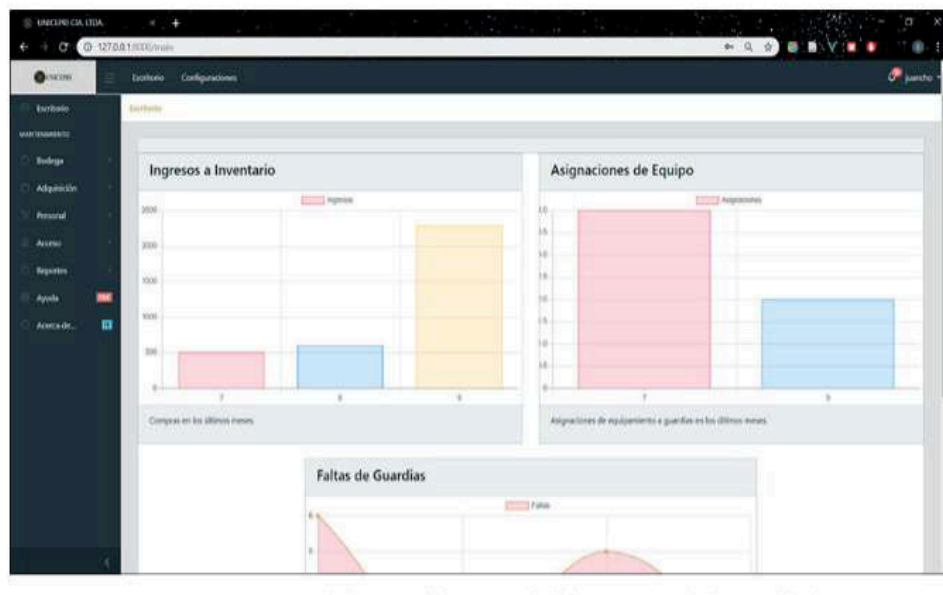


Figura 12 Sistema web desarrollado con Laravel y Vue.js (Avilés Matute, S., et .al, 2020)

El autor (Rathinam, 2022) abordó la problemática de la elección del framework para el aumento de la eficiencia y rendimiento, presentando una solución arquitectónica que analizó y comparó tres de los frameworks más populares: React, Angular y Vue.js. Entre ellos, se mencionó y destacó el respaldo de Facebook a React, y las capacidades de la librería para crear interfaces interactivas y velocidad alta de desarrollo en caliente. Esto se atribuyó al uso de un Virtual DOM (VDOM) en lugar del Real DOM, lo que agilizó las actualizaciones y, por ende, el proceso de desarrollo. Además, React se benefició de una amplia gama de extensiones de navegador que mejoraron la productividad y fomentaron la colaboración, en la Figura 13, se observó la principal diferencia. A diferencia de Angular, que fue desarrollado por Google, se describió como un framework con un código bien segmentado, pero que podía resultar más lento en la actualización de la interfaz al depender del Real DOM, lo que podría generar demoras en el flujo de trabajo de los desarrolladores. Por otra parte, Vue.js, basado en

la arquitectura MVVM (a diferencia de Angular que usa MVC), fue reconocido por su ligera curva de aprendizaje y su creciente popularidad, a pesar de no contar con el mismo nivel de soporte corporativo que React o Angular. Vue.js también incorpora el VDOM, lo que contribuyó a un renderizado eficiente.

Tras el análisis el autor concluyó que React gozó de una gran preferencia en la comunidad de desarrolladores. Sin embargo, para aquellos que buscaban facilidad de aprendizaje e inicio rápido, tanto React como Vue.js fueron opciones viables. Si la prioridad era un soporte de calidad y una visión a largo plazo, React y Angular se consideraron opciones más comprensibles debido a su respaldo por grandes corporaciones.

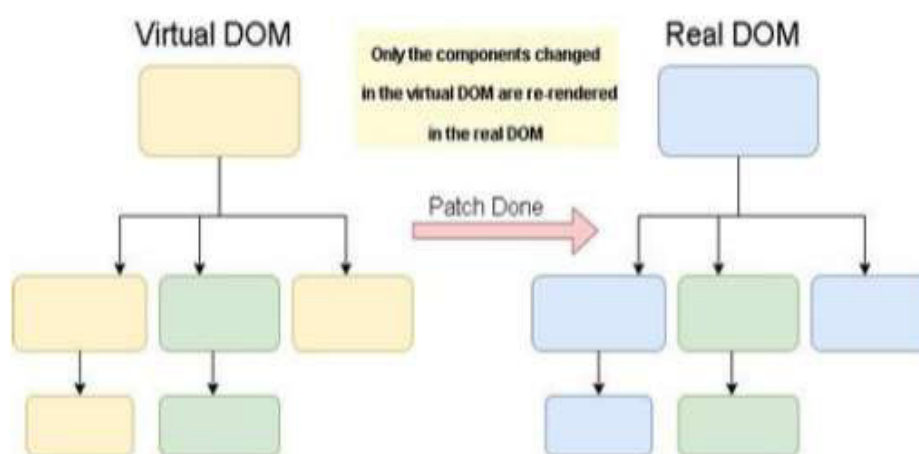


Figura 13 Diferencia entre el VDOM y el Real DOM (Rathinam, S., 2022)

Los autores (Saundariya et al., 2021) abordaron el problema de la ineficiencia y la dificultad asociadas con la búsqueda de profesionales de servicios de hogar, para casos tradicionales, donde la comunicación con el profesional generaba demoras o incrementos de costos. La propuesta de los autores fue el desarrollo de una plataforma web amigable que conectaba usuarios y trabajadores de manera eficiente, brindaba oportunidades laborales a los profesionales y garantizaba un servicio confiable a los clientes, como se observó en la Figura 14, la sección de ingreso para profesionales. El sistema contó con dos tipos de inicio de sesión, para usuarios y trabajadores, y existió un proceso en el que los administradores debieron aprobar a los profesionales que usarían el sitio.

La metodología se centró en la implementación de una aplicación web utilizando la pila MERN (MongoDB, Express.js, React.js, Node.js). React.js se empleó para el frontend debido a su velocidad y optimización SEO, mientras que MongoDB se utilizó como base de datos por su flexibilidad y facilidad de escalado. Los resultados de la implementación de esta aplicación web demostraron que los usuarios pudieron reservar servicios de profesionales con facilidad, sin demoras ni dificultades. El sistema propuesto no solo mejoró la conveniencia para los usuarios, sino que también normalizó y optimizó el acceso a oportunidades para los trabajadores del sector informal. El estudio propuso futuras mejoras como la inclusión de un sistema de seguimiento para mayor comodidad del usuario.

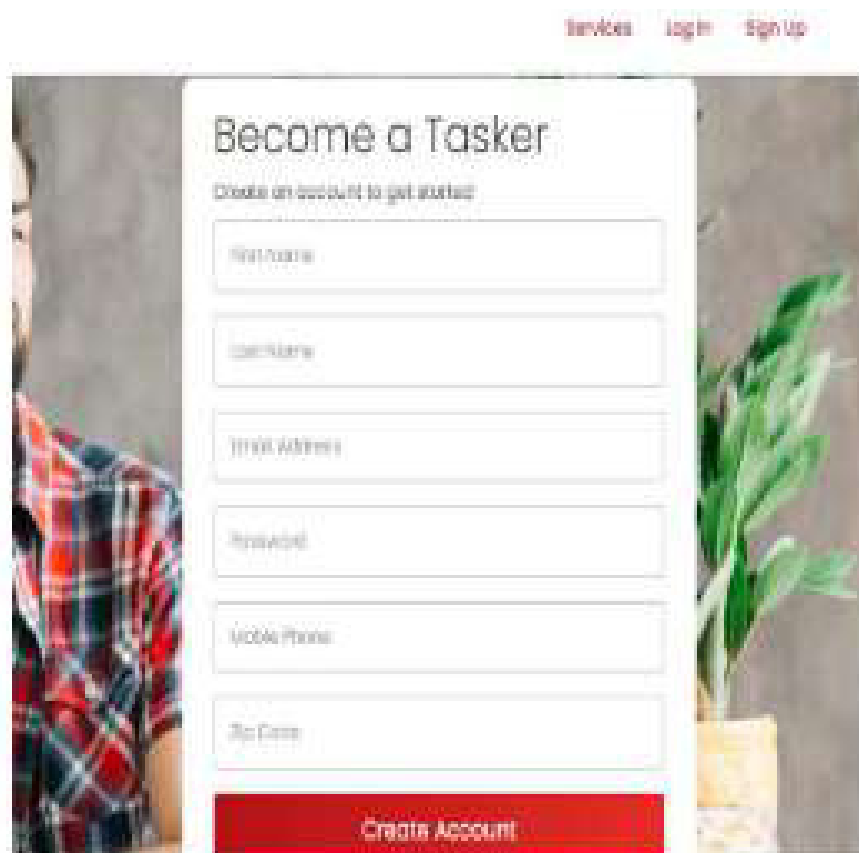


Figura 14 Sistema creado con la librería de React JS. (Saundariya, K., et. al. ,2021)

Según el estudio realizado por (Lei et al., 2022), se abordó el problema de la necesidad de universalidad en los sistemas web para servir a usuarios con múltiples orígenes, lo cual implicó eliminar las barreras del idioma. Específicamente en el contexto de laboratorios remotos y virtuales para la educación en ingeniería, la capacidad de ofrecer una plataforma multi-idioma fue crucial para la adopción a nivel mundial. El principal objetivo del estudio fue introducir y detallar un sistema web multi-idioma para laboratorios remotos y virtuales

basado en React, con la finalidad de transformar un sistema nacional en una plataforma internacional.

La metodología se centró en una propuesta de arquitectura de sistema de lenguaje implementada sobre un sistema ya existente, llamado Networked Control System Laboratory (NCSLab), el cual tuvo un rediseño usando React para mejorar la experiencia del usuario (la arquitectura del sistema se apreció en la Figura 15). La implementación realizada del sistema de lenguaje se basó en una herramienta de código abierto llamada “react-intl-universal”. Por otra parte, se realizó una exploración en detalle del diseño e implementación del sistema desde la perspectiva de su arquitectura separada por frontend y backend, donde se resaltó la facilidad brindada por la implementación realizada para agregar nuevos idiomas y componentes.

Los resultados demostraron que la integración del sistema multi-idioma propuesto, basado en React, mejoró la universalidad y la escalabilidad de la plataforma NCSLab, permitiendo que no solo los usuarios en China, sino también los de todo el mundo, pudieran utilizarla. El uso de React facilitó un diseño modular que permitió una fácil adición de nuevos idiomas y componentes. Se implementaron con éxito el chino y el inglés como estudio de caso, y el sistema permitió una fácil extensión a otros idiomas como el español y el francés. La contribución de React fue fundamental, ya que el sistema de lenguaje se construyó sobre su arquitectura de componentes y usó herramientas específicas del ecosistema React, lo que simplificó la internacionalización a pesar de la complejidad de la arquitectura de cuatro niveles y la separación de frontend y backend.

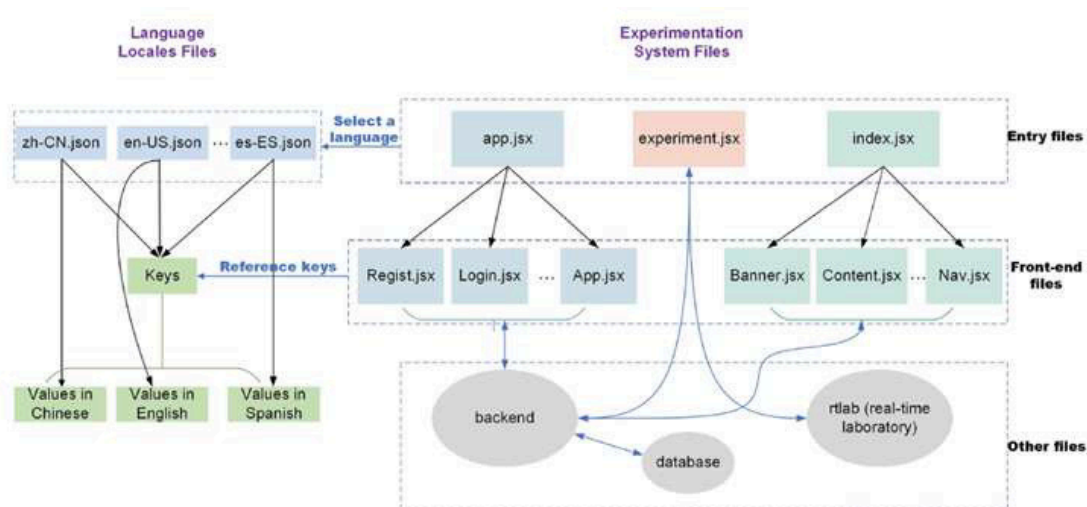


Figura 15 Arquitectura del Sistema de Lenguaje (Lei, Z., et al., 2022).

Según los autores (Ollila et al., 2022), investigaron sobre las estrategias de renderizado adoptadas por frameworks de vanguardia como Angular, React, Vue, Svelte y Blazor, tal como se indicó en la Figura 16. A diferencia de generaciones anteriores, estas herramientas promovieron un desarrollo de aplicaciones declarativas, simplificando la manipulación del DOM y mejorando la interactividad, especialmente en interfaces de usuario con estados complejos. La contribución del estudio, se basó en la comparación de las tácticas de renderizado del DOM, donde se distinguieron fundamentalmente dos aproximaciones. La primera, la comparación de árboles o Virtual DOM (VDOM), es característica de React y Vue, buscando minimizar las diferencias entre el DOM real y una representación virtual. Aunque su complejidad algorítmica era elevada, en la práctica se optimizó para hacer las actualizaciones más eficientes. La segunda categoría, la verificación de enlaces (Dirty Checking), fue empleada por Angular y Svelte. Este método generó conexiones directas para modificar el DOM, y las variaciones en el rendimiento al actualizar el contenido se debieron principalmente a los costos fijos inherentes a cada framework.

Los resultados de la comparación indicaron un rendimiento variable según el escenario. Cuando se trató de renderizar numerosos componentes en un único árbol, React sobresalió en velocidad de ejecución y renderizado, seguido de cerca por Angular. Sin embargo, en un contexto de árbol binario, Blazor se posicionó como líder, superando a React y Svelte. Estos resultados indicaron que React y Angular fueron opciones consolidadas y de alto rendimiento en el panorama actual de los frameworks. No obstante, Blazor emergió como una alternativa con un gran potencial, aunque su menor adopción en la comunidad podría implicar un soporte menos extenso. Por último, los autores resaltaron que las diferencias de rendimiento entre estos frameworks tendieron a ser imperceptibles en plataformas de desarrollo con capacidades robustas. Por lo tanto, la elección del framework idóneo debió alinearse con las condiciones específicas de cada proyecto, lo que consideró los recursos disponibles y los requisitos particulares de rendimiento.

Framework	Components Processed	Elements Processed	Virtual DOM
Angular	All	Bindings only	No
React	Subtree of updated component	All	Yes
Vue	Dirty components only	Bindings only	Yes
Svelte	Dirty components only	Bindings only	No
Blazor	Subtree of updated component	All	Yes

Figura 16 Performance de los diferentes frameworks (Ollila, R., Mäkitalo, N., & Mikkonen, T., 2022).

Según el estudio de los autores (Robayo et al., 2024), la problemática fue la necesidad de la información sobre las características o enfoques de las aplicaciones web que se desarrollaban con el framework Blazor. Resaltó la importancia de la creación de web eficientes y dinámicas, que eran requeridas por una comunidad de desarrolladores que buscaban adoptar tecnologías que mejoró la eficiencia, por lo que el autor indicó que era necesario realizar una investigación profunda sobre el framework. El objetivo del estudio fue proporcionar una visión en conjunto de las aplicaciones de una sola página (SPA) respaldadas por Blazor, destacando sus fortalezas y los puntos clave para el respaldo de decisiones informadas que puedan adoptar los desarrolladores.

La metodología que empleó fue una revisión de trabajos previos relacionados al enfoque SPA sobre Blazor de .NET Core, centrándose principalmente en el análisis de comparaciones sobre rendimiento y características generales de Blazor con otros frameworks basados en JavaScript como React, Ember, Svelte, Angular y Vue (en la Figura 17 se mostraron los tiempos de ejecución por frameworks, tomados en cuenta para la evaluación). Se consideró el uso de Lighthouse de Google para revisar casos de estudio sobre comparativas entre Blazor WebAssembly con una aplicación progresiva de React y Blazor con Angular. Las métricas de rendimiento usadas de Lighthouse fueron First Contentful Paint (Primer Pintado con Contenido) y Time to Interactive (Tiempo para Interactuar).

Los resultados de la revisión indicaron que, en ese momento, a pesar de ser un framework nuevo, poseía una comunidad que le brindaba un soporte sólido y robusto. En los estudios de rendimiento, se encontró que Blazor fue superior en un máximo de dos métricas, pero se mantuvo estable en la mayoría de ellas. En una comparación con ReactJS, Blazor mostró tiempos de generación del DOM virtual (VDOM) más rápidos, mientras que ReactJS demostró ser más consistente en sus mediciones. En el caso de Angular, si bien Blazor fue

superado por Angular en cuatro de las seis métricas evaluadas, este demostró una mayor consistencia y estabilidad en sus resultados. La conclusión del estudio indicó que Blazor podía llegar a ser una opción viable en cuanto al desarrollo con C#.

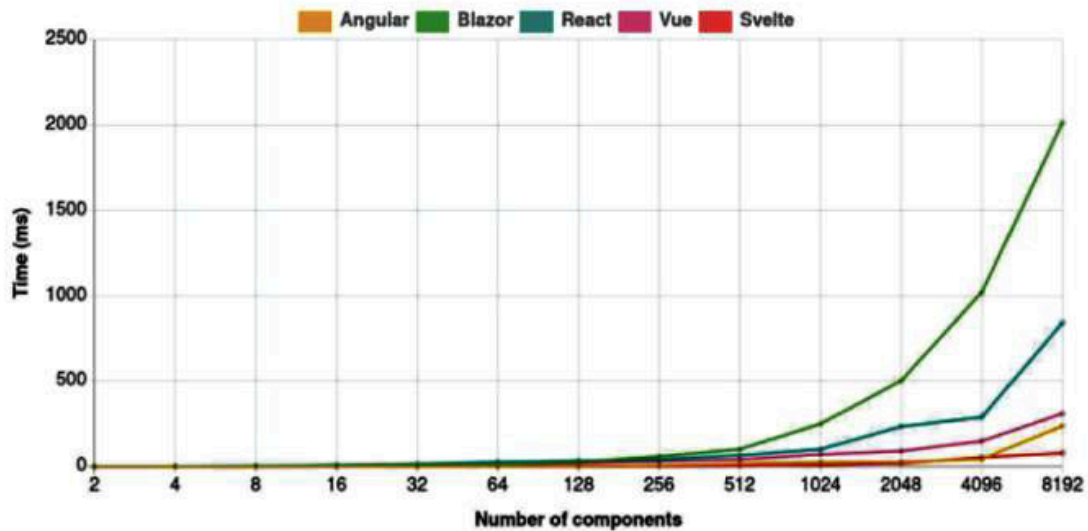


Figura 17 Gráfico de tiempos de ejecución al actualizar los componentes en un árbol de X componentes (Robayo, J., et al., 2024)

De acuerdo con los autores (Rashid et al., 2024), la principal problemática fue la falta de investigación exhaustiva que evaluara el rendimiento, adopción y desafíos de frameworks de forma integral. Debido a la gran cantidad de opciones disponibles, existió una dificultad para empresas y desarrolladores para elegir el framework más adecuado para sus necesidades, incluyendo aspectos como la escalabilidad, eficiencia en costos y seguridad. El objetivo planteado fue analizar las prácticas de escalabilidad, costo y seguridad de los frameworks más usados e identificar los desafíos de su implementación, así también, ofrecer recomendaciones para el uso de frameworks dependiendo de los requisitos de diferentes industrias.

La metodología empleada fue un enfoque mixto, que combinó el análisis cuantitativo de datos de la industria con conocimientos cualitativos de profesionales TI de la industria. Para llevar a cabo el estudio, se recuperó un conjunto de datos de 10 frameworks web de 10 dominios, donde se utilizaron bibliotecas de Python como Pandas, Matplotlib y Seaborn. Se utilizaron técnicas de representación visual como gráficos de dispersión y regresión para analizar la relación entre métricas clave, entre ellas, el tiempo de implementación, costo, tasa de adopción y cuota de mercado; con ello, se buscó comparar el rendimiento y fiabilidad de los frameworks. El estudio realizado por los autores indicó que React fue un framework

escalable y rentable, orientado al usuario en base al comercio electrónico; por otro lado, con respecto a frameworks backend, Laravel y Django fueron los más apropiados para aplicaciones complejas para bancos y gobiernos. El estudio indicó los desafíos, problemas de seguridad y dificultad en la integración con sistemas heredados. Finalmente, se concluyó que la elección del framework debía ser específica para cada dominio, tomando en cuenta el costo, escalabilidad y seguridad para asegurar una implementación exitosa logrando una aplicación óptima, tal como se indicó en la Figura 18.

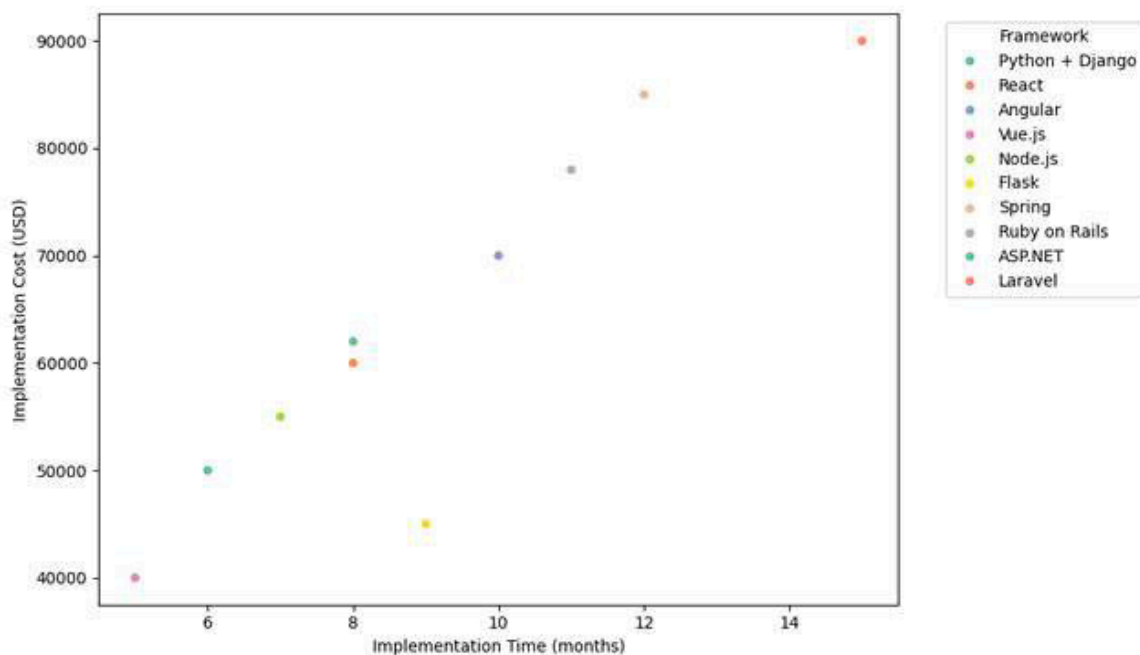


Figura 18 Gráfico de comparativa de frameworks, costo versus tiempo de implementación
(Rashid, R., et al., 2024)

3.4.3. Principios y patrones arquitectónicos

De acuerdo con (García et al., 2021), el desarrollo del ciclo de vida del software depende estrechamente de una buena arquitectura, la cual resultó fundamental para el diseño y evolución de un sistema. Por esta razón, comprender y mantener actualizada la arquitectura fue esencial. Esta fase de análisis constituyó una parte del mantenimiento para los arquitectos. Fue así que se debieron comprender estos pasos: (1) analizar el sistema, (2) recuperar la arquitectura desde su implementación, (3) representar los conocimientos adquiridos. Luego de un análisis experimental en el proyecto, se utilizó la herramienta ARCADE'S smell-detection. Los componentes smell arquitectónicos fueron la dependencia cíclica, la

sobrecarga de enlaces y la sobrecarga de preocupaciones (en la Figura 19 se observaron los smells identificables). En conclusión, con el tiempo la ingeniería de software sufrió un deterioro arquitectónico, resultado de una gran cantidad de técnicas y herramientas para abordar estos problemas. Ello conllevó un desafío abierto que implicó: (1) facilitar la inclusión de conjuntos de datos y (2) creación de microservicios para prototipos. Además, con puntos de las tácticas arquitectónicas, se encontró la dependencia cíclica con 38% de problemas, sobrecarga de enlaces con un 6% y la sobrecarga de preocupación compartiendo un 17 % junto a la sobrecarga de enlaces.

Top Five Tactic-ArchRoots	ARCADE Smell Instance (# Files)									
	Dp. Cycle_1 (# 78)		Link OL_1 (# 33)		Link OL_2 (# 3)		Link OL_3 (# 18)		Link OL_4 (# 1)	
	Single	Accu	Single	Accu	Single	Accu	Single	Accu	Single	Accu
UserGroupInformation	29%	29%	3%	3%	100%	100%	11%	11%	0%	0%
CommonConfigurationKeysPublic	3%	31%	3%	6%	100%	0%	11%	100%	100%	100%
MiniDFSCluster	0%	31%	0%	6%	0%	100%	0%	11%	0%	100%
FileContext	1%	32%	0%	6%	0%	100%	6%	17%	0%	100%
Token	32%	38%	0%	6%	100%	100%	6%	17%	0%	100%
All Tactic-ArchRoots	38%		6%		100%		17%		100%	

Figura 19 Instancias smells (García et al., 2021)

Según (Vale et al., 2022), a medida que los profesionales superaron los diversos desafíos arquitectónicos, surgieron soluciones basadas en la variedad de técnicas y patrones, lo que contribuyó a reducir los riesgos técnicos para los proyectos. Estos patrones fueron importantes herramientas cuyo propósito garantizo un conjunto de atributos de calidad. Los autores sostuvieron un diseño de software basado en microservicios apoyado en el uso de patrones de software. Para el análisis de datos, se empleó una combinación de técnicas manuales y asistidas por ordenador, se utilizó la herramienta NVivo para el análisis cualitativo, así mismo se usaron perfiles de profesionales como también de practicantes en la materia. Durante el análisis se identificaron 14 nuevas ganancias y 6 nuevos dolores, entre ellos fueron Gateway Routing, Strangler, External Configuration Store, Static Content Hosting, Pipes and Filters, Responsibility Segregation, Gateway Offloading, Computer Resource Consolidation, Ambassador, entre otros. Los patrones obtenidos fueron siete para la mantenibilidad. Algunos resultados importantes fueron que el patrón Gateway Aggregation fue visto muy mal por todos los participantes en la evaluación, pues en su aplicación resultó no valer la pena los costos. Por otro lado, el patrón Computer Resource Consolidation fue visto como el mejor, y ningún participante reportó algún dolor dentro de este patrón de diseño de software. Los resultados de la evaluación se presentaron en la Figura 20.

Pattern	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	#
GATEWAY ROUTING	●	●	●	●	●	●	●	●	●	9
STRANGLER	●	●	●	●	●	●	●	×	×	9
EXTERNAL CONFIGURATION STORE	●	●	●	○	●	●	×	●	●	8
STATIC CONTENT HOSTING	●	●	●	○	●	●	●	●	●	8
PIPES AND FILTERS	●	●	●	●	×	●	●	○	●	8
COMMAND QUERY RESPONSIBILITY SEGREGATION	●	●	○	○	○	●	×	×	●	6
GATEWAY OFFLOADING	●	●	○	○	●	●	○	●	○	6
BACKENDS FOR FRONTENDS	●	●	○	○	●	●	○	○	○	5
COMPUTE RESOURCE CONSOLIDATION	●	×	×	○	○	×	○	×	×	5
SIDECAR	●	●	○	○	●	●	○	○	○	5
ANTI-CORRUPTION LAYER	○	●	×	●	○	○	×	●	●	4
AMBASSADOR	●	●	○	○	○	○	○	○	○	3
GATEWAY AGGREGATION	●	●	○	○	●	○	○	○	○	3
LEADER ELECTION	○	●	○	○	●	○	●	×	○	3
Number of patterns adopted by participant	12	14	6	5	12	10	7	7	9	

● The pattern was said to have been applied by the participant;

●× The pattern was said to have been applied, but the participant was not familiar with the given pattern's name;

○ The pattern was said to not have been applied by the participant;

○× The pattern was said to have neither applied nor even known to the participant

○ The pattern was said to have been applied but the participant's description of their usage of the pattern does not match the intent originally documented.

Figura 20 Reporte de Patrones adoptados que fueron reportados por los entrevistados (Vale et al. 2022).

Según (Piyawardhana et al., 2023), la calidad de código en el software fue un aspecto muy importante en la industria de la ingeniería de software; en este sentido, el papel de los patrones de software fue de vital importancia. El objetivo del estudio fue investigar el impacto de estos patrones de diseño de software en la calidad de código. Para ello, se recopilaron datos a partir de una encuesta que distribuyó una muestra de desarrolladores con experiencia en el rubro del software, en dicha encuesta se indagó sobre la implementación de patrones de diseño de software y la evaluación de las métricas de calidad de software. Los hallazgos, obtenidos mediante un análisis cuantitativo, proporcionaron una comprensión entre los patrones de diseño y la calidad de software (en la Figura 21 se mostró la respuesta a la pregunta sobre qué tipo de patrones de diseño usaron los encuestados). La encuesta fue dada por 11 ingenieros, por ende, se tomó que el 100% de los encuestados usaron patrones de software. El patrón Singleton fue usado por un 54.5%; el método fábrica por un 27.3%; el patrón adaptador y el patrón estrategia por un 9.1% cada uno. Todos los encuestados afirmaron que usar patrones de diseño era beneficioso gracias al mejor mantenimiento del código, mayor escalabilidad, menor complejidad y un mejor rendimiento del software, además todos concordaron que el uso de pruebas ayudó a identificar errores y garantizo que el código fuera fiable y mantenible. En conclusión, el impacto de los patrones de diseño de software sugirió un impacto positivo en la calidad del código, mejorando la mantenibilidad, la compresibilidad, la fiabilidad y la reutilización de los sistemas de software; sin embargo, los patrones también fueron afectados por el contexto del sistema y la experiencia de los desarrolladores que los implementan.

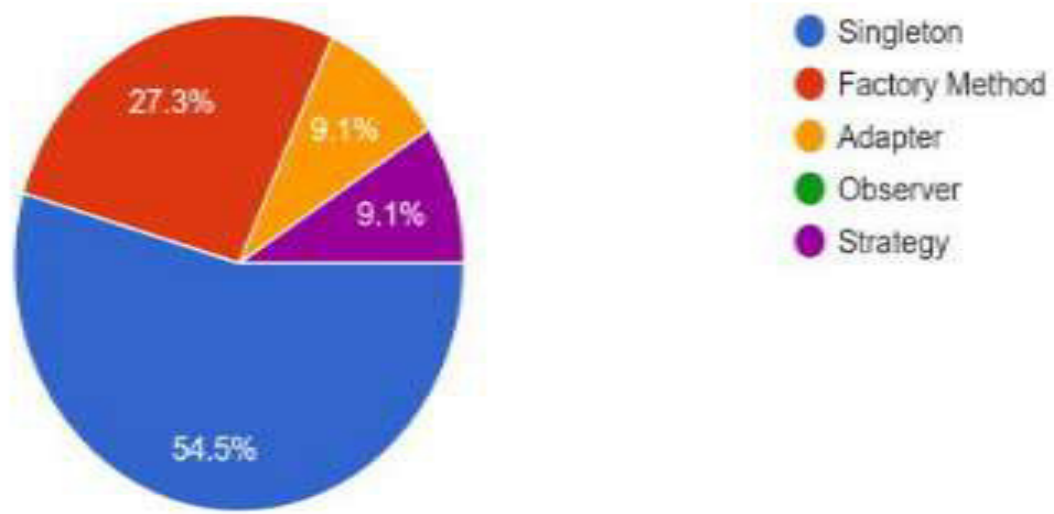


Figura 21 *Diseño de patrones de software (Piyawardhana et al., 2023)*

Acorde con (Jhunjhunwala et al., 2021), en la automatización del software, los algoritmos, la comunicación y los controladores especializados se habían convertido en un elemento cada vez más relevante que garantizó el funcionamiento de sistemas complejos. Por lo general, el acceso público al código fuente de los componentes no estaba disponible. Por ello, la norma IEC 61499 representó una arquitectura moderna basada en componentes y se utilizó frecuentemente para la especificación de aplicaciones de control, dicha norma facilitó el diseño modular, la reutilización de funciones y la interoperabilidad entre sistemas, lo que fue clave para la automatización de sistemas. Los adaptadores simplificaron las conexiones al estructurar y agregar las interconexiones de los componentes. El caso de estudio fue patrocinado por EIT Manufacturing y por el proyecto SWARM cofinanciado por la Comisión Europea: se implementó un modelo de monitoreo en un reactor líquido, el cual empleó sensores para medir parámetros como el nivel de la temperatura. Estos datos fueron procesados por los diferentes controladores que, a su vez, mediante el patrón de diseño para la detección y gestión de conexiones de adaptadores, ofreció un medio para informar errores o excepciones de forma oportuna. Por ejemplo, se verificó el valor antiguo con el nuevo mostrando una máquina de estado en caso de algún error que siguió así en forma de bucle y este fue informado con un código de error. Finalmente, el enfoque propuesto demostró que la combinación de la norma IEC 61499 con patrones de diseño orientados al monitoreo de conexiones fue efectiva para realizar el diagnóstico y la capacidad de respuesta de los sistemas de automatización industrial. En la Figura 22 se mostró el monitor explícito entre las

conexiones del adaptador, observando datos y eventos sin modificarlos y generó alertas de “Monitoring Violation” para la recuperación de errores.

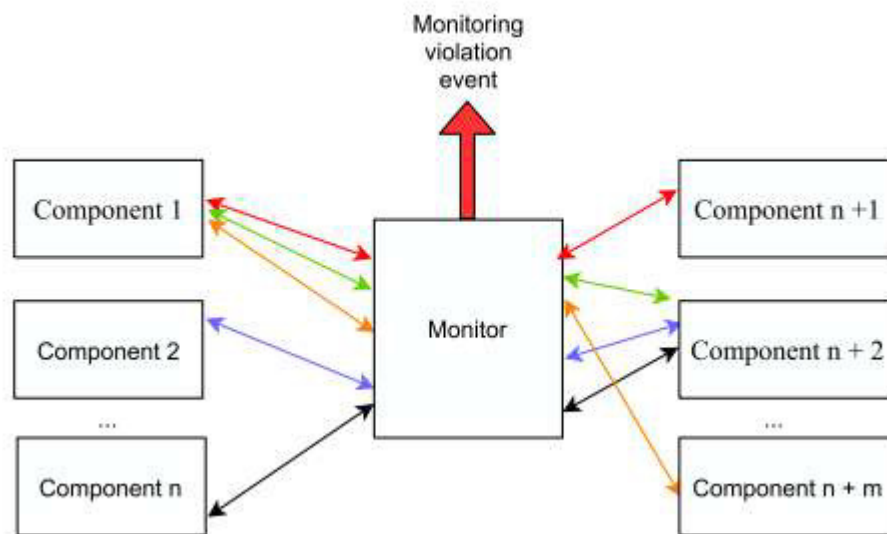


Figura 22 Patrón de diseño genérico basado en IEC 61499 para monitorear interacciones de componentes a nivel de aplicación (Jhunjhunwala et al., 2021)

Según (Houichime & El Amrani, 2025), se propuso un enfoque híbrido para la detección automática de patrones de diseño en el código fuente de los repositorios. Este método combinó el análisis conductual y el análisis estructural, con el objetivo de generar representaciones sensibles al contexto. Dichas representaciones permitieron a los modelos de aprendizaje profundo una mayor precisión en la identificación de los patrones, al capturar no solo la estructura del código sino también su comportamiento dinámico. Para la validación, los autores emplearon unos 23 patrones de diseño de GoF, extraídos de repositorios remotos como GitHub y Bitbucket. Su propuesta alcanzó una exactitud del 92%, demostrando la relevancia de la identificación de patrones de software. En la evaluación, el modelo mostró un desempeño sólido y robusto en la detección de patrones de diseño de software, las métricas obtuvieron valores sobresalientes para el desarrollo como Composite (F1:0.99), Interpreter (F1:0.98) y chain of responsibility (F1:0.97), mostrando una capacidad para detectar estas estructuras, y otros que también fueron sobresalientes como el patrón Singleton con (F1:0.94) y Strategy (F1:0.85) alcanzaron valores ligeramente similares en usar estos patrones, como se observa en la Figura 23. Un aporte fundamental del estudio fue la utilización de grafos de llamadas y propiedades del código, lo que resultó clave para la diferenciación de los patrones de diseño y encontrar irregularidades en el comportamiento,

este enfoque no solo optimizó la clasificación, sino que también surgió posibilidad de un mejor mantenimiento de software y la automatización en procesos de refactorización.

design pattern	Precision	Recall	F1-Score
abstract factory	0.90	0.84	0.87
adapter	0.88	0.82	0.85
bridge	0.87	0.94	0.90
builder	0.98	0.88	0.93
chain of responsibility	0.95	0.98	0.97
command	0.95	0.96	0.95
composite	0.98	1.00	0.99
decorator	0.95	0.96	0.95
facade	0.88	0.92	0.90
factory	0.94	0.95	0.95
flyweight	0.97	0.97	0.97
interpreter	0.97	0.99	0.98
iterator	1.00	0.97	0.98
mediator	0.97	0.97	0.97
memento	0.89	0.98	0.93
observer	0.92	0.93	0.93
prototype	0.92	0.85	0.88
proxy	0.91	0.92	0.91
singleton	0.85	0.82	0.84
state	0.92	0.90	0.91
strategy	0.88	0.81	0.85
template method	0.90	0.98	0.94
visitor	0.92	0.92	0.92

Figura 23 Evaluación de las métricas de Precisión, Recall y F1-Score para cada patrón de diseño (Houichime & El Amrani, 2025)

De acuerdo con los autores (Aktar et al., 2025), la computación cuántica ofreció nuevas posibilidades para abordar problemas complejos con las decisiones arquitectónicas en software. Un estudio empírico desarrollado por los autores analizó 385 issues de 87 proyectos en GitHub y 70 publicaciones en tres sitios de Stack Exchange, con el fin de identificar patrones de software. Entre los más frecuentes destacaron Solution Proposal (35.93%) e Information Giving (32.80%). En cuanto a los tipos de decisiones, predominaron las Implementation Decisions (22.10%) y las Technology Decisions (18.81%). Mientras que en la arquitectura de software tradicional predominaron atributos como rendimiento, escalabilidad y mantenibilidad, en la computación cuántica los patrones buscaron resolver desafíos de diseño, implementación y pruebas. En este contexto, patrones como Visitor, Builder y Recursive resultaron relevantes, ya que permitieron estructurar el software de manera más mantenible y abordar problemas como la gestión del estado, la construcción de circuitos

complejos y la optimización de operaciones. Los resultados evidenciaron que las decisiones arquitectónicas se expresaron mediante patrones de software, destacando como atributo más importante la mantenibilidad. Finalmente, los principales desafíos corresponden a problemas de diseño y altas tasas de error, lo que reforzó la necesidad de contar con enfoques arquitectónicos más sólidos para el desarrollo de software cuántico. Las limitaciones y retos asociados a las decisiones arquitectónicas, junto con sus conteos y porcentajes, se mostraron en la Figura 24.

Decision type	Example	Count	%
Implementation Decision	We designed and implemented an architectural framework that aims to standardize the interaction of an existing infrastructure with IBM Quantum, leveraging the most used open source technologies. (ecosystem issue #266)	141	22.10%
Technology Decision	Also, since the technology of partial measurement/non-demolition measurement is under development (I believe in some quantum architectures), may I expect to use this SDK to change when the technology is mature? (amazon-braket-sdk-python issue #158)	120	18.81%
Architecture Pattern Decision	If we are willing to provide extensibility (which sounds reasonable in a pluggable ecosystem), we should shift to a more suitable pattern such as the observer or pub-sub pattern (...).Again, the pub/sub or observer patterns are examples of decoupled architectures. (Qiskit #825)	87	13.64%
Component Decision	If we want to be strict about the architecture, the log forwarding component should be a controller. (artiq issue #691)	67	10.50%
Tool Decision	Unfortunately this is something coming from Amber build system. It compiles cuda source files for all possible architectures that a given toolkit supports. (QUICK issue #218)	67	10.50%
Data Decision	This seems like part of the discussion about long term architecture, harmonising dc.data with tf.data and TensorGraph layers with Keras layers. (deepchem issue #1036)	54	8.46%
Decomposition Decision	After we have finished the decomposition and local Covalent has a microservice architecture, what should happen when a user does covalent start? (covalent issue #186)	25	3.92%
Integration Decision	I will try integrating quantum layers into existing CNN architectures (QC post #33215)	21	3.29%
Behavior Decision	I would propose not keeping the old behavior, simply to make the refactor easier and avoid code complexity (as well as design decisions). (PennyLane issue #226)	17	2.66%
Design Pattern Decision	Builder Pattern for defining control problems(...).The optimize_pulse API takes in a lot of optional arguments. Wouldn't it make more sense to use a builder pattern to define a control problem in a ControlProblem class? (qutip issue #692)	17	2.66%
Service Decision	The sync method in covalent/results_manager/results_manager.py needs to be moved and updated to be compatible with the new covalent services architecture. (Covalent issue #370)	7	1.10%
Testing Decision	The assert for the third derivatives in test_einset.cpp fails due to what looks like some finite precision issue, but the issue can't be reproduced on other architectures. Unit test for thirdderivative components disabled until this issue is identified. (QMCPACK issue #1218)	7	1.10%
Deployment decision	QPIC is designed to be compatible from Python 2 up to the latest Python 3 (...). Is already fixed in this repo but not deployed to PyPI. Could you update the package in PyPI? (qpic issue #12)	3	0.47%
Containerization decision	I agree that there is a problem (smell) with the design. We are treating the amount of work as related to the size of the container when with padding this is not the case. (qmcpack issue #4290)	3	0.47%
Concurrency decision	The architecture implemented at the moment is showing some issues that are non trivial to fix (...).and then carefully design how we want to do concurrency and parallelism. (Qcodes issue #272)	2	0.31%

Figura 24 Decisiones arquitectónicas en el desarrollo de software cuántico (Aktar et al., 2025)

Según (Liu et al., 2024), el estudio de los anti-patrones había cobrado gran relevancia por su impacto en la calidad y la mantenibilidad de los sistemas de software, en el análisis de gran escala realizado por los autores, se tomaron 1,717 proyectos de código abierto escritos en Java, C/C++ y Python, con el objetivo de evaluar la prevalencia, severidad de los anti-patrones y su relación con los costos de mantenimiento. Los resultados mostraron que el 99.55% de los proyectos analizados presentaron a gran escala los anti-patrones, siendo el más resaltante la Violación de Modularidad, detectada en el 83.54% de los casos y respaldo del 61.55% del esfuerzo del mantenimiento en los proyectos de software, los anti-patrones como

Unstable Interface y *Crossing*, aunque fueron menos frecuente, generaron más costos en el mantenimiento en los proyectos esto a la gran inestabilidad de archivos influyentes en el sistema. El estudio resaltó que el uso de tipado dinámico en proyectos Python incrementó la aparición de anti-patrones y elevó el esfuerzo de mantenimiento, lo que superó costes en proyectos Java y C/C++, se pudo observar en la Figura 25. Los patrones poseían instancias triviales que no representaron un impacto negativo sustancial, en conclusión, los anti-patrones relacionados con dependencias visibles o implícitas resultaron más costosas de ser mantenidos en los proyectos, y la presencia de archivos influyentes pero inestable deben abordarse inicialmente para evitar la acumulación de deuda técnica, estos hallazgos en los proyectos evidenciaron las violaciones a principios de diseño generalizados y representaron un problema crítico en la arquitectura de un software.

Language	CLQ (%)	CRS (%)	MVG (%)	PKC (%)	UIH (%)	UIF (%)	Any (%)
All	82.00	39.00	83.54	84.79	91.62	34.26	99.55
C/C++	49.28	43.12	83.16	93.02	91.17	39.84	98.77
Java	92.46	21.66	73.67	94.06	98.09	19.64	99.68
Python	92.01	48.96	100	29.51	75.69	35.76	100

Figura 25 Porcentaje de proyectos en Python con anti-patrones detectados con posibles dependencias analizadas (Liu et al., 2024)

Tal como lo mencionó (Yanakiev, Lazar, & Capiluppi, 2025), la presencia de deuda técnica en sistemas heredados constituyó una consecuencia casi inevitable tras años de mantenimiento; por ende, las métricas adquirieron un papel fundamental para orientar la priorización de las tareas de mantenimiento que buscaron reducir dicha deuda. En el caso de estudio, una empresa colaboradora de los Países Bajos describió el proceso de refactorización, mitigación y priorización de parte de la deuda técnica arquitectónica acumulada durante el desarrollo. Para ello, se recurrió a una estrategia de priorización basada en datos, esto consideró el impacto que el mantenimiento surgiría en ellas.

La intervención se centró en la aplicación de los principios de diseño SOLID, con especial atención en el Principio de Inversión de Dependencias (DIP). Además, se estableció un repertorio de acciones recurrentes que formalizó de manera explícita los patrones de refactorización, aplicados de manera organizada. En términos cuantitativos, se lograron registrar más de 290 dependencias simbólicas en una base de código que abarcaba más de 5,000 archivos entre fuentes y lenguajes de dominio específico (en la Figura 26 se detalló la estructura del patrón de refactorización e implementación). Los patrones de refactorización

demonstraron ser esenciales para el mantenimiento del proyecto, permitiendo dar consistencia y abordar los problemas de manera sistemática.

Aunque las acciones se ejecutaron únicamente dentro del periodo previsto, los resultados fueron valorados positivamente por la empresa participante y los patrones definidos mostraron ser de gran utilidad en la gestión de los proyectos de gran escala. Así, el enfoque basado en datos y la sistematización de patrones brindaron al equipo una guía consistente para abordar problemas arquitectónicos y realizar una mejor refactorización homogénea en el proyecto.

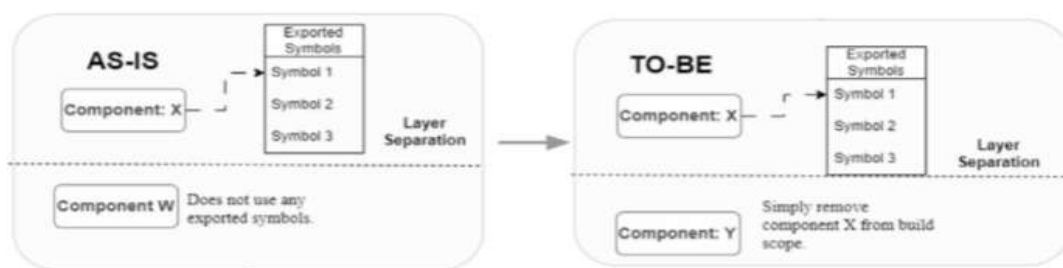


Figura 26 Estructura e implementación del patrón de refactorización: Regla de 3 símbolos exportados (Yanakiev, I., et.al., 2025).

3.4.4. Métricas arquitectónicas

Acorde con (Falco, Scott & Robiolo, 2020), se sostuvo que las empresas de TI de alto rendimiento implementaron software con plazos muy cortos. Con tiempos de lanzamientos tan cortos, fue casi imposible no verse presionado sobre la velocidad de liberación del software, aumentando el número de posibles defectos en producción. Por ende, la creación de este marco basado en el método de evaluación de la calidad del producto (PQEM) estructurada por la norma ISO/IEC 25010, proporcionó características de calidad: Funcionalidad, idoneidad funcional, eficiencia de rendimiento, compatibilidad, uso de capacidad, confiabilidad, seguridad, mantenibilidad y portabilidad; además de la norma ISO/IEC 25023 que definió las medidas de calidad para evaluar cuantitativamente el software.

El método de evaluación PQEM fue un método que constó de 5 pasos cuyo objetivo principal fue analizar, estudiar, medir y evaluar el nivel de calidad de las diferentes iteraciones dentro de un producto de software, y que produjo un valor único entre 0 y 1 como resultado final que representó el nivel de calidad del producto. El método se aplicó en la

segunda iteración de una aplicación llamada HeartCare, integrada en el campo de la salud, asegurando la recuperación física del corazón de los pacientes. El valor de calidad inicial para esta iteración fue de 0.90, y el valor final fue de 0.70. En el paso 1, se definió el valor de calidad deseado (0.90) y el actual (0.70). En el paso 2, la característica de calidad QAR contenía las métricas y criterios de aceptación (entre 1 aprobado y 0 fallido). Para el paso 3, se realizó un análisis para indicar si la pregunta fue implementada o no; solo se encontró un error de compatibilidad y entonces este QAR se estableció como aprobado (esto se realizó para todos los QAR). El paso 4 definió las ecuaciones que se representó la extensión de cobertura, estas fueron disponibilidad, fallo-tolerancia, recuperabilidad, idoneidad funcional, interoperabilidad, capacidad de modificabilidad, rendimiento, eficiencia, seguridad, usabilidad y portabilidad. Por último, se concluyó que la interacción del producto fue aceptada al obtener un mayor nivel de calidad previamente definido, a su vez PQEM como una herramienta semiautomática podría mejorar la adaptabilidad de los desarrolladores en el quehacer diario. En la Figura 27 se observó el resumen de resultados obtenidos a partir de la aplicación de PQEM.

QC	NQAR _{q2}	NpQAR _{q2}	OC _{q2}	EC _{q2}	OvC _{q2}
Availability	12	10	1	0.086	0.086
Fault-Tolerance	5	5	1	0.02	0.02
Recoverability	7	5	0.71	0.03	0.02
Funct. Suitability	59	56	0.95	0.23	0.22
Interoperability	6	4	0.67	0.02	0.02
Modifiability	59	59	1.00	0.23	0.23
Performance Eff.	17	15	0.88	0.07	0.06
Security	19	13	0.68	0.07	0.05
Usability	64	58	0.91	0.25	0.22
Portability	10	8	0.80	0.04	0.03
Total	258	233		1.00	0.90

Figura 27 Resultados de las métricas de la aplicación del PQEM en la segunda iteración en HeartCare (Falco, Scott & Robiolo, 2020)

Según (Mena & Santorum, 2021) que en la actualidad el desarrollo de aplicaciones estuvo en constante crecimiento. Fue así que existieron más de 3.800 millones de celulares inteligentes que fueron posibles usuarios potenciales para estas aplicaciones, lo cual generó un mayor trabajo para los desarrolladores, en consecuencia podría afectar la modificación del producto del software en el tiempo. Se planteó la evaluación de la mantenibilidad y portabilidad de React Native utilizando la norma ISO/IEC 25010 para evaluar las

subcaracterísticas de mantenibilidad y portabilidad; y la ISO/IEC 25022 para las ecuaciones que fueron definidas por los parámetros A y B. El primer paso fue obtener los valores de cada métrica e identificar los valores A y B, luego se procedió a calcular el valor de X según cada métrica, los recursos a usar fueron el código fuente e información proporcionado por el desarrollador y evaluador, por lo que fue necesario un caso de estudio. Además, para mejorar el cálculo se consideró ponderar el resultado en la misma escala de 0 a 1. La metodología aplicada permitió corregir errores y generar buenas prácticas de desarrollo, lo cual resultó en MS1M1 con un valor de 0.5, MS2M1 con un valor de 0.2, MS3M2 con un valor de 0.9, con un valor de MS4M2, con un valor de 0.5, PS1M1 con un valor de 0.9, PS1M2 con un valor de 0.8, PS1M3 con un valor de 0.9, PS2M2 con un valor de 0.67 y PS3M1 con un valor de 0.75, los autores argumentaron que fueron las métricas más importantes para mantenibilidad y portabilidad, en la Figura 28 se mostraron las ecuaciones de métricas de mantenibilidad. En síntesis, de manera indirecta se demostró que usar React Native fue una excelente alternativa para desarrollo multiplataforma en sistemas operativos móviles actuales, además de ayudar a modificar y mantener un entorno de desarrollo durante toda la vida del software.

Metric & equation	Description
MS1M1- Condensation capacity $X = A/B$	A = Number of components that are not affected by changes of other components B = Total number of specific components Where: $B > 0$. $0 \leq X \leq 1$; The closest to 0, is the best
MS1M2- Coupling classes $X = A$	A = Number of relations that a function has respect to other classes. $1 \leq X \leq 4$; The closest to 1, is the best
MS2M1- Execution of reusability $X = A/B$	A = Number of reused elements; B = Total number of items in the reusable library; Where: $B > 0$. $0 \leq X \leq 1$; The closest to 1, is the best
MS3M1- Audit trail capacity $X = A/B$	A = Number of data actually recorded during operation; B = Number of expected data to be recorded to control the state of the system during the operation. $0 \leq X \leq 1$; The closest to 1, is the best
MS3M2- Diagnosis of sufficient functions $X = A/B$	A = Number of diagnostic implemented functions; B = Number of diagnosis functions required in the specification of requirements ($B > 0$). $0 \leq X \leq 1$; The closest to 1, is the best
MS4M1- Cyclomatic complexity $X = A + 1$	A = Number of conditional instructions that a function has [13] $1 \leq X \leq 15$; The closest to 1, is the best
MS4M2- Inheritance depth $X = A$	A = Number of hierarchies used for a given function $0 \leq X \leq 4$; The closest to 0, is the best
MS4M3- Impact correction localization degree $X = A/B$	A = Number of failures appearing after a fault has been solved; B = Number of solved faults. ($B > 0$) $0 \leq X \leq 1$; The closest to 0, is the best
MS4M4- Modification complexity $X = A/T$	A = Number of modifications; B = Work time that takes the developer to modify. ($T > 0$) $X = A/T$ The furthest to 0 is the best
MS4M5- Modification success rate $X = A/B$	A = Number of problems within a certain period before maintenance B = Number of problems in the same period after maintenance ($B > 0$). $0 \leq X \leq 1$; The closest to 0, is the best
MS5M1- Functional complexity of test functions $X = A/B$	A = Number of implemented test functions; B = Number of required test functions. ($B > 0$) $0 \leq X \leq 1$; The closest to 1, is the best
MS5M2- Autonomous testing ability $X = A/B$	A = Number of tests that are depending on of other systems; B = Total number of tests depending of other systems. ($B > 0$) $0 \leq X \leq 1$; The closest to 0, is the best
MS5M3- Test restart ability $X = A/B$	A = Number of cases in which the maintainer can pause and restore the test; B = Number of pause cases in the execution tests. ($B > 0$). $0 \leq X \leq 1$; The closest to 1, is the best

Figura 28 Ecuaciones de métricas de mantenibilidad según la norma ISO/IEC 25022:2016 (como se citó en Mena & Santorum, 2021).

De acuerdo con (Quiña-Mera et al., 2022), fue un desafío diseñar arquitecturas de software altamente escalables y eficientes para el desarrollo de software con una gran calidad. En este sentido, los autores basaron su investigación en el enfoque de Design Science Research (DSR) con una propuesta que buscó evaluar la calidad de un microservidor Spring Cloud. Para ello, se evaluaron las actividades fisioterapéuticas del caso de estudio usando el marco Scrum, el cual constó de definir los requisitos, diseño de la aplicación e implementación y despliegue de la aplicación. En la primera fase se automatizó el registro y supervisión médica. En la segunda fase se diseñó la arquitectura basada en microservicios con la herramienta de Spring Cloud para implementar patrones que proporcionó resiliencia, confiabilidad y la coordinación de los microservicios. En la última fase, los documentos y procesos usados del FDI-E se desplegaron y se registró el sumario de actividades. Esta evaluación mostró una

eficiencia de 83.04% y la satisfacción fue de 78.68%, los parámetros se usaron según la norma ISO/IEC 25022, como se indicó en la Figura 29.

Sub characteristics	Metric	% SC ¹	Scale 1-10	Result	Sum	Scoring level	LS ²
Efficiency	Task time	50%	8,83	44,17%	83.04%	Exceeds requirements	S ³
	Time efficiency	40%	8,72	34,89%		Objective Range	S
	Unnecessary behavior	10%	3,98	3,98%		Minimally acceptable	U ⁴
Satisfaction	Utility	50%	8,50	42,50%	78.68%	Objective Range	S
	Confidence	35%	6,50	22,75%		Objective Range	S
	Comfort	15%	8,95	13,43%		Exceeds requirements	S

Figura 29 Métricas de calidad (Quiña-Mera et al., 2022)

Acorde con (Cai et al., 2025), se mencionó que el modelo de diseño de software pudo tener fallas en su implementación, por ende, fue posible que consumió mucho tiempo para los desarrolladores de software repararlos. Los modelos checkers se usaron para detectar fallas como propiedades violadas o metas inalcanzables. Por ello, se propuso la automatización de la reparación de los modelos de diseño usando un modelo checkers evaluado por la norma ISO/IEC 25010. Los operadores de reparación fueron: inserción que añadía una nueva transición estatal, supresión que elimina una transición estatal de un espacio, modificación una transición estatal que violaba la invariante por la sustitución con otro estado que obedecía. Los modelos reparados fueron evaluados por el modelo basado en la ISO/IEC 25010 lo cual reflejó la idoneidad funcional, las puntuaciones de seguridad, fiabilidad, eficiencia, usabilidad; además la eficacia generada por la máquina reflejó una comparación entre los modelos originales. El experimento hizo uso de ejemplos público ProB, líneas de código (LOC), número de fallas de semillas y descripciones de funcionalidades de modelos en un total de 150 casos de prueba, se encontraron fallas aleatorias en cada uno de los 15 modelos correctos dando como resultado que el proceso propuesto de GPRR y NIVE es capaz de generar reparaciones automatizadas para una amplia gama de modelos. En la Figura 30 se pudo observar el promedio de la calidad de los modelos completamente reparados.

Quality Criterion	Faulty	RSF	BNB	LR	SVM	RF
Functional Suitability						
Total Completeness Loss (%)	11.75	7.68	8.32	8.21	7.10	3.20
Partial Completeness Loss (%)	7.91	1.99	2.47	2.40	1.39	0.62
Total Correctness Loss (%)	11.72	203.11	120.25	112.16	118.72	155.26
Partial Correctness Loss (%)	7.88	198.92	115.55	107.29	113.89	153.25
Total Appropriateness Loss (%)	11.73	7.37	8.01	7.89	5.98	2.96
Partial Appropriateness Loss (%)	7.94	1.92	2.43	2.44	1.33	0.57
Security and Reliability						
Invariant Satisfiability (%)	99.25	100.00	100.00	100.00	100.00	100.00
Availability (%)	10.96	100.00	100.00	100.00	100.00	100.00
Accountability (%)	9.10	8.17	6.50	6.61	6.77	5.21
Fault Tolerance (%)	54.52	58.12	55.74	53.84	54.94	52.20
Recoverability (%)	29.95	22.41	23.80	23.32	24.90	19.20
Maintainability						
Functional Analysability (%)	63.33	76.09	74.14	74.55	73.56	77.74
Fault Analysability (%)	68.98	82.25	81.14	81.04	80.77	81.58
Modularity (%)	65.81	59.48	57.93	56.93	60.04	55.36
Reusability (%)	98.36	98.44	98.43	98.39	98.40	98.33
Performance, Efficiency and Usability						
CPU Time (s)	6.11	18.08	12.57	11.52	15.43	12.91
Peak Memory Usage (MB)	165.52	179.93	177.32	174.59	178.24	177.17
Capacity (10 ³)	7.35	12.86	10.77	10.14	11.19	10.01
Goal Appropriateness (%)	4.58	100.00	100.00	100.00	100.00	100.00
Learnability (%)	92.21	83.43	83.23	84.71	82.31	82.18

Figura 30 Calidad media de los modelos reparados (Cai et al., 2025)

Basado en (Jolak et al., 2025), se sustentó que en los últimos años existió una influencia potencial de los smells arquitectónicos (Architectural Smells). La evidencia empírica vinculó directamente estos olores con los atributos de calidad de la mantenibilidad, como la modularidad y la testabilidad. Estudios demostraron que fue más probable encontrarlos en proyectos de código abierto. Por esta razón, se desarrolló una herramienta para recopilar smells arquitectónicos según la norma ISO/IEC 25010:2023 (como se citó en Jolak et al., 2025). La calidad del software tuvo dos facetas: la calidad en uso y la calidad del producto; los olores arquitectónicos estuvieron relacionados con este último. En este estudio se examinaron varios proyectos de código abierto y se evaluó estadísticamente la relación entre los smells arquitectónicos y los atributos de calidad de la mantenibilidad, específicamente modularidad y probabilidad. La baja modularidad, caracterizada por un alto acoplamiento, baja cohesión o una mala separación de las preocupaciones, podía aumentar el número de cambios innecesarios entre los componentes vecinos. En el caso de estudio, la muestra usó 8 proyectos seleccionados de GitHub sobre la base de los siguientes criterios: ser no trivial y tener un tamaño de al menos 10.000 líneas de código, contener al menos un caso de smell arquitectónico, tener un desarrollo activo durante al menos 3 años y mostrar actividad coherente durante el desarrollo. Además, se limitó a proyectos Java para gestionar la complejidad de las herramientas. El estudio analizó 7 smell arquitectónicos en 378 versiones en 8 proyectos de código abierto del cual se obtuvo *size matters*, los proyectos más grandes

tendieron a tener mayor modularidad; *architectural smells and modularity*, la modularidad y los *smell* arquitectónicos aumentaron a medida que crecieron los proyectos, en la Figura 31 se observaron los *smell* que afectaron negativamente a la modularidad y testabilidad; *dense structure*, tiene un gran impacto en la modularidad potencial del proyecto; *testability and Code coverage*, tuvo una relación débil en la influencia del tamaño del proyecto, *package-level Insights*, nivel de paquetes la mayoría de los *smell* arquitectónicos no se correlacionaron con la cobertura del código lo que el impacto fue más pronunciado a nivel de proyecto que a nivel de paquetes. Por último 3 de cada cinco *smells* asociados al deterioro de la modularidad mostraron relaciones negativas más alto nivel de proyecto y estos fueron interfaz ambigua, dependencia cíclica y densidad estructural.

Architectural Smell	Modularity	Testability	Package Testability
Ambiguous Interface	X		
Cyclic Dependency	X		
Dense Structure	X		
Feature Concentration			
God Component		X	X
Scattered Functionality		X	X
Unstable Dependency			

Figura 31 Smells en las métricas de modularidad y testabilidad (Jolak et al., 2025).

A juicio de (Rahmi Dewi, Ngaliah & Rochimah, 2020), se sostuvo que a comienzo de año la universidad tenía casi 16.000 estudiantes de la ITS que accedían con el FRS (Formulario de Registro de Cursos) para realizar sus cursos. A menudo había un bajo rendimiento del sistema, por lo cual fue vital la calidad del software myITS. El propósito de este estudio fue evaluar la mantenibilidad del módulo FRS usando información que se pudiera medir, como la cantidad de funciones, la cantidad de líneas de código, la complejidad, la cantidad de errores y ensayos utilizados para apoyar la planificación, organización y gestión. Por ello, la investigación se centró en la mantenibilidad y los estándares basados en la norma ISO/IEC 25010 y en sus 5 sub cualidades: en primer lugar la modularidad, que comprendió el acoplamiento y la complejidad; la reusabilidad, midió la capacidad de ser reutilizado en el sistema; analizabilidad, cualidad que diagnosticó las deficiencias y fallas del productos; capacidad de prueba, midió la capacidad de probar de un sistema; modificabilidad, midió hasta qué punto un sistema pudo ser modificado o corregido eficientemente.

El propósito fue obtener el valor de calidad ‘X’ usando las variables ‘A’ y ‘B’ obtenidas de (ISO/IEC 25010:2011, como se citó en Rahmi Dewi, Ngaliah & Rochimah, 2020), en la Figura 32 se indicaron los resultados de las medidas de mantenibilidad por sub-calidad. En este caso de estudio el código fuente del módulo FRS de myITS Lectures y myITS Student fue proporcionado por la dirección de tecnología y desarrollo del sistema de información (FPT SI). Los resultados demostraron que myITS Student obtuvo una puntuación de 2.083 y myITS profesor de 2.670, significó que el módulo FRS en myITS tuvo buena mantenibilidad pero necesitó la sub calidad de reutilización y capacidad de prueba de esta métrica puede mejorar los smells del sistema.

Sub-quality	Point	Point's Code	Tools Code /Calculate Solution	Result (1)	Result (2)
Modularity	Coupling of component conformance	1	A	0.5	0
	Cyclomatic complexity	2	B	1	1
Reusability	Reusability of assets	3	A	0	0
	Conformance to coding rule	4	A	0	1
Analysability	System log completeness conformance	5	C	1	1
	Diagnosis function effectiveness	6	D	0.5	1
	Diagnosis function sufficiency conformance	7	D	0.5	1
Testability	Test function completeness conformance	8	E	1	1
	Autonomous testability	9	F	0	0
	Test restartability	10	E	1	1

Figura 32 Resultados de la medición de mantenimiento en el aplicativo del estudiante y profesor de myITS (Rahmi Dewi, Ngaliah & Rochimah, 2020)

De acuerdo con (Baul Canlas et al., 2021), un indicador para conocer el desempeño de las instituciones académicas fue la productividad del personal, que en consecuencia llevó a realizar una evaluación de calidad en dichas instituciones. El estudio propuso proporcionar una solución usando el seguimiento de las tareas de investigación de los profesores a través de un sistema de gestión que pudiera integrarse a través de programas de productividad, seguimiento y predicción, usando los parámetros de calidad de la norma ISO/IEC 25010. La metodología usada para esta investigación fue en esencia descriptiva y de desarrollo. Se empleó un diseño de investigación cuantitativa tras el desarrollo del sistema y la evaluación del desempeño, usando la técnica KDD (Knowledge Discovery in Databases) como proceso de minería de datos para extraer datos útiles para el sistema que predijo la productividad del docente.

El sistema desarrollado fue evaluado por 15 encuestados, quienes fueron clasificados como expertos con funciones como administradores, profesionales de TI y expertos en investigación. Además, hubo 10 expertos para medir la precisión y coherencia de los ítems. Los criterios fueron la idoneidad funcional, eficiencia, compatibilidad, usabilidad, confiabilidad, seguridad y mantenibilidad. Finalmente, se obtuvo la media general de la evaluación percibida por los expertos, que fue de 3.87 (como se indicó en la Figura 33) , con una interpretación de “Totalmente de acuerdo”, lo que significó que el sistema era confiable. Por otro lado, se sometió también a un análisis ANOVA, que dio un valor de 0.257 como valor significativo, siendo mayor que el alfa de 0.05. Esto demostró que no hubo una diferencia significativa en términos de medias del grupo encuestado. En conclusión, el sistema tuvo las características necesarias para determinar la proactividad de la investigación.

Indicators	Average Mean	Verbal Interpretation
Functional Suitability	3.93	Strongly Agree
Performance Efficiency	3.93	Strongly Agree
Compatibility	3.89	Strongly Agree
Usability	3.84	Strongly Agree
Reliability	3.78	Strongly Agree
Security	3.87	Strongly Agree
Maintainability	3.85	Strongly Agree
Portability	3.89	Strongly Agree
Grand Mean	3.87	Strongly Agree

Figura 33 Performance del sistema (Baul Canlas et al., 2021)

Según (Paliuanan et al., 2022) a medida que los datos se volvieron el nuevo petróleo que impulsó la economía del conocimiento, fue vital tener procesos científicos. Por ello, fue esencial la proliferación de la investigación con la convergencia de la tecnología, entonces generó un mayor impacto en la comunidad. Este estudio estuvo orientado a promover la utilización de nuevos conocimientos a través del uso de una aplicación Web Institutionally-Farmed (InFORMs) cumpliendo el grado de software (ISO/IEC 25010:2011 como se citó en Paliuanan et al., 2022), la metodología propuesta de desarrollo fue SCRUM en el diseño que involucró proceso claves de la vida del software; la cantidad de expertos y profesionales fueron 10 que trabajaron en la industria del TI y también 30 usuarios generales quienes fueron evaluados en la usabilidad, aceptabilidad y satisfacción. El estudio usó un cuestionario estructurado, adicionalmente la herramienta de desarrollo fue XAMPP; el marco usó herramientas de código abierto para su evaluación dando como resultado para la funcionalidad según expertos, 3.72 y según usuario general 4.24; reusabilidad según expertos, 3.78 y según usuario, general 4.40; usabilidad según expertos, 3.78 y según usuario general,

4.37; performance 3.87 según expertos y 4.37 según usuario general; eficiencia con un valor de 3.87 según experto y según usuario general 4.39; mantenibilidad con un valor de 3.92 según expertos y 4.39 según usuarios general; seguridad con un valor de 3.90 según expertos y 4.20 según usuarios general; portabilidad con un valor de 3.48 según expertos y 4.36 según usuario general y por último la compatibilidad con un valor de 3.57 según expertos y 4.33 según usuario general. En resumen, el sistema proporcionó apoyo a la facultad de búsqueda de información científica del sistema InFORMs ya que la evaluación general cumplió según las características de calidad de software ISO 25010, como se observó en la Figura 34.

Criteria	IT Experts (N=10)		Users (N=30)	
	W. Mean	Description	W. Mean	Description
Functional Suitability	3.72	High Extent	4.24	Very High Extent
Reliability	3.78	High Extent	4.40	Very High Extent
Usability	3.82	High Extent	4.37	Very High Extent
Performance Efficiency	3.87	High Extent	4.39	Very High Extent
Maintainability	3.92	High Extent	4.32	Very High Extent
Portability	3.90	High Extent	4.20	High Extent
Security	3.48	High Extent	4.36	Very High Extent
Compatibility	3.57	High Extent	4.33	Very High Extent
Overall Weighted Mean	3.76	High Extent	4.33	Very High Extent

Figura 34 Evaluación sobre el grado de cumplimiento de la ISO 25010 en el sistema (Paliuanan et al., 2022).

Capítulo IV: Metodología e Implementación

En el presente capítulo se proporcionó una descripción detallada de la metodología de investigación utilizada para diseñar una arquitectura con el fin de disminuir la complejidad inicial del software basada en Clean Architecture, la propuesta implica la incorporación de un conjunto de datos de prueba que permitió desarrollar un proyecto web y validar esta variante de arquitectura, el cual constó de cuatro fases: (i) evaluación arquitectónica (ii) diseño de la arquitectura de tres capas (iii) implementación del sistema y (iv) validación de la propuesta, tal como se mostró en la Figura 35.

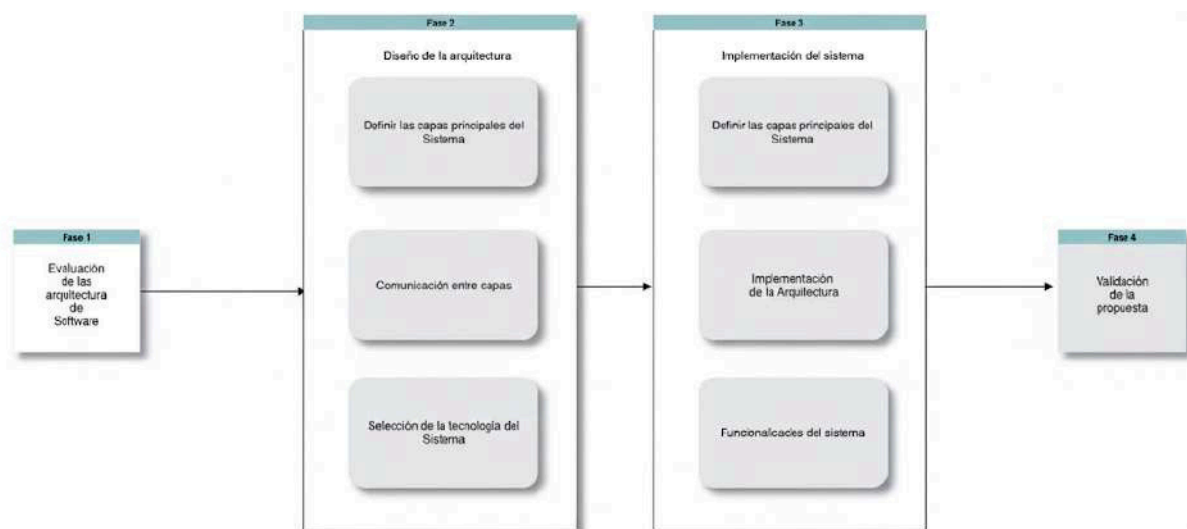


Figura 35 Diagrama de la propuesta arquitectónica

4.1. Evaluación arquitectónica

Luego de un extenso análisis respecto a las arquitecturas, en esta Fase 1 se planteó evaluar los diferentes aportes con el fin de mejorar la Clean Architecture en el impacto que esta sostenía en la mantenibilidad del Software.

La adopción de la Clean Architecture se justificó como el patrón de diseño interno, dado su impacto directo y cuantificable en la mantenibilidad de código (Arias-Orezano et al., 2021; Aguilar y Figueira, 2020). Los resultados empíricos demostraron mejoras frente a arquitecturas tradicionales como el MVC (Arias-Orezano et al., 2021); asimismo, otros autores también la recomendaron porque abordó desafíos de costo y escalabilidad en entornos empresariales (Rojas et al., 2021; Sadek et al., 2022), donde se sugirió en el diseño de software para asegurar la mantenibilidad inherente es sistemas distribuidos.

La mantenibilidad requirió que la aplicación estuviera estrictamente ligada a principios de diseño y composición de patrones, trascendiendo el uso del patrón Adapter, la base de esta estrategia radica en el cumplimiento de los principios SOLID, para mitigar la deuda técnica y la violación de la modularidad, considera el antipatrón más costoso y prevalente en el desarrollo de software (Yanakiev, Lazar, & Capiluppi, 2025; Liu et al., 2024). Para la robustez del sistema ante cambios tecnológicos se justificó la combinación de los patrones Bridge, que desacopló la abstracción de la implementación, y Strategy, que gestiona implementaciones intercambiables (Houichime & El Amrani, 2025; Piyawardhana et al., 2023). Esta composición garantizó una mejora en la adaptación.

La selección de React como librería principal se justificó sobre alternativas como Angular o Blazor debido a su solidez y ventajas en rendimiento, popularidad, facilidad de adopción y soporte (Swacha et al., 2023; Rashid et al., 2024). Aunque frameworks como Angular se presentaron como opciones consolidadas y Blazor emerge con potencial (Ollila et al., 2022; Robayo et al., 2024), React utilizó VDom para la comparación de árboles, lo que se tradujo en una consistencia superior; además, el modelo se alineó con los objetivos de mantenibilidad de la arquitectura basada en Clean Architecture (Avilés Matute et al., 2020).

La elección de una metodología ágil, específicamente SCRUM (Avilés Matute et al., 2020), se estableció como el enfoque de gestión ideal para este proyecto, por su capacidad de manejar complejidad e incertidumbre, contrario al modelo cascada, que fue rígido y penalizó los cambios tardíos, Scrum permitió respuestas a cambios tecnológicos (Naimi et al., 2025). Esto garantizó que la deuda arquitectónica se gestione de forma sistemática (Yanakiev, Lazar, & Capiluppi, 2025; Piyawardhana et al., 2023), mientras que modelos como el Cascada dejaría las pruebas y validaciones en etapas finales.

4.2.Diseño de la Arquitectura de 3 capas

La arquitectura que se planteó se sostuvo en la Clean architecture de (Cecil, R, 2017), la cual comprendió principios y patrones que cumplieron la función de mejorar el proceso de creación del software, desacoplando y definiendo bien las responsabilidades que son detalladas en la Figura 36. Dado lo anterior, se combinó con el trabajo de (Jhunhunwala et al., 2021), que se fundamentó en la norma IEC 61499, una arquitectura moderna basada en componentes utilizada frecuentemente para la especificación de aplicaciones de control.

Esta fase comprende los puntos de: (i) Capas principales, donde se determinó si el proyecto necesitaba una arquitectura de tres a cuatro capas, dependiendo de sus objetivos o

complejidad. (ii) Comunicación entre capas, donde se recalcó la forma en que se interrelacionaron las capas seleccionadas, estableciendo sus responsabilidades y manteniendo claramente sus funciones; y, por último (iii) Selección de la tecnología, donde, aunque esta arquitectura no se sujetó a una sola tecnología, se identificaron frameworks y herramientas que se adaptaron a los diversos problemas planteados en la creación de un sistema.

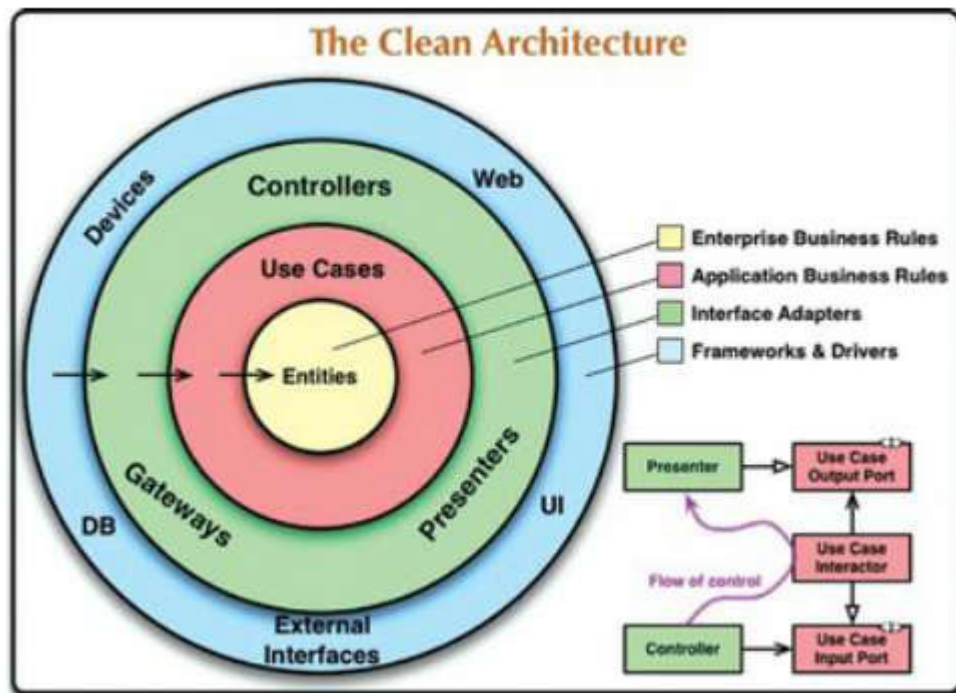


Figura 36 Clean Architecture.(Cecil, R , atl , 2017)

4.2.1. Capas Principales

En la Clean Architecture se contemplaron cuatro capas principales: la primera capa la de presentación, la segunda capa adaptador, la tercera capa de aplicación o capa de uso donde se representaron los servicios y, por último, la capa de entidades. Según (Cecil, R, 2017), se indicó que se debía seguir una regla de dependencia para ingresar entre capas de mayor nivel en los círculos internos y de menor nivel en círculos externos. Nuestra propuesta fue una variante de esta arquitectura que abstraigo la capa adaptadora, la cual usa el patrón de diseño Adapter, por los patrones Bridge y Strategy. El diseño contó con: (1) capa interfaz, la cual representó la capa más externa y más variante, (2) capa de aplicación, que fue la lógica del sistema y la que gestionó las funcionalidades; y, finalmente (3) capa de entidad, que representó la entidades de negocio, tal como se mostró en la Figura 37. Respecto a la capa

adaptador, esta fue extraída por puertos que disminuyen la complejidad que esta representa a medida que crece.

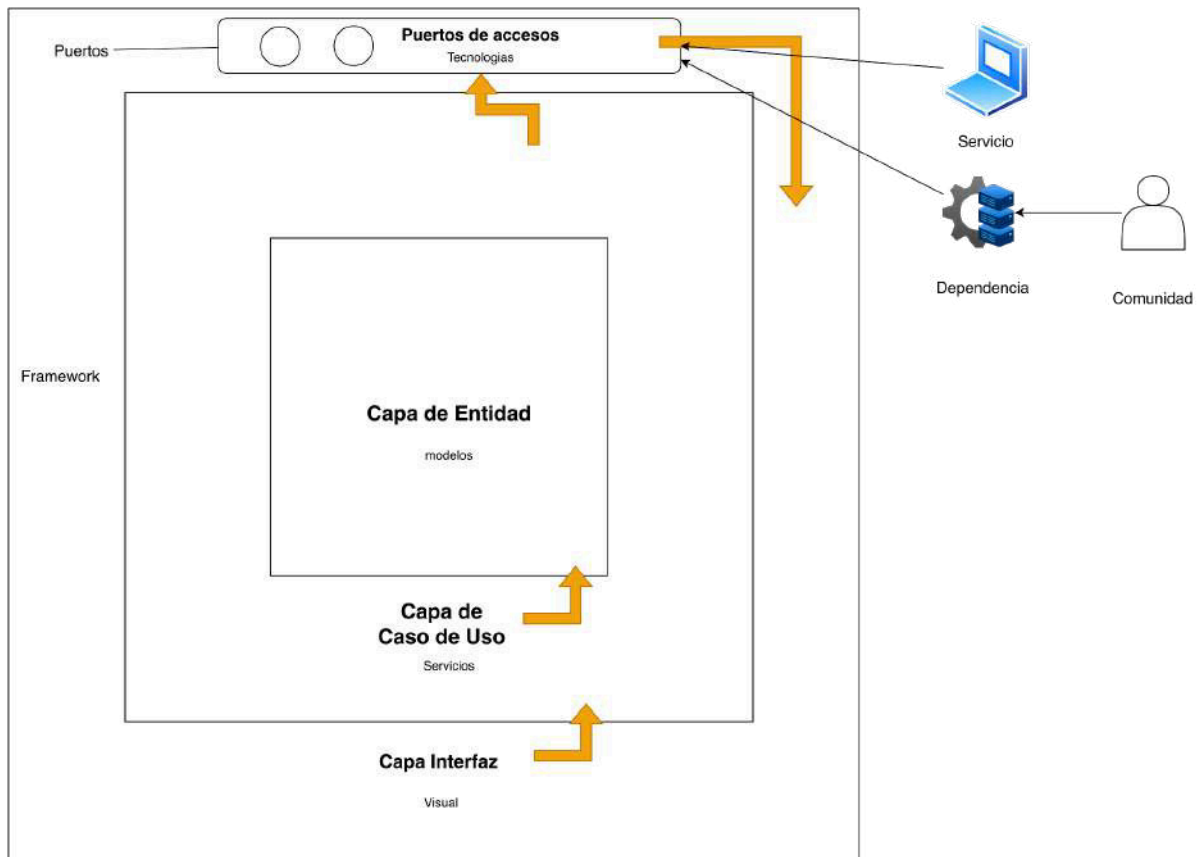


Figura 37 Propuesta de diseño de Clean Architecture

4.2.1.1. Capas Interfaz

Nuestra propuesta se fundamentó en el estudio de (Cecil, R, 2017), quien señaló que la capa presentación es la más variable dentro de la Clean Architecture al estar expuesta a constantes cambios tecnológicos, nuestra capa interfaz comprendió estos principios, así como el trabajo de (Arias-Orezano et al.,2021) que indicó los principios SOLID, asumiendo la responsabilidad de gestionar esa variabilidad, tal como se mostró en la Figura 38. Para afrontar este desafío, se consideró la incorporación de puertos de acceso, tal como fue sugerido por (Jhunjhunwala et al., 2021), con el fin de reducir la complejidad de integrar de manera eficiente distintas herramientas tecnológicas, esta integración se apoyó en el patrón Bridge, que actúa como un puente de enlace para la creación de aplicaciones, sin depender de

una tecnologías específica. Asimismo, se tomó en cuenta el principio de inversión de dependencias, lo que permitió encapsular las tecnologías seleccionadas, ya fueron frameworks, librerías propias o externas y utilizarlas mediante abstracciones que aseguren la independencias de la lógica de alto nivel.

4.2.1.1.1 Puerto de Accesos

La arquitectura clásica descrita en (Cecil, R, 2017), definió con precisión las responsabilidades de cada una de sus capas; no obstante, a medida que el proyecto crece, se produce una sobrecarga de responsabilidades, tal como nos indicaron (Garcia et al., 2021), por esta razón, se introdujo el concepto de puertos de accesos como interfaz abstracta para mitigar la sobrecarga de responsabilidades mediante la sustitución de la capa adaptadora –basada en el patrón Adapter– por los patrones Bridge y Strategy, favoreciendo así una estructura más flexible y mantenible.

El puerto de acceso interno mantuvo la responsabilidad única de abstraer funcionalidades reutilizables, no solo dentro de un proyecto, sino también entre diferentes proyectos que compartieran la misma tecnología base o framework, para ello, se planteó la creación de una librería interna que agrupara dichas funcionalidades exportables, de manera que pudieran ser implementadas en distintos proyectos de la organización. Según (Jhunjhunwala et al., 2021), el acceso público a las dependencias o tecnologías no siempre están disponibles fuera de una organización, por lo que este puerto provee un mecanismo controlado y exclusivo, accesible únicamente para proyectos desarrollados en nuestra organización.

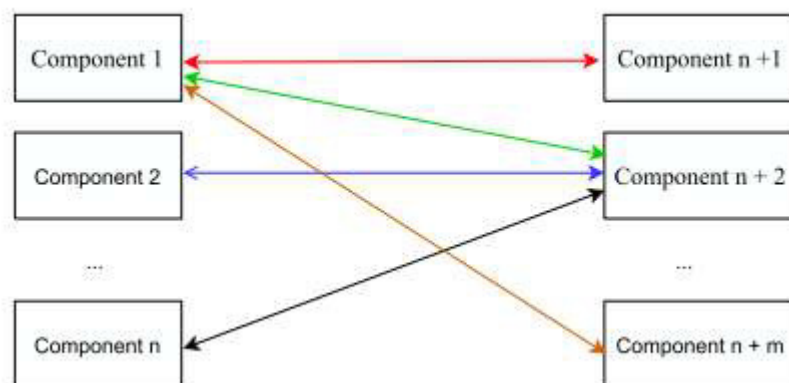


Figura 38 Escenario genérico de interacción de componentes que debe ser monitoreado (Jhunjhunwala et al., 2021).

Por otro lado, el puerto de acceso externo se fundamentó en los principios de automatización del software, en relación con (Jhunjhunwala et al., 2021), los adaptadores permitieron simplificar conexiones con servicios o dependencias; en consecuencia, este puerto se encargó de instanciar y encapsular dichas dependencias, facilitando su modificación o sustitución. El control de estas dependencias se gestionó mediante un repositorio central y herramientas de monitoreo, como GitHub Dependabot, que permiten detectar vulnerabilidades, depreciaciones o versiones obsoletas. De este modo, se garantizó la trazabilidad y el reemplazo oportuno de dependencias externas, reduciendo riesgos asociados a la pérdida de control en la capa adapter. La Figura 39 representó el control de la tecnología mediante un ente organizador.

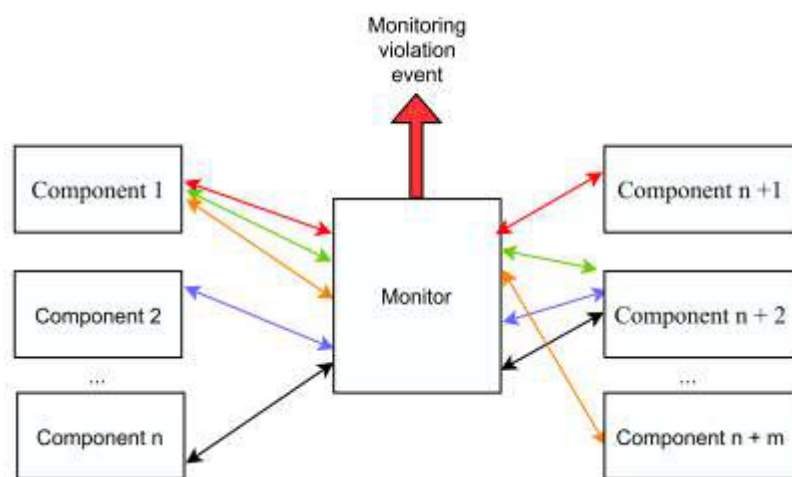


Figura 39 Patrón de diseño genérico basado en IEC 61499 para el monitoreo de interacción de componentes a nivel de aplicación (Jhunjhunwala et al., 2021)

La elección entre utilizar un puerto interno o un puerto externo dependió de las necesidades específicas de las organizaciones o del proyecto acorde con (Vale et al. 2022), fue importante solucionar los nuevos retos arquitectónicos con nuevas ideas. Por ejemplo, en los proyectos, algunas librerías resultaban demasiado pesadas en relación con el uso limitado que se les da, o ciertas funcionalidades internas que solo competen al negocio pueden justificar la implementación de puertos internos. Para este proceso de decisión, fue fundamental aplicar el patrón Strategy, evaluando criterios como accesibilidad, el nivel de uso y el peso de las librerías dentro del proyecto, entre otros factores relevantes, se gestionaron versionados tanto

en puertos internos y externos. La Figura 40 representó como el framework sirve como puente para conectar la parte lógica con la aplicación abstrauyendo la tecnología.

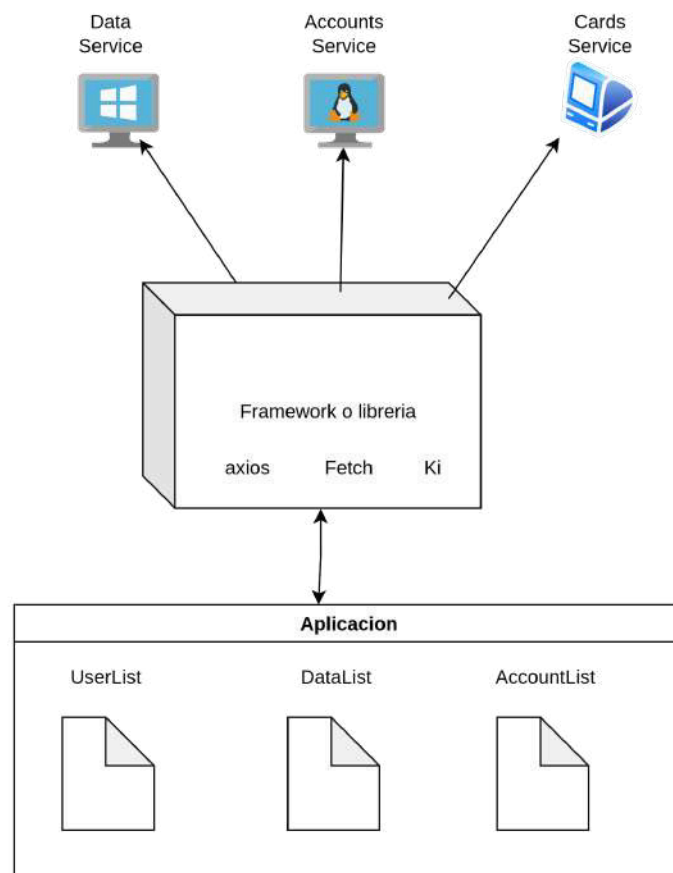


Figura 40 Aplicación del patrón bridge en la Capa de Interfaz con React para Abstraer Servicios de Datos

En síntesis, se aseguró que la nueva propuesta cumpliera con la función equivalente de la capa adapter; sin embargo, en lugar de basarse únicamente en el patrón Adapter, se incorporó el uso del patrón Strategy, que facilitó la selección dinámica de dependencias, y el patrón Bridge, que diversificó y extendió las herramientas en múltiples proyectos. Tal como lo indicó (Liu et al., 2024), los proyectos suelen aumentar su mantenibilidad y la repetición del código, este diseño obtuvo una estructura más clara, flexible y alineada como los principios de bajo acoplamiento, lo que incrementa la escalabilidad y la mantenibilidad del sistema a largo plazo.

4.2.1.2. Capas de uso

Esta capa representó la lógica de la aplicación y fue la encargada de interactuar entre las entidades de la capa de dominio, además gestionó el negocio del sistema sin estar comprometida con la capa externa, como Frameworks o librerías así como lo mencionó (Cecil, R, 2017).

Como aporte a la arquitectura, se dividió la capa en 2 módulos, el primer módulo como “Repositorio” y el segundo como “Servicios”, ambos respetaron el principio de responsabilidad única de la capa de uso; por esta razón, sólo usó esta capa para gestionar la parte funcional del proyecto y dependió del framework o librerías que deben estar encapsuladas para evitar estar comprometida con algún aspecto técnico excluyente de la lógica del sistema así como lo indicó el principio de inversión de dependencias, para ello se usó el patrón Strategy para la elección de la tecnología, el patrón constructor, para construir el servicio según la definición del repositorio y el principio de inversión de dependencia, con el fin de encapsular a la tecnología, tal como lo mencionaron (Arias-Orezano et al.,2021) en su trabajo donde usa los principios SOLID.

El módulo de Repositorio fue el director que tuvo la responsabilidad de definir cómo se construyeron los servicios de nuestra aplicación; en consecuencia, se necesitó de los principios de software así como indicó (Yanakiev, I., et.al., 2025). Los principios que se emplearon son los de “abierto y cerrado” y el principio de “segregación de interfaces” con el fin de incluir o eliminar más interfaces o eliminar según la necesidad de negocio.

El módulo de servicio se implementó según la dirección del módulo de repositorio de la misma manera usamos los principios como el de inversión de dependencia, debido que este módulo es de alto nivel y no debe depender de detalles concretos para su construcción, sino de abstracciones de las herramientas tecnológicas, a su vez, en su implementación es donde se procesa el módulo del servicio, actuando como una caja negra para otras capas. La Figura 41 representó cómo se construyó un servicio mediante pasos definidos construidos a través del repositorio, utilizando una relación didáctica en comparación de cómo se moldea un sándwich de una máquina industrial con pasos definidos.

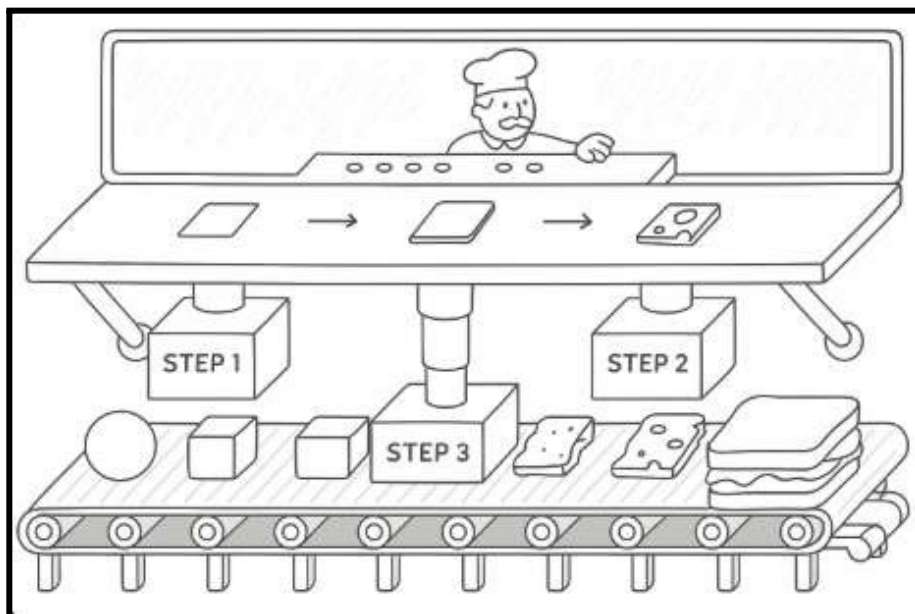


Figura 41 Aplicación de la capa de uso, donde se define la funcionalidad de crear un servicio según los pasos definidos.

4.2.1.3. Capas de Entidad

La capa entidad o de dominio constituyó el núcleo del negocio, ya que esta tuvo la responsabilidad de encapsular las entidades de negocio, tal como lo señaló (Cecil, R. ,2017) y también en la Figura 42. Esta capa se caracteriza por su independencia, puesto que no requiere estar acoplada a otras capas, lo que permitió mantener una arquitectura limpia y alineada con su principio de responsabilidad única.

En nuestra propuesta, reconocimos la importancia de su responsabilidad, por ello, realizamos una sugerencia, en aquellos casos donde el negocio presente una gran complejidad en sus reglas o estructura, se consideró la creación de un módulo denominado *structure*, este módulo tuvo la finalidad de incorporar patrones estructurales, los cuales permitieron modelar, organizar y dar forma al negocio de una manera más flexible y escalable. Tal como lo señaló (Aktar et al., 2025), el uso de patrones resulta clave para facilitar la escalabilidad y el crecimiento ordenado de los proyectos.

Este enfoque permitió la adaptación del sistema frente a futuros cambios en los requerimientos del dominio, garantizo que la arquitectura permanezca más robusta.



Figura 42 Representación de entidad y atributos que constituyen en la capa entidad

4.2.2. Comunicación entre capas

La comunicación entre capas se mantiene definida de manera unidireccional, avanzando siempre desde las capas más externas hacia las más internas, tal como lo propuso (Cecil, R., 2017), este enfoque guarda similitud con la metáfora concéntrica planteada en el trabajo original, en donde cada capa externa mantiene una responsabilidad y la capa externa depende de la siguiente más interna, pero nunca a la inversa. De este modo, se garantizó que las reglas de negocio contenidas en el núcleo del sistema permanezcan independientes y libres de acoplamientos con las capas externas, por consiguiente, las capas periféricas siempre apuntan hacia el núcleo, lo que permite observar la estabilidad del dominio, asegurando que los cambios de usuario no afecten la lógica central del negocio, tal como se representó la Figura 43 donde los cambios más variables están en la capa más externa dejando libremente a la capa entidad.

En la presente propuesta se planteó la incorporación de un puerto de acceso para la gestión de dependencias, lo cual permitió reducir la complejidad de la capa Adapter. Esto se debe a que, a medida que el proyecto evolucionó, dicha capa también aumentó en tamaño y responsabilidad, por ello generó dificultades en su mantenimiento, con este enfoque, el puerto

de acceso se ubicó en la capa más variable definida como interfaz, dado que es en este nivel donde se proporcionó las herramientas necesarias para interactuar con las capas externas. A medida que avanzamos hacia el núcleo del sistema, la importancia de estas dependencias se va reduciendo progresivamente, hasta volverse inexistente en la capa de entidad, lo que permite mantenerla completamente independiente.

En síntesis, se mantiene el flujo de comunicación unidireccional entre capas y se logró que la nueva propuesta cumpla el rol equivalente al de la capa adapter, solo comunicándose hasta la capa de uso para brindar tecnología.

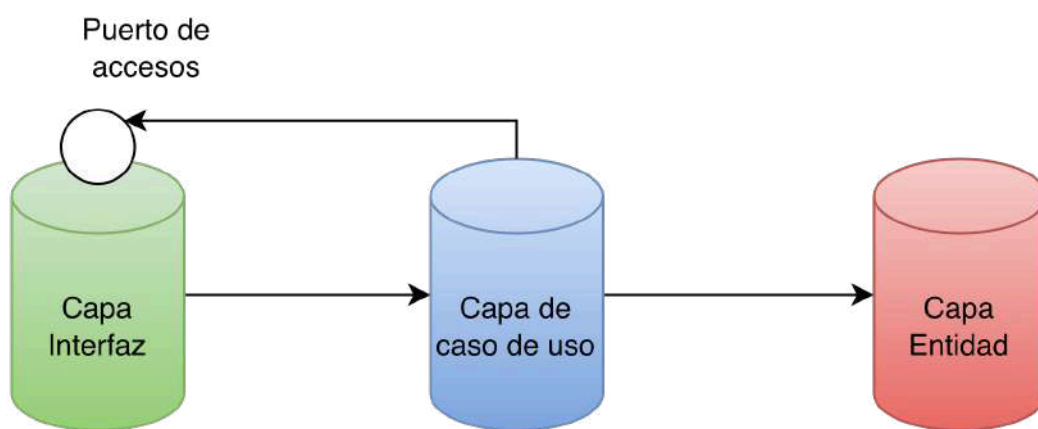


Figura 43 Comunicación de capas de manera unidireccional

4.2.3. Selección de la tecnología

La selección de la tecnología constituye un aspecto relevante dentro del proceso de diseño e implementación de la arquitectura y es una etapa muy compleja, tal como lo mencionan (Swacha et al., 2023). No obstante, en nuestra propuesta esta decisión se encuentra encapsulada y delegada a la capa más externa, lo que garantizó que el núcleo del sistema permanezca independiente, libre de acoplamientos. Según (Rojas et al., 2021) los sistemas debieron reducir los costos y tener sus funciones definidas así como se presenta en nuestra arquitectura.

La verdadera importancia de esta elección recae en la experiencia del desarrollador y en su capacidad para identificar qué tecnologías se adaptan mejor a las necesidades específicas del negocio como (Rathinam, S., 2022), nos orienta al comparar entre tecnologías React, Angular

y Vue para identificar sus beneficios y sus desventajas de usar una tecnología. De este modo, el desarrollador puede comparar alternativas tecnológicas, evaluando criterios como escalabilidad, mantenibilidad, rendimiento o facilidad de integración.

Asimismo, este proceso de decisión es muy complejo tal como lo sugiere (Avilés Matute, S., et .al, 2020) donde la mejor opción tecnológica estuvo en función del contexto y los requisitos del proyecto tal como se indica en la Figura 44, el cual debió favorecer un diseño más claro y alineado con los principios de bajo acoplamiento y alta cohesión para mejorar la evolución de los sistemas.

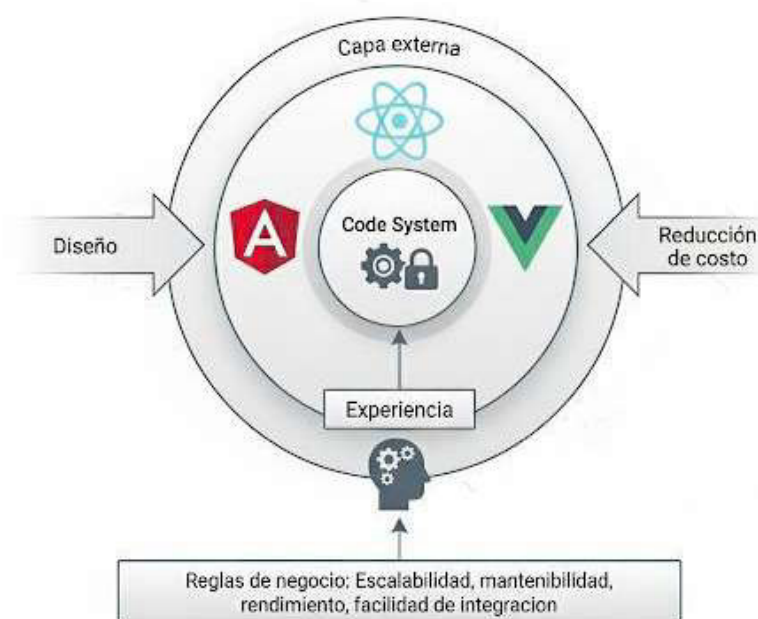


Figura 44 Selección de la tecnología del diseño Arquitectónico

4.3. Implementación del sistema

En la construcción del sistema se dividirá en 3 partes: (i) Diseño del sistema, donde se define los módulos del sistema, como el módulo de películas (ii) Implementación de la arquitectura, donde se da la integración con el Framework de desarrollo seleccionado y tecnologías adyacentes, para este medio se usó el entorno de desarrollo integrado (IDE) Visual Studio Code (iii) Uso del sistema, donde se explicó el funcionamiento del sistema con sus principales módulos que se han desarrollado.

En este punto se explicó el proceso del diseño de la arquitectura utilizada en esta investigación, este punto puede ser comprendido como indica en la Figura 45.

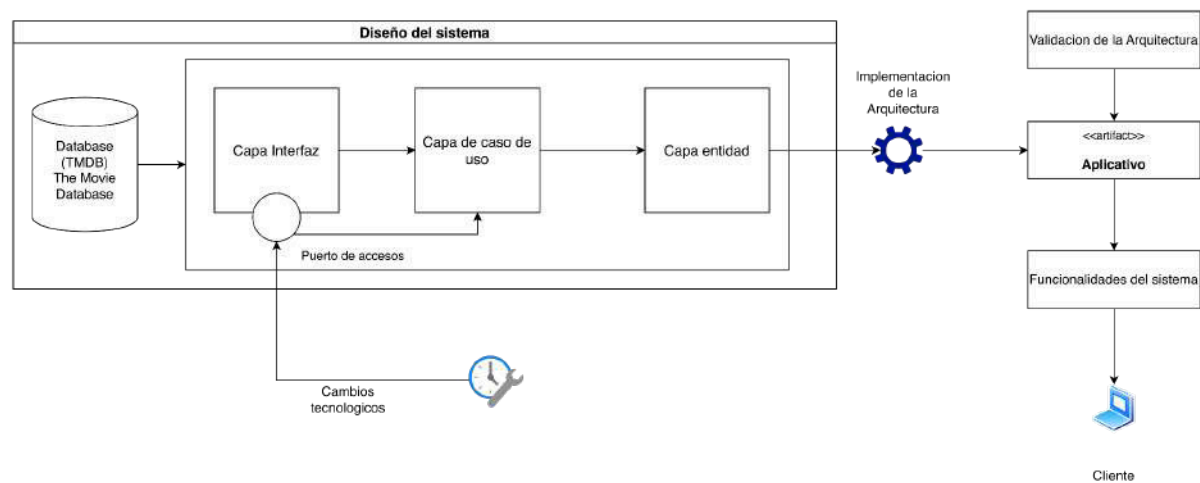


Figura 45 Metodología de la implementación de la arquitectura de 3 capas

4.3.1. Diseño del Sistema

4.3.1.1. Database TMDB

Para la implementación del sistema se utilizó la base de datos The Movies Database el cual contiene una base actualizada con todas las películas estrenadas a nivel global. Está database cuenta además con los siguientes grupos de atributos mostrados en la Tabla. 3

Tabla 3 Métodos usados para la implementación del sistema

Nombre	Definición	Métodos HTTP
Crear Sesión	Este método crea un ID de sesión totalmente válido	POST
Crear token	Este método crea un token de sesión totalmente válido	GET
Obtener cuenta	Este método Obtiene los datos de la cuenta	GET
Obtener estado de la cuenta	Este método obtiene el rating, lista de, favoritos	GET
Obtener película	Este método obtiene la película	GET

- **Crear sesión**

La Figura 46 indica el método de creación de sesión

```
https://api.themoviedb.org/3/authentication/guest_session/new
{
  succes: boolean,
  guest_session_id: string,
  expires_at: string
}
```

Figura 46 Método de creación de sesión

- **Crear Token**

La Figura 47 indica el método de creación de Token

```
GET
https://api.themoviedb.org/3/authentication/token/
{
  "success":boolean,
  "expires_at":string,
  "request_token":string
}
```

Figura 47 Método de creación de token

- **Obtener cuenta**

La Figura 48 indica el método para obtener el detalle de la cuenta

```
GET
https://api.themoviedb.org/3/account?session_id
{
  "gravatar":{
    "hash":string
  }, "tmdb":{
    "avatar_path":null
  }
},
  "id":number,
  "iso_639_1":string,
  "iso_3166_1":string,
  "name":string,
  "include_adult":boolean,
  "username":string
}
```

Figura 48 Método para obtener detalle de la cuenta

- **Obtener el estado de la cuenta**

La Figura 49 indica el método para obtener el estado de la cuenta.

```
GET
https://api.themoviedb.org/3/movie/911430/account_states?session_id
{
  "id":number,
  "favorite":boolean,
  "rated":{"
    "value":number
  },
  "watchlist":boolean
}
```

Figura 49 Método para obtener el estado de la cuenta

- **Obtener la información de una película**

La Figura 50 indica el método para obtener una película

```

GET
https://api.themoviedb.org/3/movie/1180831?language=en-US&api_key=
{
  "adult":boolean,
  "backdrop_path":string,
  "belongs_to_collection":{
    "id":number,
    "name":string,
    "poster_path":string,
    "backdrop_path":string
  },
  "budget":number,
  "genres":
  [
    { "id":number,
      "name":string
    },
  ],
  "homepage":string,
  "id":number,
  "imdb_id":string,
  "origin_country":array,
  "original_language":string,
  "original_title":string,
  "overview":string,
  "popularity":number,
  "poster_path":string,
  "production_companies":[
    {
      "id":number,
      "logo_path":string,
      "name":string,
      "origin_country":string
    }
  ],
  "production_countries":array,
  "release_date":string,
  "revenue":number,
  "runtime":number,
  "spoken_languages":[
    {
      "english_name":string,
      "iso_639_1":string,
      "name":string
    }
  ],
  "status":string,
  "tagline":string,
  "title":string,
  "video":boolean,
  "vote_average":number,
  "vote_count":number
}

```

Figura 50 Método para obtener una película

4.3.1.2. Preprocesamiento de los datos

4.3.1.2.1. Diseño de entrada

Se consideró la gran cantidad de películas que tiene la base de datos y adicionalmente con el pasar del tiempo se actualiza constantemente; por esta razón, la base de datos TMDb nos proporcionó una estructura de paginación que es visible por la capa de interfaz (UI) y así reducir la latencia de nuestro proyecto.

- **Paginación:** La totalidad de los 1,079,147 registros de películas actuales diciembre del 2025 se organizó en 53,958 páginas. Esta división se realizó aplicando un tamaño de página fijo de 18 resultados por array.

Esta estrategia permite que la aplicación solo cargue y procese un subconjunto de datos a la vez, en lugar de la totalidad del dataset como lo indica la Figura 51.

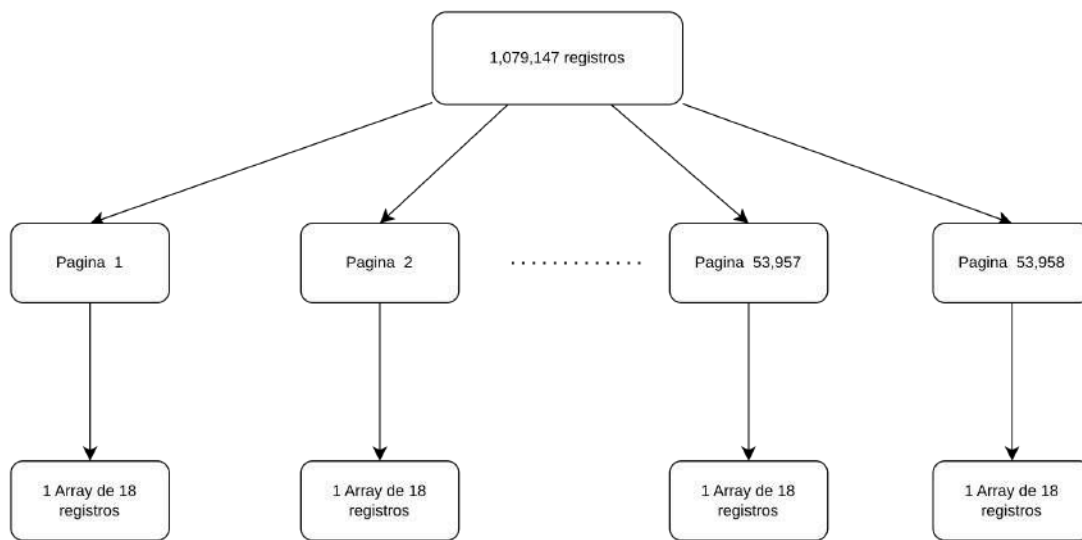


Figura 51 Diagrama de Paginación

4.2.3.2.2. Transformación de los datos de entrada

Para la obtención de los datos fue necesario un tratamiento de estos a través de la capa de entidad, debido a que con el pasar del tiempo es posible que la base datos cambie y nuestro proyecto debe estar preparado antes esta interrogante. La Tabla 4 representa la transformación de los datos.

Tabla 4 Diseño de la transformación de los datos de entrada

Propiedades de la entidad Película	Transformación (n)	Resultado
id	T(id)	id
title	T(title)	title
overview	T(overview)	overview
adult	T(adult)	adult
backdrop_path	T(backdrop_path)	backdrop_path
poster_path	T.poster_path)	poster_path
popularity	T(popularity)	popularity
release_date	T(release_date)	release_date
vote_count	T(vote_count)	vote_count

vote_average	T(vote_average)	vote_average
original_language	T(original_language)	original_language
asian_movie	isValidAsianMovie(original_language)	original_language === "zh"
discount	false	false

Como se observó, la capa entidad se encarga de transformar la entrada donde las propiedades de la capa entidad tiene una adaptación a la variabilidad de la base de datos, donde T(n), simboliza la transformación de las propiedades a un formato acorde a las entidades planteadas por el proyecto.

4.3.1.3. Diseño del sistema basado en 3 capas

El diseño del sistema según (Kerdoudi et al.,2022), partió de un objetivo que permita generar un buen ciclo de vida del software en el caso del Clean Architecture, por esta razón se presentó los diseños propuestos para las 3 capas planteadas.

Este sistema propuesto fue inducido en el entorno de desarrollo con el rol principal de gestionar la visualización de la cartelera de las películas, para ello se identificó los principales módulos que se se implementa en nuestro proyecto y cómo nuestros usuarios interactúan en ella como lo indica la Figura 52, en la representación de los módulos con los usuarios.

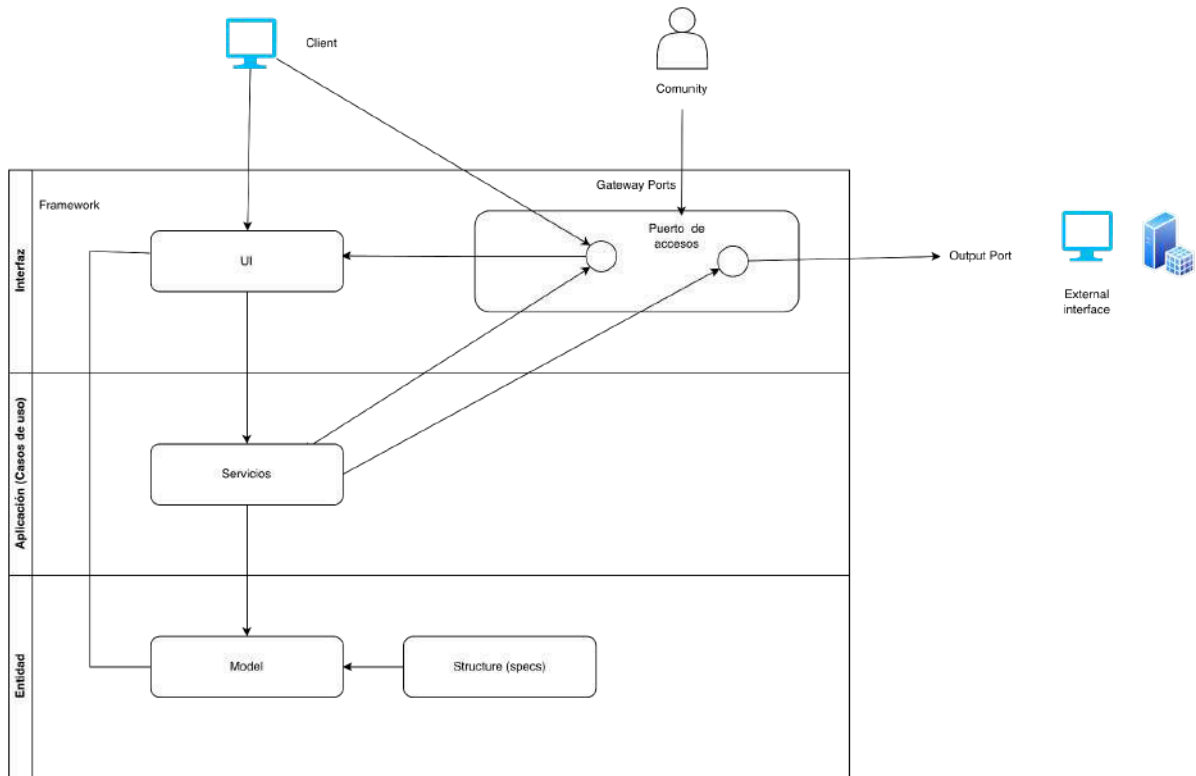


Figura 52 Diagrama general de diseño del sistema usando Clean Architecture de 3 capas

4.3.1.3.1. Diseño de la capa interfaz

Esta capa tiene como objetivo ser la más variable por ello es importante elegir la tecnología correcta para desarrollar la parte visual del sistema sin depender de ningún framework en específico como en (Saundariya, K., et. al., 2021), donde usando React nos permitió construir un sistema amigable y rápido, por esta razón, se decidió usar React como herramienta de desarrollo para nuestro sistema.

El uso de puertos de acceso nos permitió facilitar el desarrollo del sistema a través de la encapsulación de estas herramientas tecnológicas ya sea de origen externo (Público) o interno (Privado) de la organización del negocio. Estos puertos actúan como conectores entre el núcleo de la aplicación y los servicios externos, instalándose a través de dependencias que dieron posibilidad de reutilización de código no solo en este proyecto, sino también en otros futuros y dependen exclusivamente de la selección de los desarrolladores según los criterios convenientes, tal como lo indicó (Robayo, J., et al., 2024), respecto a la comparación de una tecnología con otra.

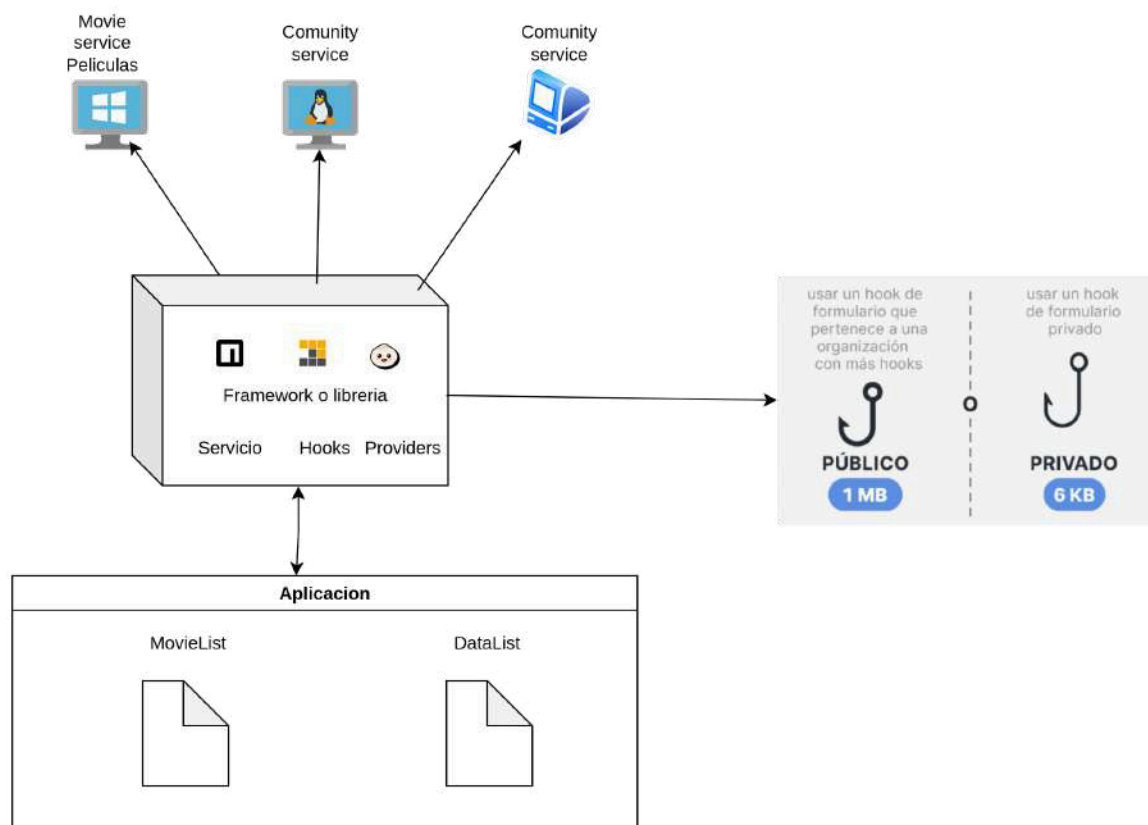


Figura 53 Diagrama de la capa interfaz para la aplicación del sistema de películas

La Figura 53 muestra la interacción y conexión que existe entre el módulo visual y el módulo de puertos que componen la capa interfaz, esta última la parte tecnológica, esta comunicación permite recibir y enviar información de una manera más amigable para el cliente.

4.3.1.3.2. Diseño de la capa de uso o aplicación

La capa de caso de uso se encarga de gestionar las funcionalidades de nuestro sistema de películas, el cual organiza en dos módulos principales el sistema: el repositorio, que actúa como director al definir y generar las plantillas de construcción lo cual garantizó una base ordenada y reutilizable, y el servicio, que mediante abstracciones tecnológicas implementó la lógica del negocio y concreta las operaciones necesarias para que el sistema funcione de manera eficiente; en conjunto, ambos modelos ofrecen una estructura sólida que facilita la escalabilidad, el mantenimiento y la traducción de los requerimiento del negocio en implementaciones técnicas sostenibles. Tal como lo mencionó (Kerdoudi et al., 2022) sobre la

arquitectura que deben evolucionar por ello esta capa lleva consigo los principios de abierto y cerrado y segregación de interfaces. La Figura 54 nos muestra la bifurcación de la capa de uso para posteriormente la construcción de un servicio dentro del sistema.

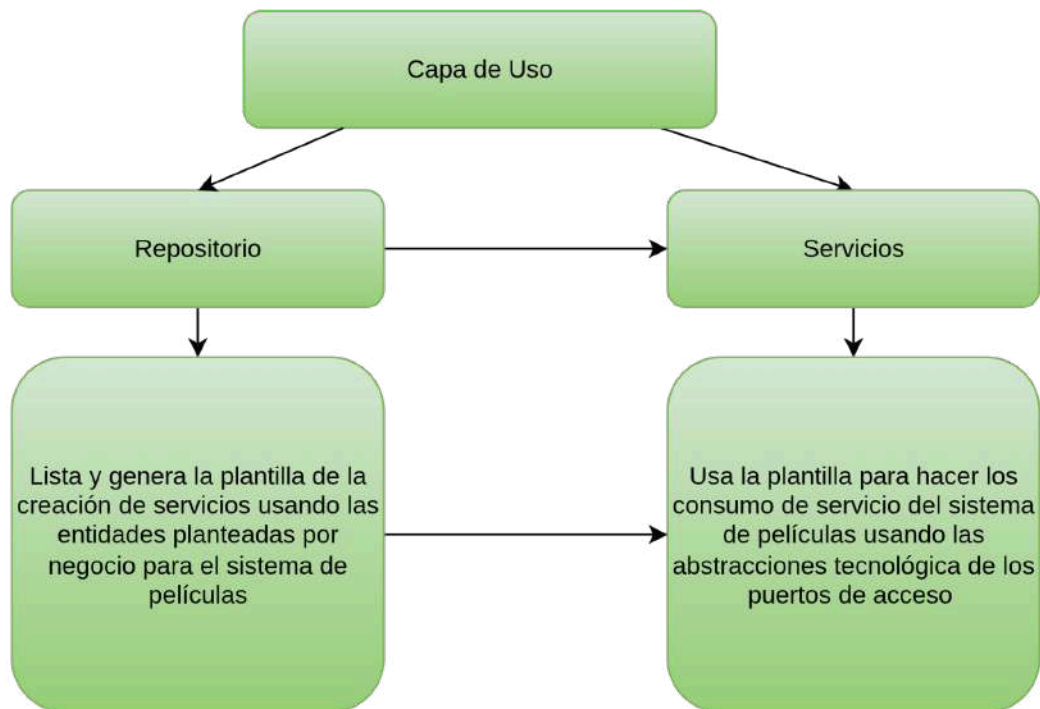


Figura 54 Diagrama de la capa de uso para la aplicación del sistema de películas.

4.3.1.3.3. Diseño de la capa entidad

Esta capa representa la lógica del negocio, tal como lo mencionó (Aguilar, P., & Figueira, L. B ,2020) en su trabajo, en ellas se definieron las entidades principales que constituyen el núcleo del sistema. A partir de estas entidades, se derivan los casos de uso que fueron gestionados en la capa correspondiente, garantizo así una separación clara de responsabilidades y una base sólida para el desarrollo de funcionalidades. Por esta razón en nuestro sistema identificamos dos entidades importantes la del usuario (Persona) y película (Movie). La Figura 55 representa cómo se conforma una entidad.

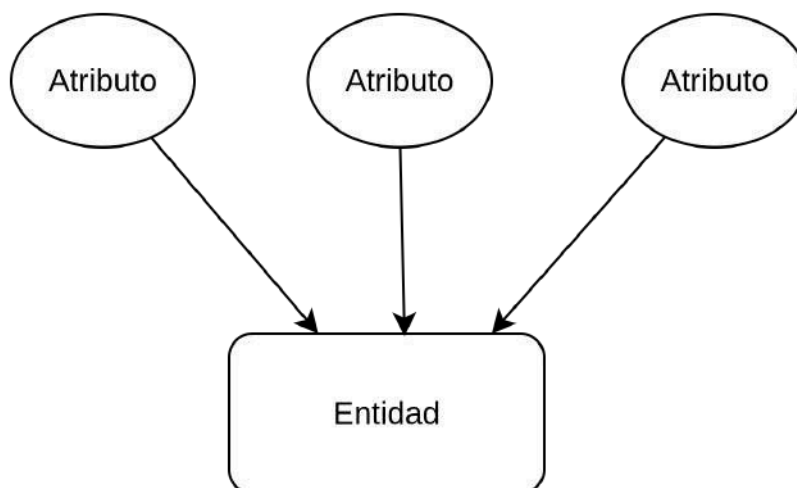


Figura 55 Diagrama de la capa entidad para la aplicación del sistema de películas

4.3.1.4. Vista del Sistema

Teniendo en cuenta la vista de representación del sistema y sus componentes, en este documento de investigación se colocó las vistas modulares, funcional y vista de componentes.

4.3.1.4.1. Vista modular

En la siguiente Figura 56, se muestra la vista modular separada en sus submódulos, la división propuesta: gestión de usuario, gestión de venta y gestión de producto.

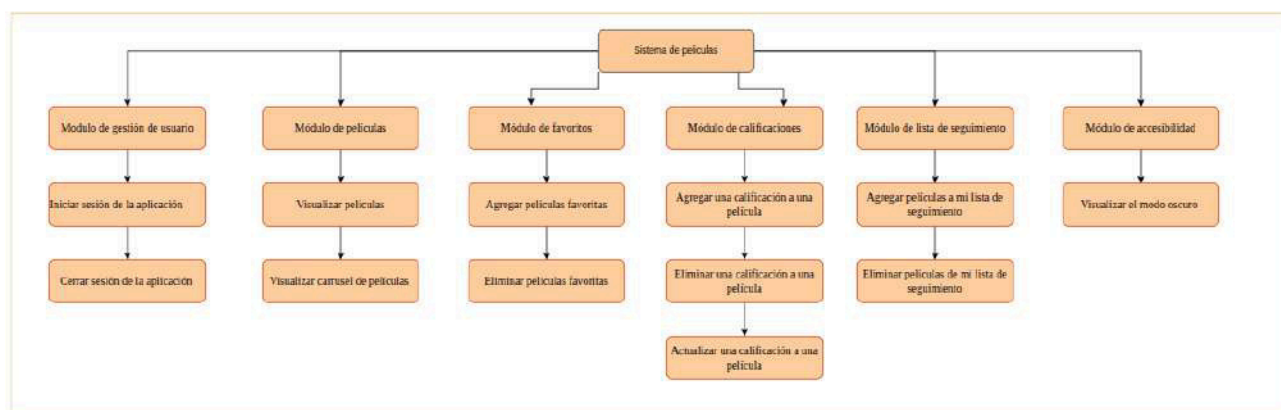


Figura 56 Diagrama de vista modular

4.3.1.4.2. Vista de casos de uso

En la siguiente Figura 57 se muestra la vista de casos de uso, se definen los módulos del sistema, y estos se vieron reflejados en los documentos de requisitos.

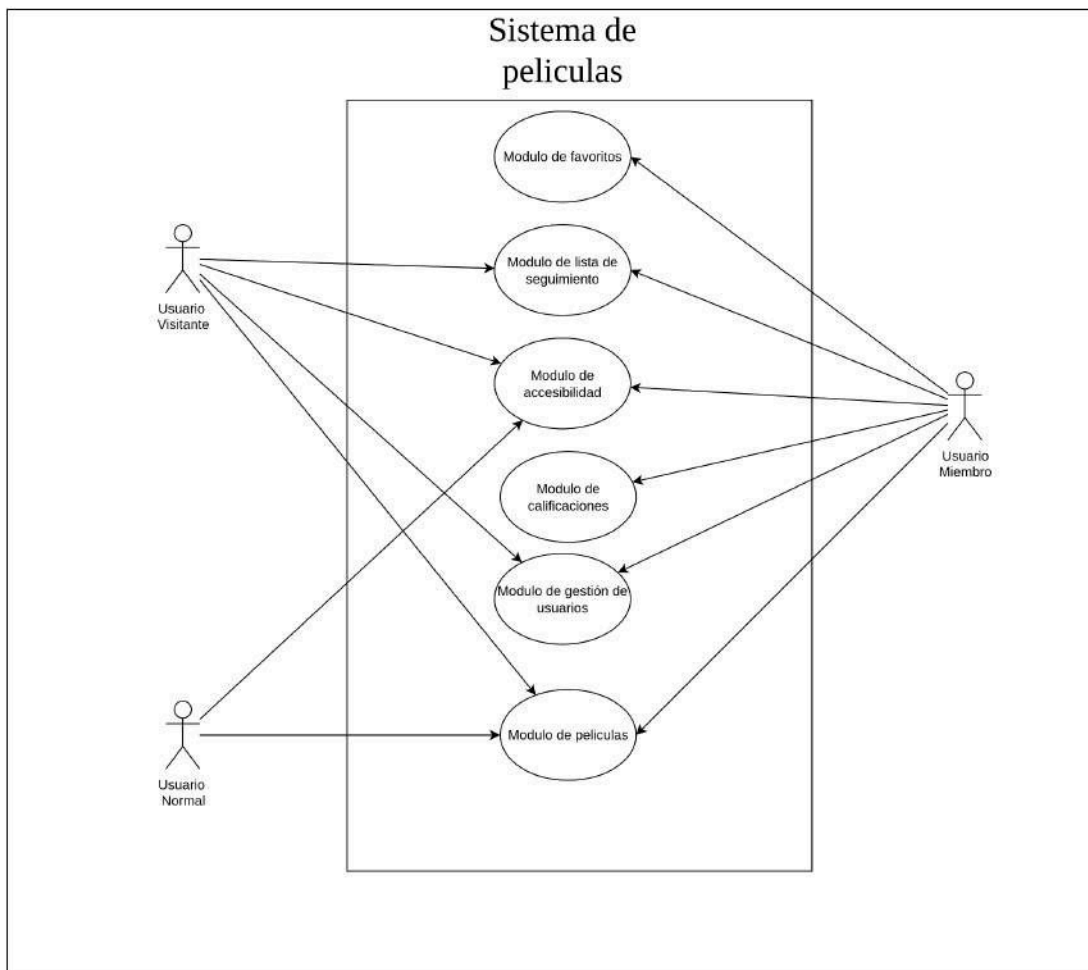


Figura 57 Diagrama de casos de uso

4.3.1.4.3. Vista de componentes

Está conformado por una capa presentación (**UI**), una capa de casos de uso representada por (**Application**) y las entidades como (**Entity**), la Figura 58 indica cómo están estructurados los componentes.

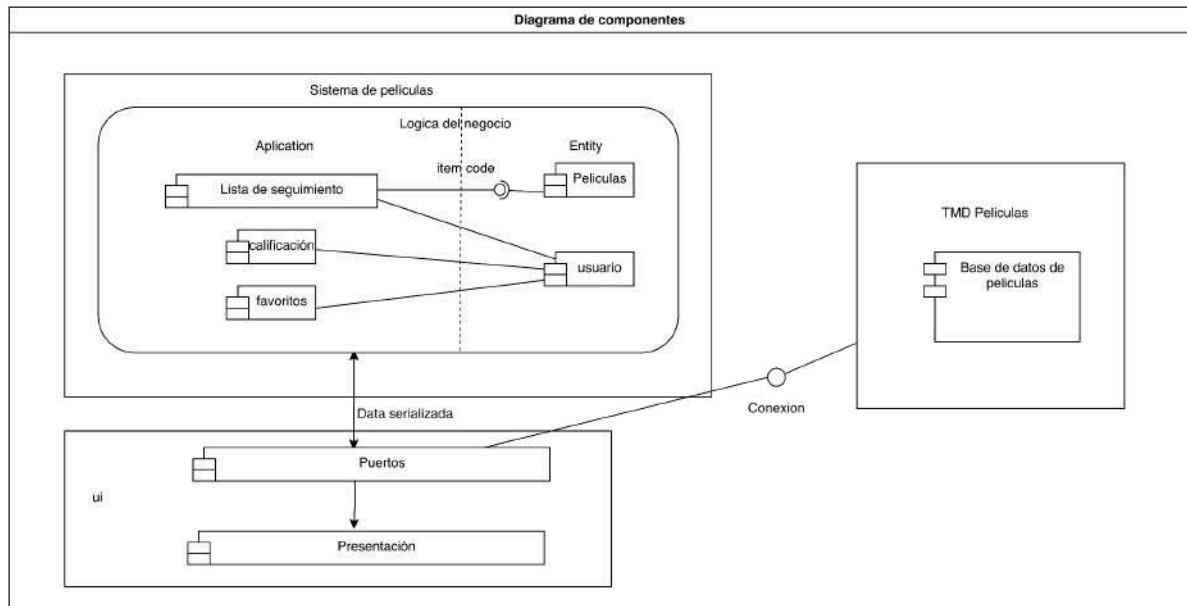


Figura 58 Diagrama de vista de componentes

4.3.1.5. Requerimientos

4.3.1.5.1. Alcance

Desarrollo de un sistema de películas Web como apoyo para evaluar el código estático del proyecto.

4.3.1.5.2. Actores del sistema

- Usuario Normal
- Usuario Visitante
- Usuario Miembro

4.3.1.5.3. Historia De usuario

En las historias de usuario se especificaron los requisitos para la construcción del sistema de películas acorde a la metodología ágil según en el Anexo A. La Tablas 5 a la Tabla 10 nos indica las historias de usuario analizadas.

Tabla 5 Historia de usuario número 1

HISTORIA DE USUARIO	
Número: 1	Actor(es): Usuario normal, Usuario

	visitante y Usuario miembro
Historia	Gestionar usuario
Prioridad: Alta	
Tiempo estimado: 2 días	
Descripción: El caso de uso de Autenticación de Usuario validó las credenciales para verificar la identidad, lo que garantizó que solo usuarios autorizados accedan de forma segura al sistema	
Pre-condiciones • Tener acceso a internet	
Post-condiciones • El sistema permite el ingreso del usuario	
Flujo Básico • Ingresa a la página web • Ingresa al sistema • Sale del sistema	
Flujo Excepción • Fallo en el servicio	
Criterios de Aceptación • El sistema valida la existencia del usuario	
Requerimientos asociados • FR001:Ingresar y cerrar sesión de la aplicación	
Prototipo Prototipo	

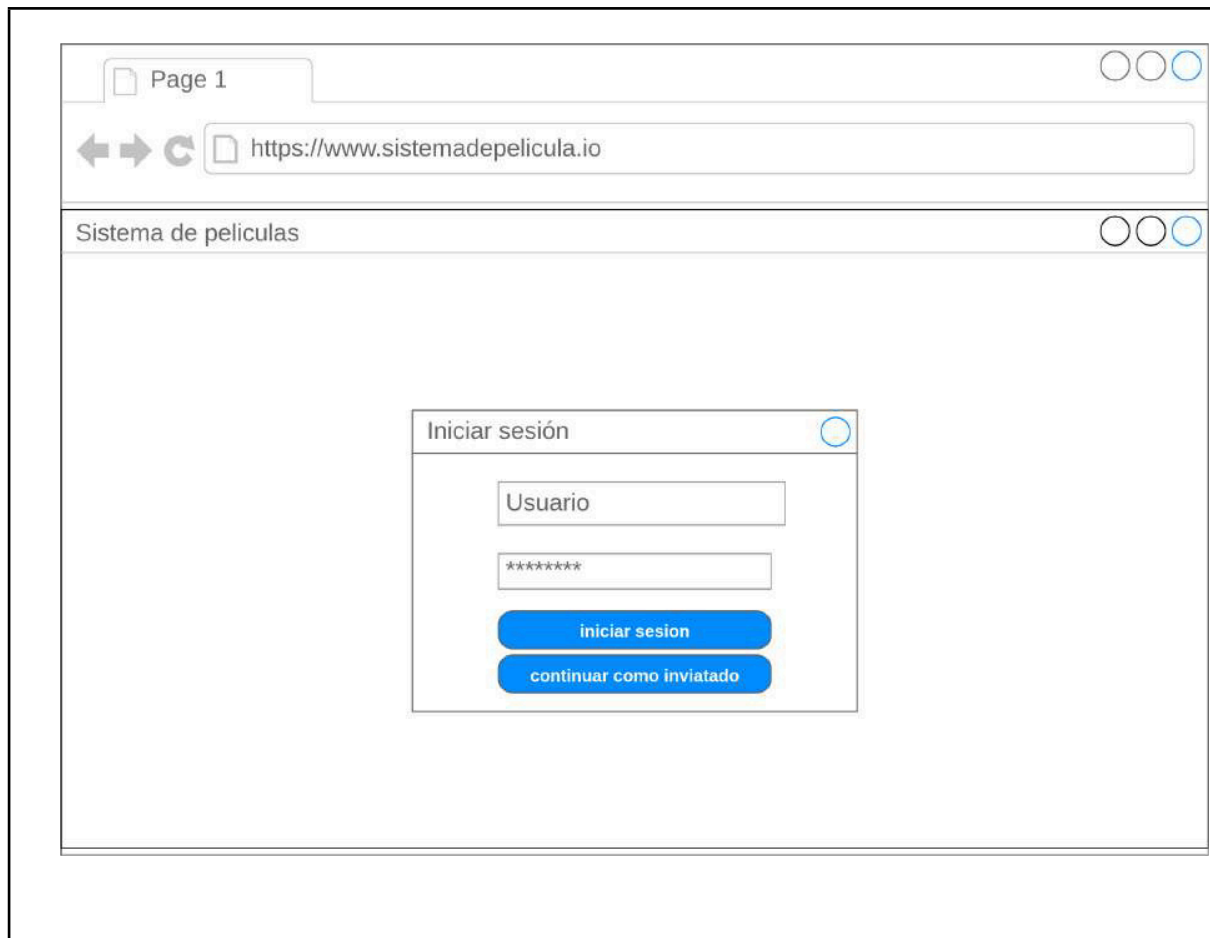


Tabla 6 Historia de usuario número 2

HISTORIA DE USUARIO	
Número: 2	Actor(es): Usuario normal, Usuario visitante y Usuario miembro
Historia	Gestionar Películas
Prioridad: Alta	
Tiempo estimado: 4 días	
Descripción: Se implementó el Módulo de Películas, donde se permite al usuario explorar de manera intuitiva la cartelera	
Pre-condiciones Se tiene que ingresar correctamente con el servicio de películas activo	

Post-condiciones

- El sistema permite la gestión de películas

Flujo Básico

- El usuario ingresa al sistema
- El usuario logra visualizar sus películas
- El usuario revisa el detalle de una película

Flujo Excepción

- Fallos con el servicio de película

Criterios de Aceptación

- El sistema logra mostrar el detalle de la película del usuario

Requerimientos asociados

- **FR002:**Visualizar película
- **FR003:**Visualizar carrusel de películas

Prototipo

Prototipo

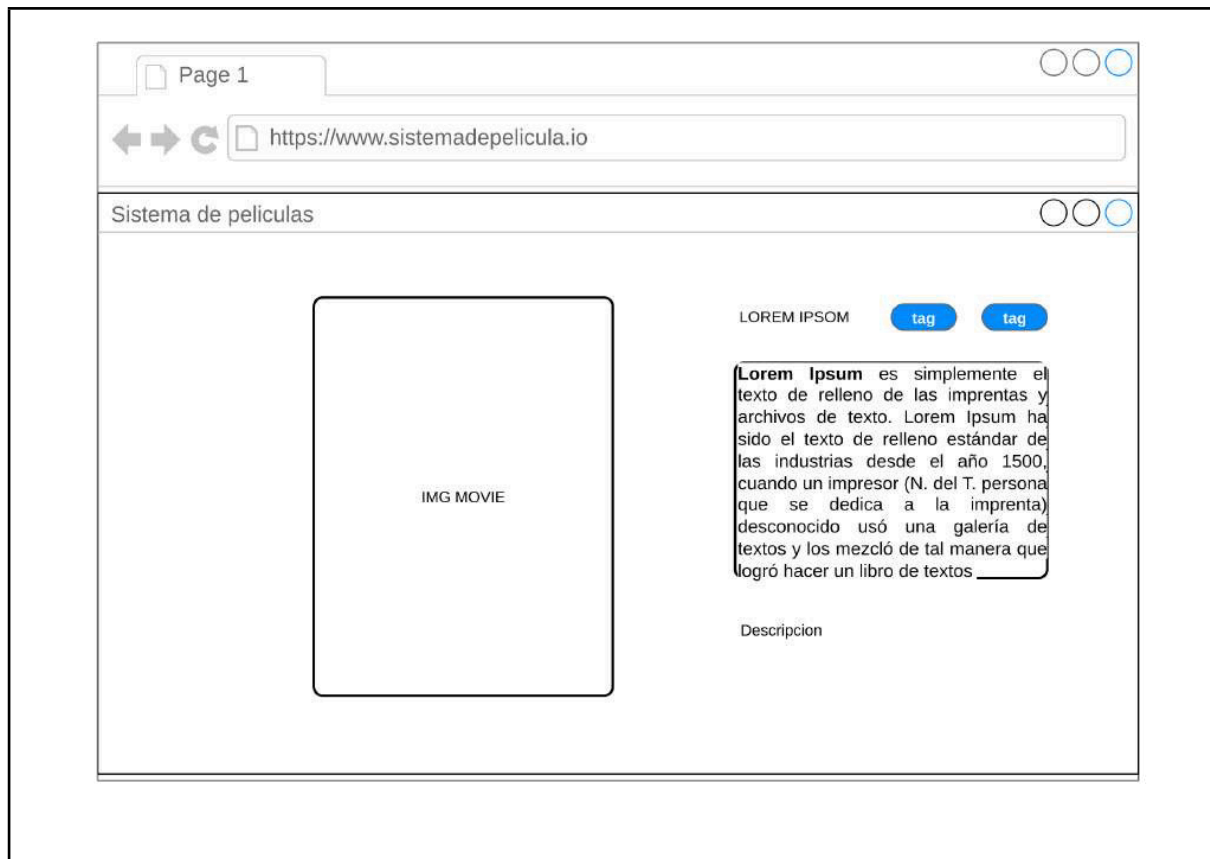
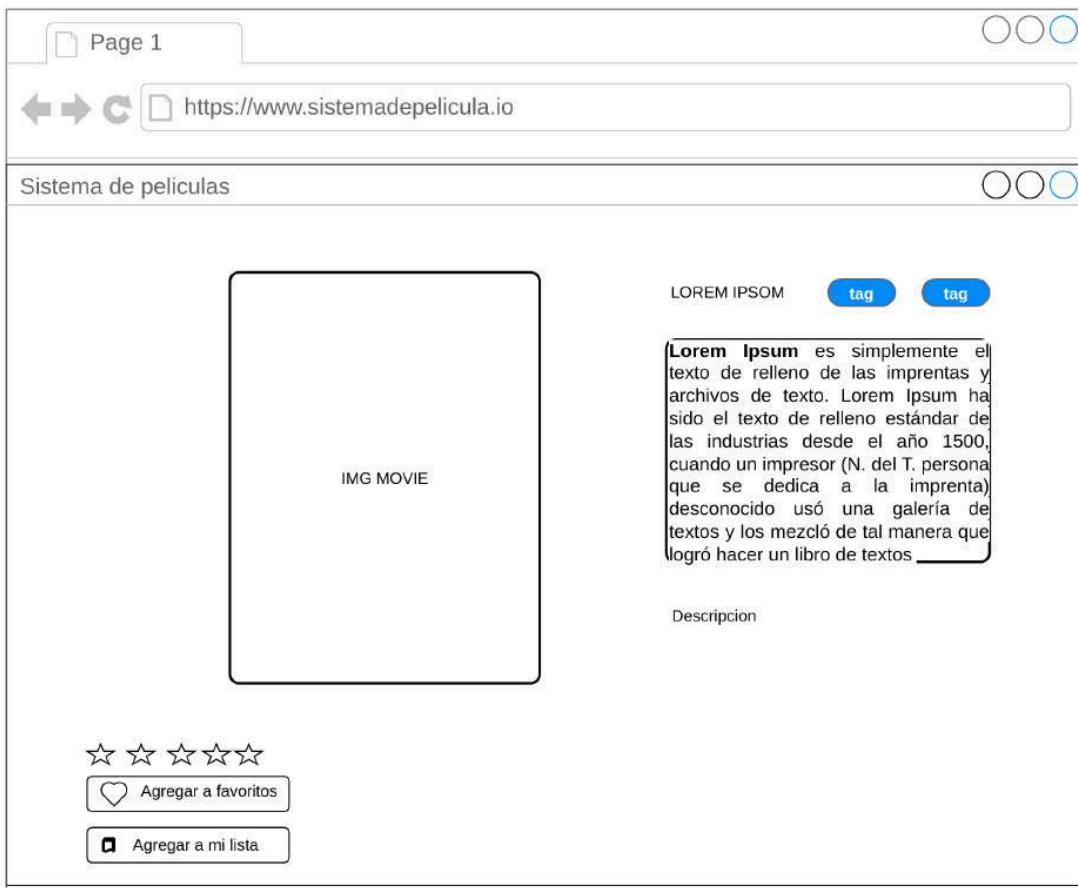


Tabla 7 Historia de usuario número 3

HISTORIA DE USUARIO	
Número: 3	Actor(es): Usuario Miembro
Historia	Gestionar Películas Favoritas
Prioridad: Media	
Tiempo estimado: 2 días	
Descripción: Se implementó el caso de uso de Gestión de Películas Favoritas, lo que permitió a los usuarios seleccionar, almacenar y consultar su lista personalizada de títulos dentro del sistema.	
Pre-condiciones	

• El usuario debe estar logueado
Post-condiciones • El sistema visualizar sus películas favoritas
Flujo Básico • El usuario ingresa al sistema • El usuario logra visualizar sus películas • El usuario añade a favoritos su película
Flujo Excepción • Fallos con el servicio de película
Criterios de Aceptación • El usuario visualiza sus películas favoritas
Requerimientos asociados • FR004:Agregar películas favoritas • FR005:Eliminar películas favoritas
Prototipo Prototipo



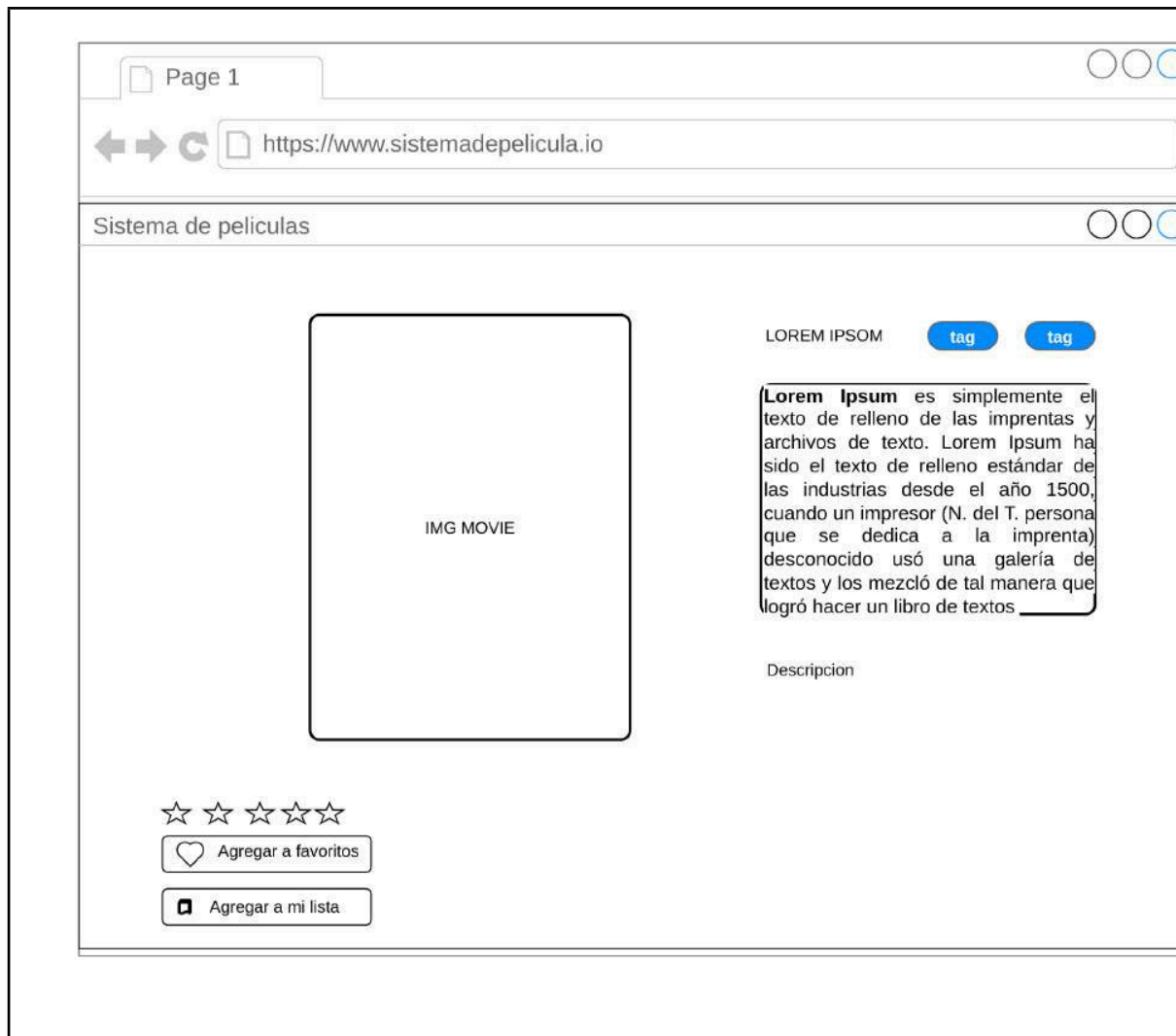


Tabla 8 Historia de usuario número 4

HISTORIA DE USUARIO	
Número: 4	Actor(es): Usuario Miembro
Historia	Gestionar calificación de películas
Prioridad: Media	
Tiempo estimado: 3 días	
Descripción: Se implementó el caso de uso de Asignación de Calificación, el cual permitió a los usuarios puntuar las películas y actualizar la calificación promedio de los títulos dentro del sistema.	

Pre-condiciones • El usuario debe estar logueado
Post-condiciones • El sistema visualiza la calificación de su película
Flujo Básico • El usuario ingresa al sistema • El usuario logra visualizar sus películas • El usuario añade a calificaciones a sus películas
Flujo Excepción • Fallos con el servicio de película
Criterios de Aceptación • El usuario visualiza sus películas calificadas
Requerimientos asociados • FR006: Agregar una calificación a una película • FR007: Actualizar una calificación a una película • FR008: Eliminar una calificación a una película
Prototipo Prototipo

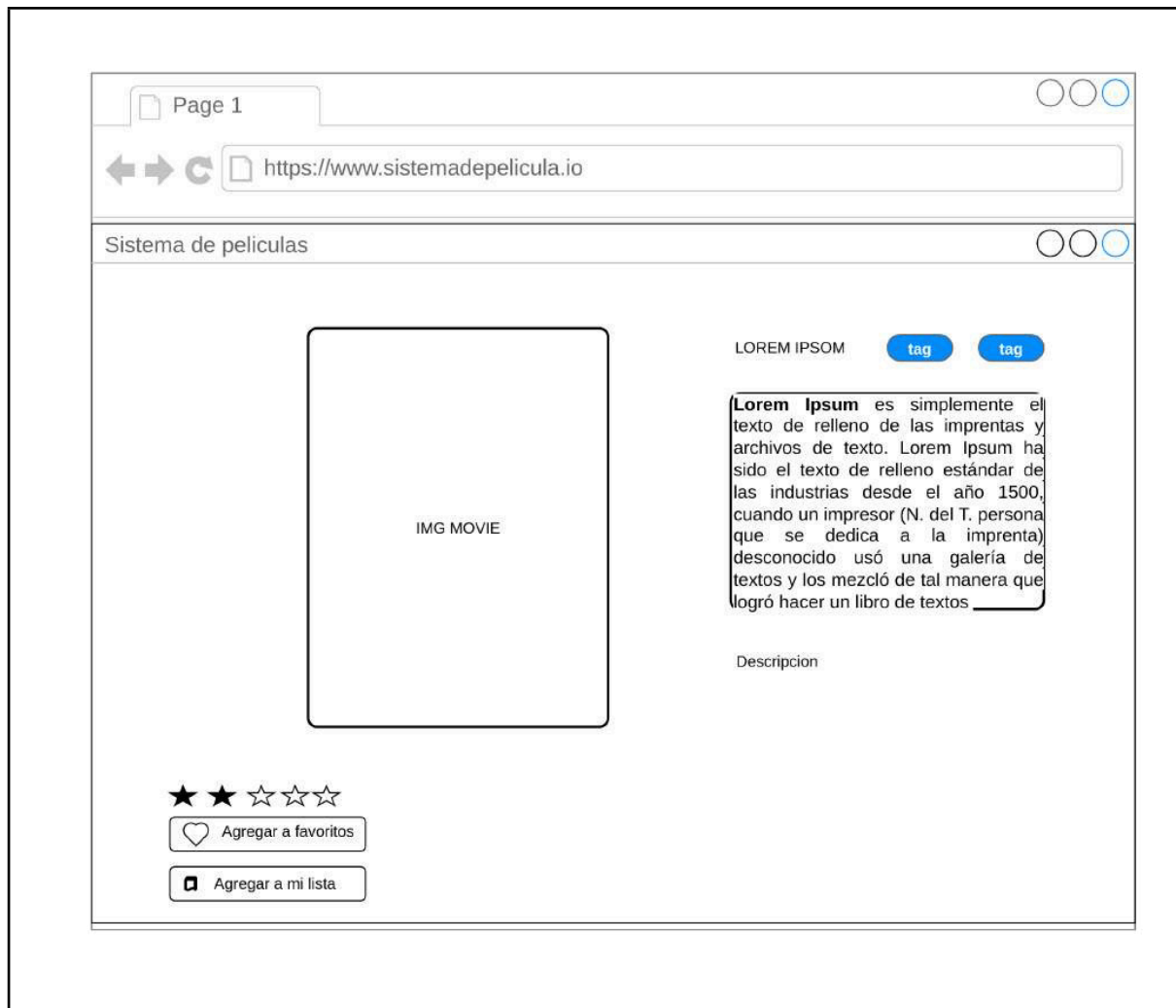
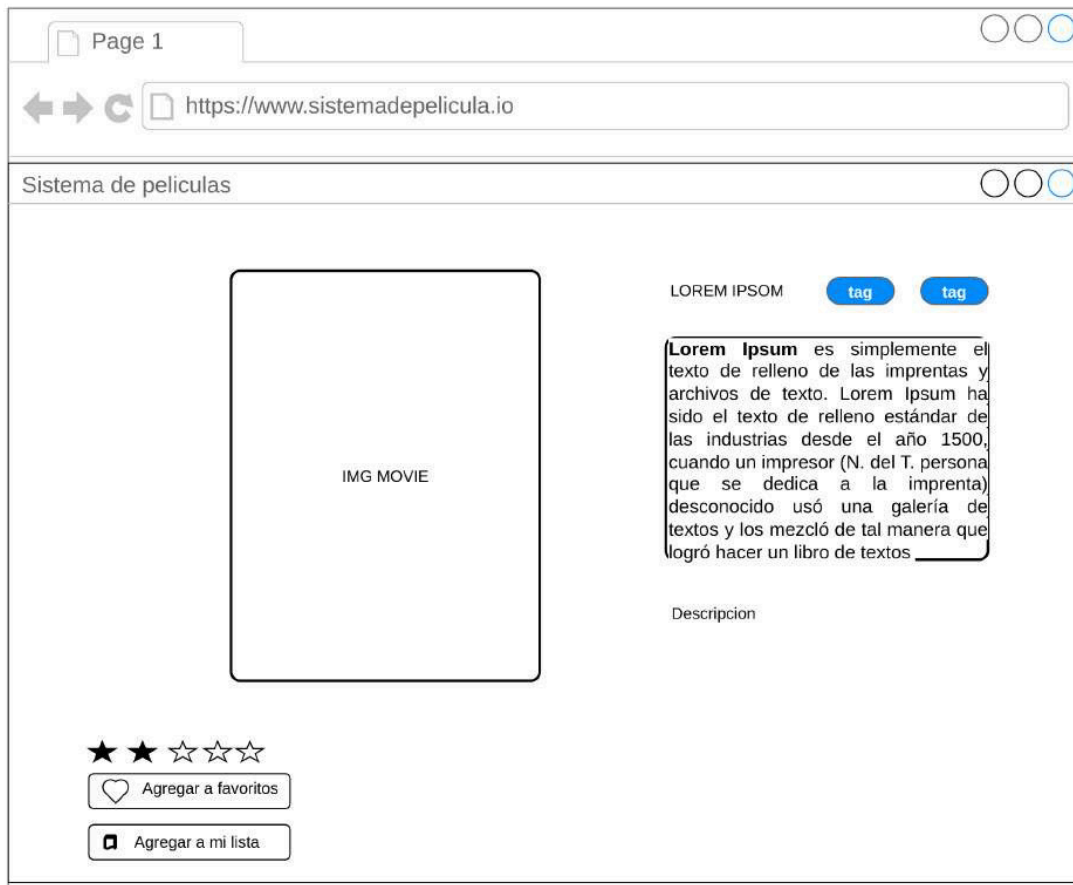


Tabla 9 Historia de usuario número 5

HISTORIA DE USUARIO	
Número: 5	Actor(es): Usuario visitante y Usuario Miembro
Historia	Gestionar la lista de Seguimiento
Prioridad: Media	
Tiempo estimado: 2 días	
Descripción: Se implementó el caso de uso de Gestión de Lista de Seguimiento (Watchlist), el cual otorgó a los usuarios la capacidad de crear, modificar y consultar una lista personalizada de películas pendientes de ver.	
Pre-condiciones	

• El usuario debe estar logueado
Post-condiciones • El sistema visualizar su lista de películas seguidas
Flujo Básico • El usuario ingresa al sistema • El usuario logra visualizar sus películas • El usuario añade una película a su lista de seguimiento
Flujo Excepción • Fallos con el servicio de película
Criterios de Aceptación • El usuario visualiza su nueva película en la lista de seguimiento
Requerimientos asociados • FR009: Agregar películas a mi lista de seguimiento • FR010: Eliminar películas de mi lista de seguimiento
Prototipo Prototipo



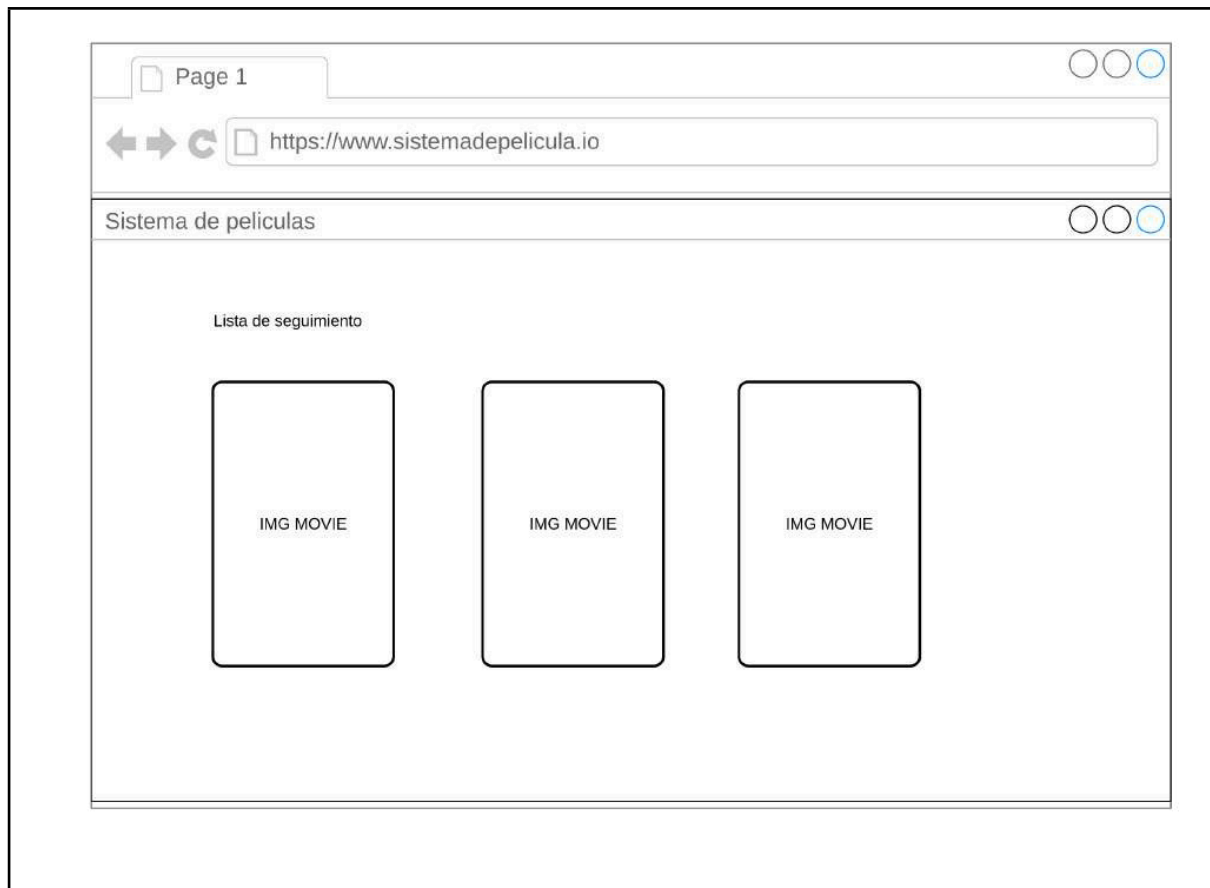


Tabla 10 Historia de usuario número 6

HISTORIA DE USUARIO	
Número: 6	Actor(es): Usuario normal, Usuario visitante y Usuario Miembro
Historia	Gestionar la accesibilidad
Prioridad: Media	
Tiempo estimado: 2 días	
Descripción: Se implementó el caso de uso de Accesibilidad del Usuario, lo que aseguró que el sistema cumpliera el modo oscuro del sistema	
Pre-condiciones El usuario debe acceder a internet	
Post-condiciones El sistema visualiza su accesibilidad.	

Flujo Básico

- El usuario ingresa al sistema
- El usuario selecciona una accesibilidad
- El usuario visualiza e cambio dentro del sistema

Criterios de Aceptación

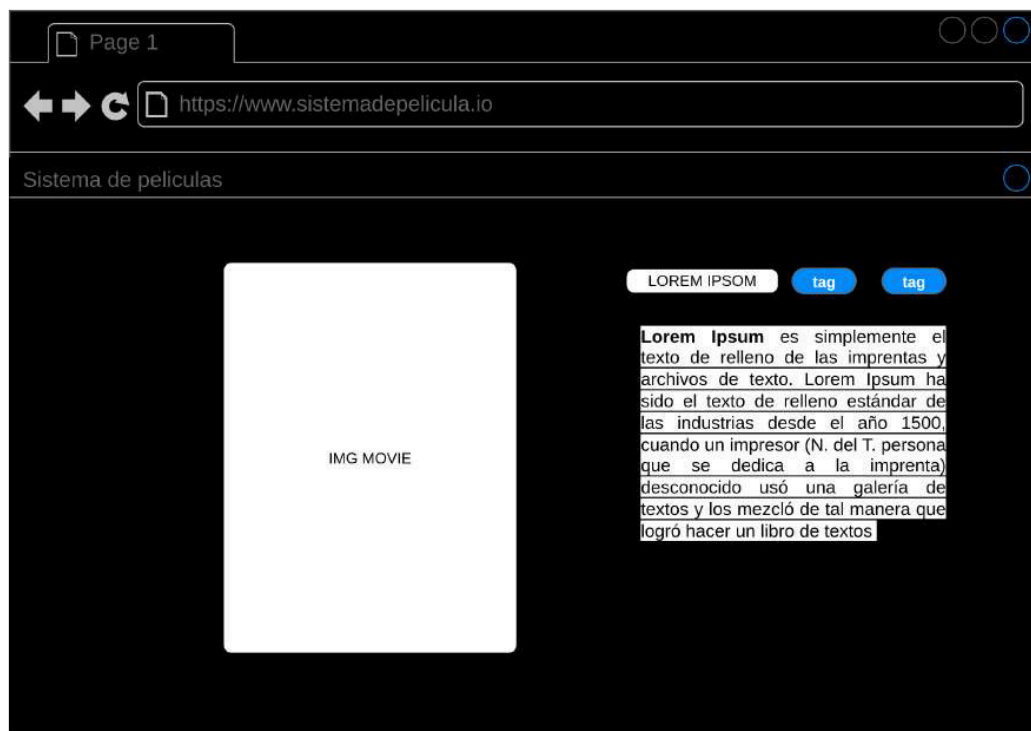
- El usuario visualiza el modo oscuro del sistema

Requerimientos asociados

- **FR011:**Visualizar el modo oscuro

Prototipo

Prototipo



4.3.1.5.4. Product Backlog

En el Product Backlog se enlista la historia de usuario y cómo se relacionan con los sprints, según como lo indica la siguiente Tabla 11.

Tabla 11 *Product Backlog*

Sprint	Identificador de la HU	Requerimiento	Tiempo estimado	Prioridad
Sprint 1	HU01	FR001	2	Alta
	HU02	FR002	4	Alta
		FR003		
	HU03	FR004	2	Media
		FR005		
	HU04	FR006	3	Media
		FR007		
		FR008		
Sprint 2	HU05	FR009	2	Media
		FR010		
	HU06	FR011	2	Media

4.3.2. Implementación de la arquitectura

Este punto corresponde a la Fase de implementación de la arquitectura donde se planteó la metodología de investigación mencionada en el anterior capítulo. Esta fase comprende las siguientes etapas: diseño del sistema, se define los módulos del sistema, implementación de la arquitectura, donde se construye con la integración del Framework de desarrollo seleccionado, funcionalidades del sistema, explica el funcionamiento del sistema con sus principales módulos. La siguiente arquitectura de 3 capas, se ilustra el software construido con la solución propuesta, con la finalidad de ser validado, también se mostró lo necesario

para su implementación como se muestra en la Figura 59 y 60 representa la estructura del repositorio y la implementación del sistema respectivamente.

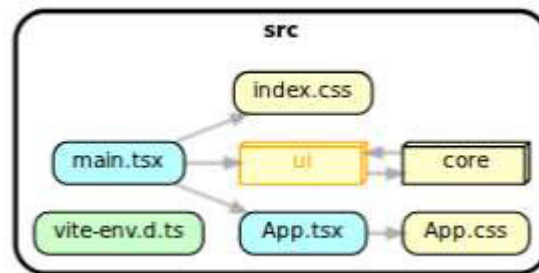


Figura 59 Diagrama del repositorio

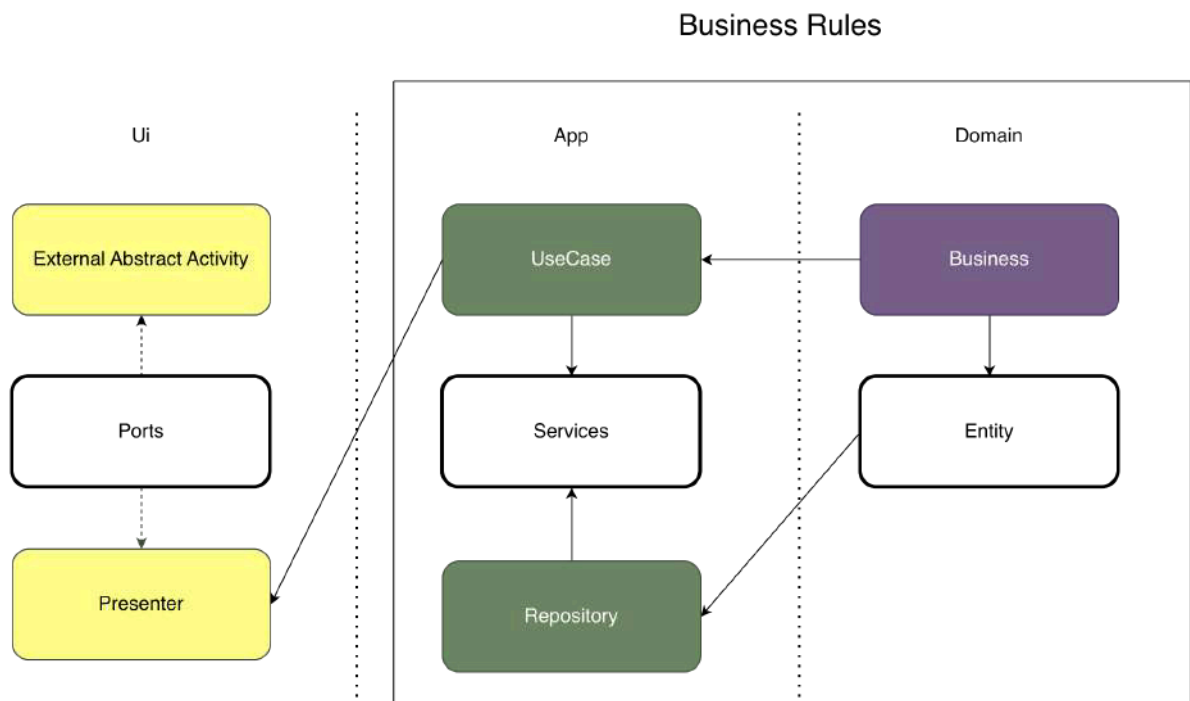


Figura 60 Diagrama de implementación del sistema

Para llevar a cabo su implementación se decidió usar Visual Studio como IDE de desarrollo web, el editor y sus herramientas nos permiten funciones que aumentan nuestra productividad de trabajo en la compilación del Aplicativo Web, además de herramientas Lint para detectar problemas de errores de sintaxis. La Figura 61 representa el IDE de trabajo.

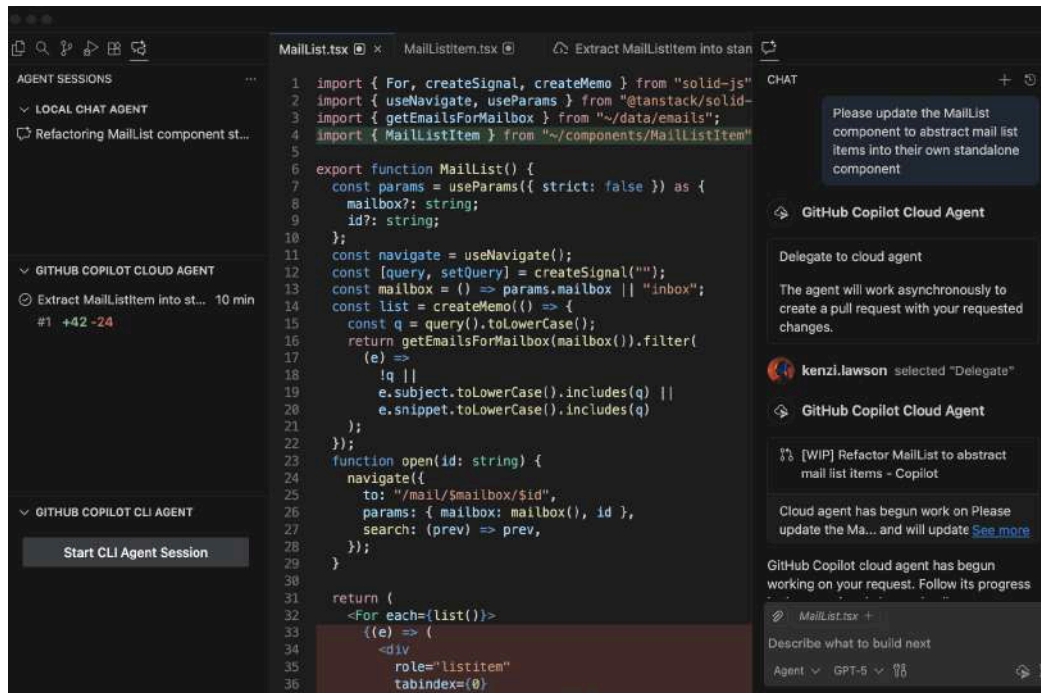


Figura 61 Captura de Pantalla de Visual Studio Code (Visual Studio Code)

4.3.2.1. Implementación de la capa interfaz

Contiene carpeta contenedora (UI) y fue la parte de presentación más variable de nuestro aplicativo. La Figura 63 muestra el paquete ui.

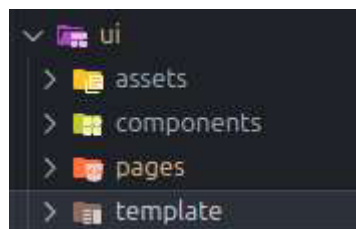


Figura 63 Diagrama de paquete del presenter (UI)

- **Components:** Contiene un bloque de construcción reutilizable, dentro de los componentes se componen por átomos que son los componentes más pequeños como los botones y las moléculas, pequeñas implementaciones como los cards o un formulario. La Figura 64 muestra la estructura del paquete de componentes.

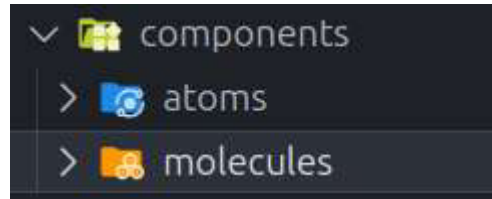


Figura 64 Diagrama de paquetes de components

Los componentes tienen propiedades que son pasados a través del ciclo de vida del Framework o tecnología además de tener variantes de esta misma y ser usados por otros componentes así como la Figura 65 muestra al Button.ts.

```
const buttonVariants = cva({
  variants: {
  },
  defaultVariants: {
    variant: "default",
    size: "default",
  },
})

function Button({
  className,
  variant,
  size,
  asChild = false,
  ...props
}: React.ComponentProps<"button"> &
VariantProps<typeof buttonVariants> & {
  asChild?: boolean
}) {
  const Comp = asChild ? Slot : "button"

  return (
    <Comp
      data-slot="button"
      className={cn(buttonVariants({ variant, size, className }))}
      {...props}
    />
  )
}

export { Button, buttonVariants }
```

Figura 65 Estructura de un button

- **Pages:** Son componentes más grandes que hacen uso de los componentes más pequeños que formó parte del diseño haciendo uso de las herramientas del framework para alimentar la data de nuestra página, en este caso se usó, los hooks del useEffect y el servicio movie de listar. La Figura 65 muestra la estructura de la página Home.

```

type CardMapMoviesProps = {
  movies?: WrapMovie[];
};

const HomePage = () => {
  const [movies, setMovies] = useState<WrapMovie[]>();

  useEffect(() => {
    const fetchMovies = async () => {
      const [data] = await movieServices.getAll();
      setMovies(data);
    };

    fetchMovies();
  }, []);

  return (
    <>
      <CarrouselMovies />
      <span className="px-10 text-2xl font-semibold text-neutral-900">
        Películas populares
      </span>
      <CardMapMovies movies={movies} />
    </>
  );
};

export default HomePage;

const CardMapMovies = ({ movies }: CardMapMoviesProps) => {
  return (
    <div className="grid grid-cols-4 gap-2 px-2 lg:px-6">
      {movies?.map((movie: WrapMovie) => (
        <CardMovies
          key={movie.id}
          id={movie.id}
          title={movie.title}
          poster_path={movie.poster_path}
          vote_average={movie.vote_average}
          adult={movie.adult}
          asian_movie={movie.asian_movie}
        />
      ))}
    </div>
  );
};

```

Figura 66 Estructura de la página Home del proyecto

- **Templates:** Este actúa como un contenedor estructural unificado para las páginas permitiendo la reutilización y optimización. La Figura 67 muestra la estructura del Template del proyecto.

```

import { AppSidebar } from "../../components/molecules/app-sidebar";
import { SiteHeader } from "../../components/molecules/site-header";

type Props = {
  sidebar?: boolean;
  children?: React.ReactNode;
};

export default function withDashboard({ sidebar = false, children }: Props) {
  return (
    <SidebarProvider
      style={
        {
          "--sidebar-width": "calc(var(--spacing) * 72)",
          "--header-height": "calc(var(--spacing) * 12)",
        } as React.CSSProperties
      }
    >
      <SideBar sidebar={sidebar} />
      <SidebarInset>
        <SiteHeader />
        <div className="flex flex-1 flex-col">
          <div className="@container/main flex flex-1 flex-col gap-2">
            <div className="flex flex-col gap-4 py-4 md:gap-6 md:py-6">
              {children}
            </div>
          </div>
        </div>
      </SidebarInset>
    </SidebarProvider>
  );
}

const SideBar = ({ sidebar }: { sidebar: boolean }) =>
  sidebar ? <AppSidebar variant="inset" /> : null;

```

Figura 67 Estructura de un Template

Además la carpeta (ports) contenedora de la adaptabilidad ya no como patrón Adapter sino haciendo uso de los patrones bridge como puente que abstrae y el patrón Strategy para tomar la mejor decisión respecto a la tecnología y dependencias, es decir sirve como enlace para dar una solución a los cambios constantes así como se muestra en la Figura 68.

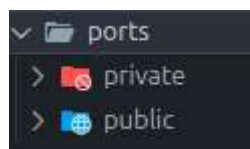


Figura 68 Diagrama de paquetes de los puertos

- **Internal port:** Maneja la tecnología interna (private) seleccionada base como dependencia, en este caso se realizó una abstracción de theme para la accesibilidad del sistema como el modo oscuro así como en la Figura 69.

```

import { STORAGE_KEYS } from "@lib/configuration/storage";
import type { Theme, ThemePort } from "../types";

export class LocalStorageThemeAdapter implements ThemePort {
  private readonly DEFAULT_THEME: Theme = "light";

  getTheme(): Theme {
    try {
      const storedTheme = localStorage.getItem(STORAGE_KEYS.THEME);
      if (storedTheme === "light" || storedTheme === "dark") {
        return storedTheme;
      }
      return this.DEFAULT_THEME;
    } catch (error) {
      console.error("Error getting theme from localStorage:", error);
      return this.DEFAULT_THEME;
    }
  }

  setTheme(theme: Theme): void {
    try {
      localStorage.setItem(STORAGE_KEYS.THEME, theme);
      this.applyThemeToDocument(theme);
    } catch (error) {
      console.error("Error setting theme in localStorage:", error);
    }
  }

  toggleTheme(): void {
    try {
      const currentTheme = this.getTheme();
      const newTheme: Theme = currentTheme === "light" ? "dark" : "light";
      this.setTheme(newTheme);
    } catch (error) {
      console.error("Error toggling theme:", error);
    }
  }

  private applyThemeToDocument(theme: Theme): void {
    const htmlElement = document.documentElement;

    if (theme === "dark") {
      htmlElement.classList.add("dark");
    }
  }
}

```

Figura 69 Módulo del Puerto privado o interno

- External port:** Gestiona las dependencia externas variables que tienen un impacto grande en nuestro proyecto (public), comúnmente soluciones de la comunidad. Actualmente, se utiliza la dependencia Axios, en consiguiente a nuestro encapsulamiento a través de este puerto garantiza la abstracción tecnológica, es decir la lógica de negocio no tiene dependencia con Axios y se pueda sustituir en el futuro con otra librería alternativa como Ki así como se muestra en la Figura 70.

```

import axios from "axios";

interface petition {
  url: string;
  params?: any;
  data?: any;
}

const doGet = async (p: petition) => {
  try {
    const response = await axios.get(p.url);
    console.log("Response data:", response);
    return response.data;
  } catch (error) {
    console.error("Error fetching data:", error);
    throw error;
  }
}

const doPost = async (p: petition) => {
  try {
    const response = await axios.post(p.url, p.data);
    return response.data;
  } catch (error) {
    console.error("Error posting data:", error);
    throw error;
  }
}

```

Figura 70 Módulo del Puerto público o externo

Adicionalmente, para una mejor gestión de posibles depreciamientos en las dependencias, se recomienda enlazar el repositorio a un gestor de dependencias automatizado así como se indicó en nuestro diseño propuesto en la Figura 71, como Dependabot.

dependabot/dependabot-core

#12535 **Dependabot
uses Node.js v20.19.3
and NPM 10.8.2. Due t...**

7 comments



Figura 71 Alerta de versión de node

Entonces obtuvimos el flujo variable en este módulo Interfaz que parte de la capa presentación antes mencionada en Clean Architecture, adicionalmente se puede considerar la carpeta (utils), contenedora de funciones, constantes, etc..

4.3.2.2. Implementación de la capa de uso o aplicación

Contiene casos de uso del sistema y está relacionado con las reglas de negocio por lo que es vital su independencia con otras capas como se representa en la Figura 72.

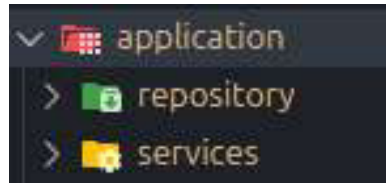


Figura 72 Diagrama de paquete de la capa de uso

- **Repository:** Es el director de las plantillas para crear los servicios, esta gestión de datos se llevó a cabo a través de repositorios independientes, siguiendo el principios de separación de responsabilidades así como se muestra en las Figura 73 y Figura 74.
 - Person Repository, encargado de las operaciones de almacenamiento y consultas de usuarios.

```
import type { GuestUser } from "@core/adapters/entities/guest";
import type { WrapMovie } from "@core/domain/entities/movie";

export interface GuestRepository {
  createGuestSession(): Promise<GuestUser>;
  validateGuestSession(sessionId: string): Promise<boolean>;
  logout(sessionId: string): Promise<void>;
  getGuestDetails(sessionId: string): Promise<GuestUser | null>;

  rateMovie(
    sessionId: string,
    movieId: number,
    rating: number
  ): Promise<boolean>;
  deleteMovieRating(sessionId: string, movieId: number): Promise<boolean>;
  getGuestRatedMovies(sessionId: string, page?: number): Promise<WrapMovie[]>;
}
```

Figura 73 Estructura del repositorio de Usuario visitante

- Movie Repository, responsable de la administración de películas

```
import type { WrapMovie } from "@core/domain/entities/movie";
import type { MovieAccountStatesable, MovieId, MovieListable } from "../movies";

export interface MovieRepository
  extends MovieListable<WrapMovie>,
    MovieId<WrapMovie>,
    MovieAccountStatesable {}
```

Figura 74 Estructura del repositorio de películas

Estos repositorios fueron implementados inicialmente sobre una estructura de datos en memoria, lo que permitió validar la lógica de manera ágil, adicionalmente, se integró la API pública de TMDb que proporciona un catálogo actualizado de películas, lo cual garantiza la pertinencia de datos en tiempo real. Este enfoque no solo fomenta la abstracción y modularidad.

- **Services:** Es la plantilla establece un contrato estricto que se guía y se definió por las reglas del repositorio, es decir las funciones usadas en los servicios están estrictamente delimitadas por el repositorio, no puede existir funciones adicionales ni se pueden omitir las ya definidas, lo cual garantiza, así una consistencia funcional inmutable. El servicio expone los casos de uso de la aplicación y actúa como contrato hacia la presentación, esta última depende de esta para obtener las operaciones de datos. como se indica en la Figura 75.

```
import { doGet } from "../../../adapters/apis";
import type { Movie, WrapMovie } from "../../../domain/entities/movie";
import { createMovie } from "../../../domain/entities/movie";
import type { MovieRepository } from "../../../repository/movies/movieRepository";
import type { MovieAccountStates } from "../../../repository/movies/movies";

const API_URL = `${API_BASE}${TRENDING_MOVIES_PATH}?language=en-US`;
const API_ID = `${API_BASE}${MOVIE_PATH}`;

export const movieServices: MovieRepository = {
  async getAll(): Promise<WrapMovie[]> {
    let data: WrapMovie[];
    try {
      const dataResponse = await doGet({ url: API_URL });
      data = dataResponse.results.map((item: Movie) => createMovie(item));
    } catch (err) {
      console.error("Error fetching all movies:", err);
      throw err;
    }
    return [data];
  },

  async getById(id: number): Promise<WrapMovie> {
    try {
      const dataResponse = await doGet({
        url: `${API_ID}/${id}`,
        params: { language: "en-US" },
      });
      return dataResponse;
    } catch (err) {
      console.error("Error fetching movies with id ${id}:", err);
      throw err;
    }
  },
}
```

Figura 75 Estructura del servicio de películas

4.3.2.3. Implementación de la capa entidad

La capa entidad va relacionada con los objetos más importantes que tiene el negocio, es vital abstraer el producto de forma óptima.

- **Entity:** Contiene las entidades del sistema e implementa adaptabilidad de cambios del negocio. La implementación del sistema se estructura en torno a dos entidades fundamentales: Usuarios (Persons) y Películas (Movies), el diseño se apoyó en principios de reutilización y extensibilidad, así como en el uso de herencia, repositorios y patrones estructurales, lo que garantizó un modelo flexible y escalable. En relación con la entidad Usuario, se definió la clase base Person, de la cual derivan tres subtipos con diferentes niveles de privilegios así como en la Figura 76.
 - Usuarios:
 1. Usuario Normal, cuya funcionalidad se limita a la consulta de películas disponibles.
 2. Usuario Visitante, que además de la consulta tiene la opción de generar listas personalizadas de películas
 3. Usuario Miembro, que amplía sus capacidades al permitir acciones como organizar, eliminar o votar por títulos favoritos.

Esta jerarquía permite compartir atributos de identificación, al tiempo que se introduce comportamiento en función al rol del usuario, promoviendo así un diseño a la extensión sin necesidad de modificar las clases.

```

export function createGuestUser(guestResponse: GuestSession): GuestUser {
  return {
    id: guestResponse.guest_session_id,
    session_id: guestResponse.guest_session_id,
    guest_session_id: guestResponse.guest_session_id,
    session_type: "guest",
    success: guestResponse.success,
    expires_at: guestResponse.expires_at,
    created_at: new Date().toISOString(),
    name: "Invitado",
    username: "guest",
    avatar_path: null,
    is_authenticated: true,
    user_type: "guest",
    capabilities: {
      can_rate: true,
      can_favorite: false,
      can_watchlist: false,
      can_access_account: false,
    },
  };
}

export function createMemberUser(
  sessionId: string,
  accountData: Account,
  requestToken?: string
): MemberUser {
  return {
    id: sessionId,
    session_id: sessionId,
    session_type: "member",
    account_id: accountData.id,
    username: accountData.username,
    name: accountData.name || accountData.username,
  };
}

```

Figura 76 Estructura de la entidad Usuario

Por su parte, la entidad Película se diseñó aplicando el patrón estructural Composite, lo que permite representar de manera uniforme películas individuales como agrupación, de este modo, el sistema facilitó la construcción de colecciones que entreguen diversos contenidos. Asimismo, se contempla la posibilidad de extender, como ZhMovie, destinada a incorporar metadatos adicionales en el caso de película de origen chino, evidenciando la capacidad del modelo para adaptarse al contexto cultural o de negocio específico como se indica en la Figura 77.

```

10
11 interface RatedMovie extends Movie {
12   genre_ids: number[];
13   original_title: string;
14   video: boolean;
15   rating: number;
16 }
17
18 function mapToWrapMovie(m: Movie | RatedMovie): WrapMovie {
19   return {
20     id: m.id,
21     title: m.title,
22     overview: m.overview,
23     adult: m.adult,
24     backdrop_path: m.backdrop_path,
25     poster_path: m.poster_path,
26     popularity: m.popularity,
27     release_date: m.release_date,
28     vote_count: m.vote_count,
29     vote_average: m.vote_average,
30     original_language: m.original_language,
31     asian_movie: isValidAsianMovie(m.original_language),
32     discount: false,
33   };
34 }
35
36 export function createMovie(m: Movie): WrapMovie {
37   return mapToWrapMovie(m);
38 }
39
40 export function createRatedMovie(m: RatedMovie): WrapMovie {
41   return mapToWrapMovie(m);
42 }
43
44 export const isValidAsianMovie = (original_language: string) =>
45   original_language === "zh";
46
47 export type { WrapMovie, RatedMovie };
48

```

Figura 77 Estructura de la entidad Película

4.3.2.4. Funcionalidades del sistema

4.3.2.4.1. Sprint 1

A. Desarrollo de la HU01 - Gestionar usuario

Para el desarrollo de la primera historia de usuario se tomaron los siguientes requerimientos.

- **RF001: Ingresar y cerrar sesión de la aplicación**

Para esta tarea se usó la api indicada en la **Figura 77** perteneciente a la base de datos de TMDB para la creación de usuario. Adicionalmente la estructura de los usuarios es la siguiente

En relación con la entidad Usuario, se definió la clase base Person, de la cual derivan tres subtipos con diferentes niveles de privilegios:

1. Usuario Normal, cuya funcionalidad se limita a la consulta de películas disponibles.
2. Usuario Visitante, que además de la consulta tiene la opción de generar listas personalizadas de películas
3. Usuario Miembro, que amplía sus capacidades al permitir acciones como organizar, eliminar o votar por títulos favoritos.

Esta jerarquía permite compartir atributos de identificación, al tiempo que se introduce comportamiento en función al rol del usuario, de esta manera se asigna el control de usuario y privilegio. El usuario puede loguearse como invitado o como miembro dentro del sistema así como lo muestra la Figura 78.

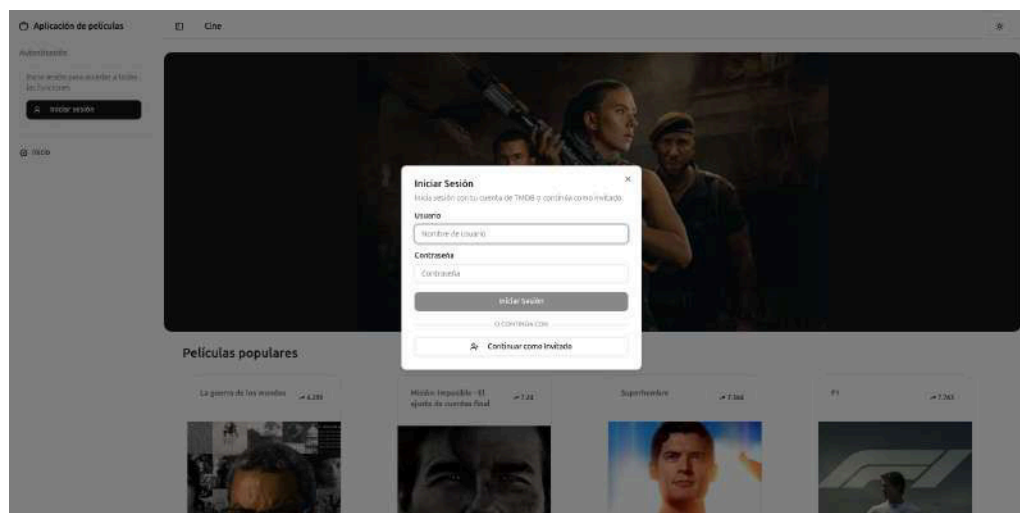


Figura 78 Inicio de sesión del sistema

El usuario podrá cerrar su sesión a través del sidebar del sistema así como lo indica en la Figura 79.

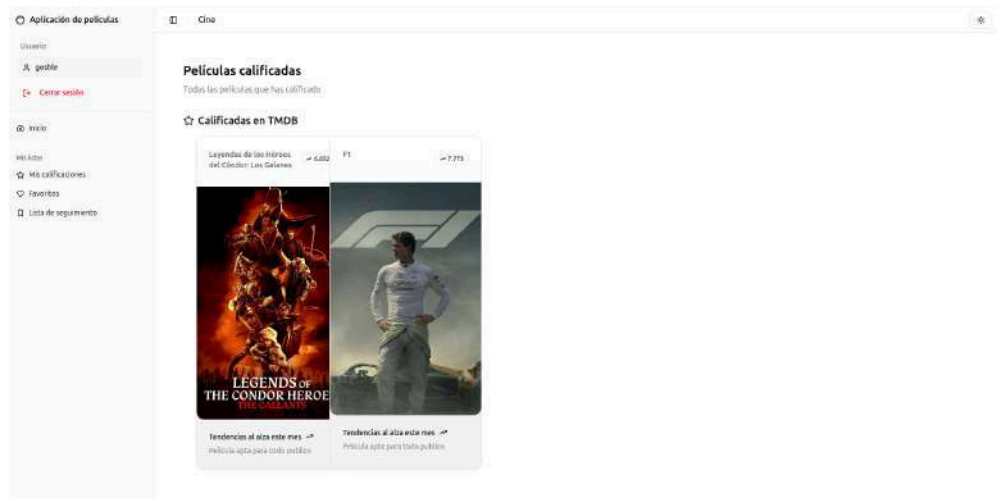


Figura 79 Cerrar sesión a través del sidebar

B. Desarrollo de la HU02 - Gestionar Películas

Para el desarrollo de la segunda historia de usuario se tomó los siguientes requerimientos.

- **FR002: Visualizar película**

Con los datos cargados y adaptados en la entidad de películas, se hace uso del servicio de películas que posteriormente alimenta a la UI con la data necesaria y mostrarlo en el DOM, de nuestros proyectos. El sistema cuenta con una página principal, como se muestra en la Figura 80, aquí el usuario puede visualizar las últimas películas de la cartelera.

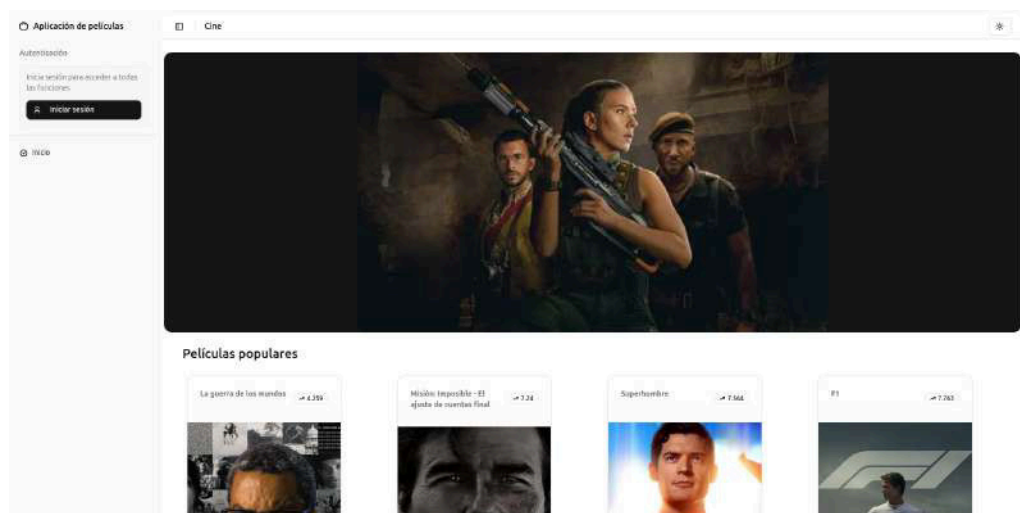


Figura 80 Landing del sistema

- **FR003: Visualizar carrusel de películas**

El componente de cartelera se encarga de obtener 3 imágenes seleccionadas estáticas bajo un algoritmo de movimiento que permite avanzar y retroceder al componentes. Este comportamiento se aprecia en la Figura 81.

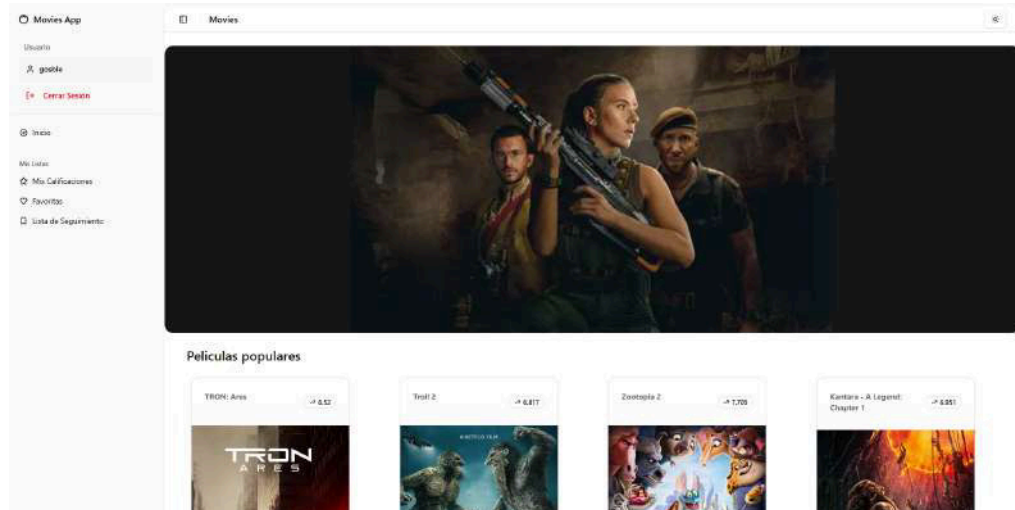


Figura 81 Visualizar carrusel de películas

C. Desarrollo de la HU03 - Gestionar Películas Favoritas

Para el desarrollo de la tercera historia de usuario se tomaron los siguientes requerimientos.

- **FR004: Agregar películas Favoritas**

Como usuario miembro puedo agregar a favoritos debido a la disponibilización del servicio de TMDB según lo indica la Figura. Esta funcionalidad de películas se visualiza en la Figura 82.

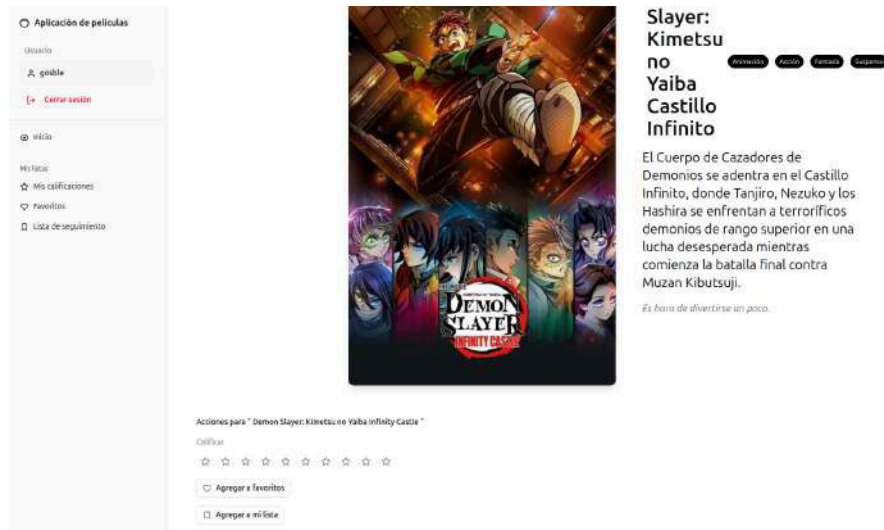


Figura 82 Agregar lista de favoritos

El usuario miembro puede observar su lista de películas favoritas así como lo muestra la Figura 83.

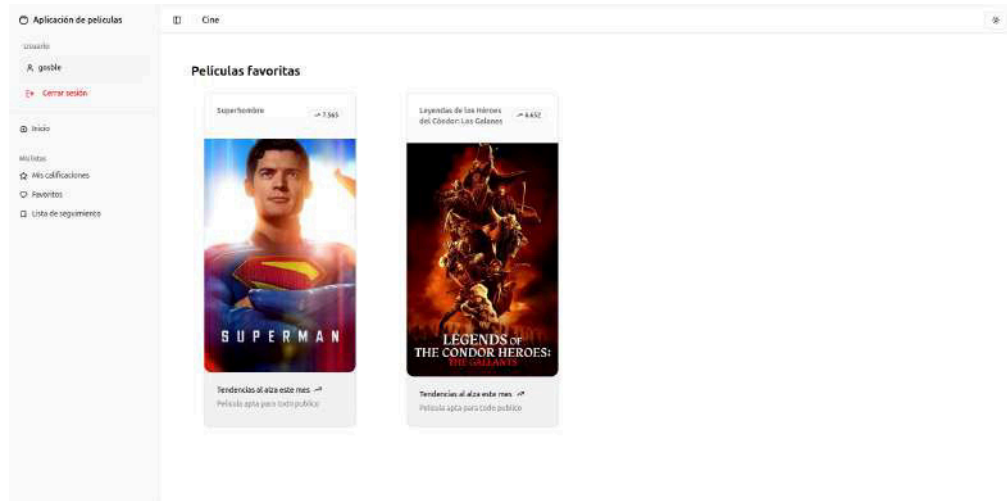


Figura 83 Página de películas favoritas

- **FR005: Eliminar películas Favoritas**

El usuario puede eliminar de sus listas de favoritos usando el servicio TMDb según lo indica la Figura. En funcionalidad se visualiza en la Figura 84.

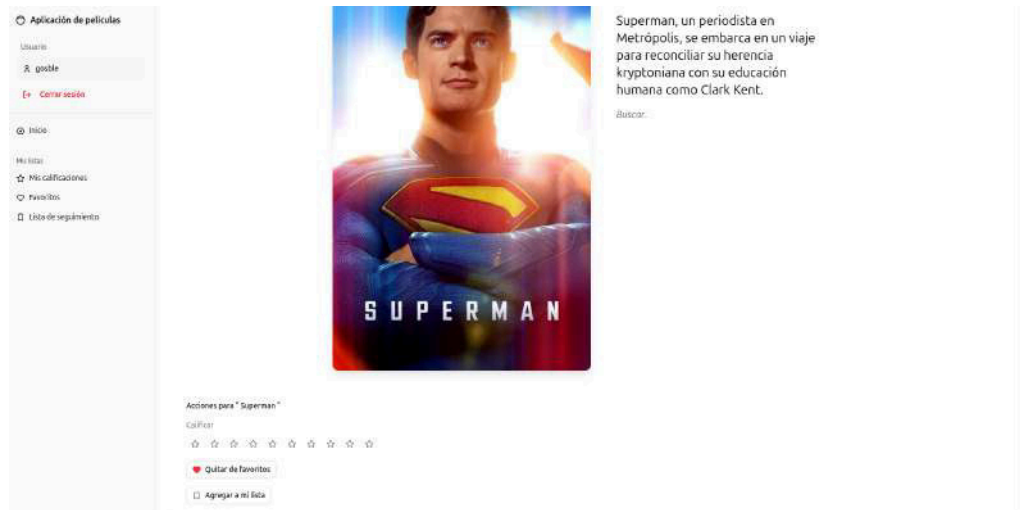


Figura 84 Eliminar las películas de mis listas

D. Desarrollo de la HU04 - Gestionar calificación de Películas

Para el desarrollo de la cuarta historia de usuario se tomaron los siguientes requerimientos.

- **FR006: Agregar una calificación a una película**

Si el usuario se loguea como invitado, miembro o en otra parte también como visitante podrá calificar y obtener su lista de calificaciones así como lo indica la Figura 85.



Figura 85 Calificación de la película

- **FR007: Actualizar una calificación a una película**

El usuario puede actualizar la calificación de una película usando el servicio TMDb. La funcionalidad se visualiza en la Figura 86.

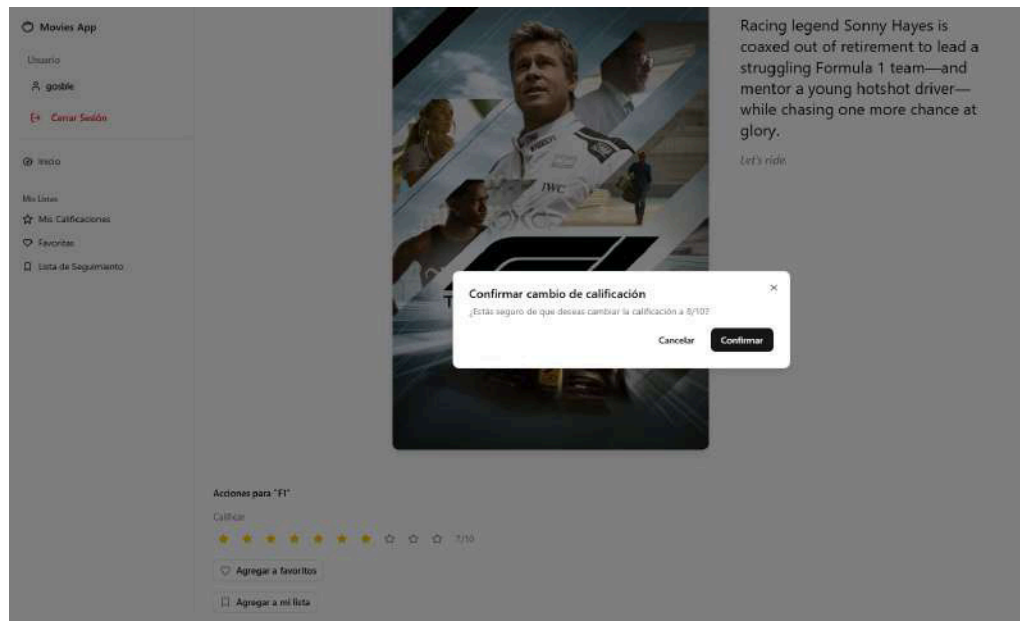


Figura 86 Actualizar calificación a una película

- **FR008: Eliminar una calificación a una película**

El usuario puede eliminar de sus listas de favoritos usando el servicio TMDb. La funcionalidad se visualiza en la Figura 87.

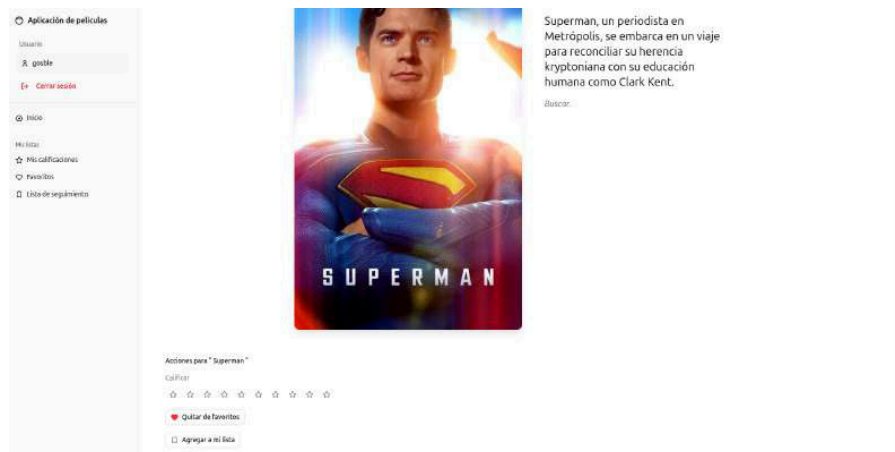


Figura 87 Eliminar la calificación de las películas de mis listas

4.3.2.4.2. Sprint 2

E. Desarrollo de la HU05 - Gestionar la lista de Seguimiento

Para el desarrollo de la quinta historia de usuario se tomó los siguientes requerimientos.

- **FR009: Agregar películas a mi lista de seguimiento**

El usuario puede agregar películas a su lista de seguimiento desde el detalle de la película. La funcionalidad se visualiza en la Figura 88.

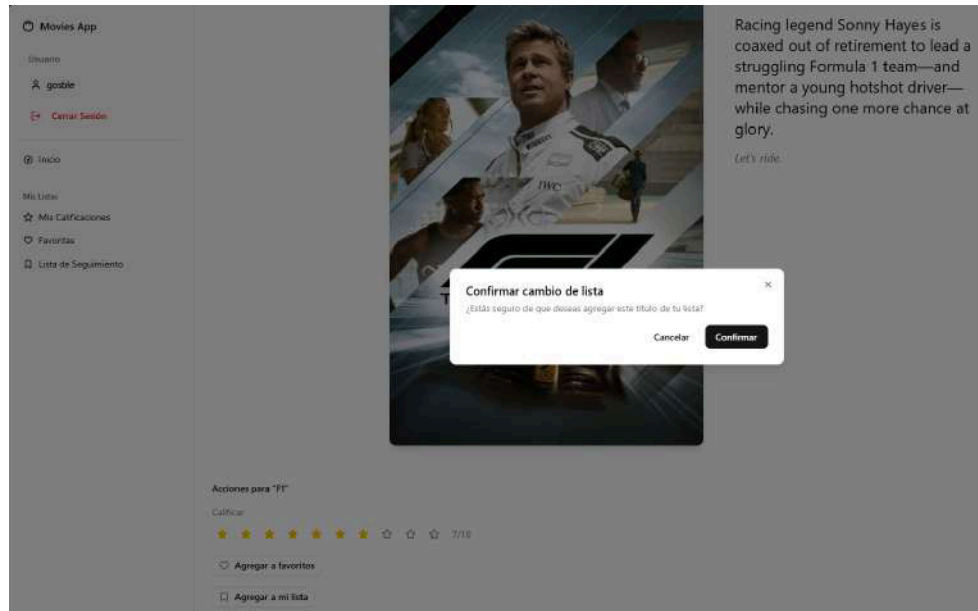


***Figura 88** Agregar las películas de mis listas*

- **FR010: Eliminar películas de mi lista de seguimiento**

El usuario puede agregar películas a su lista de seguimiento desde el detalle de la película. La funcionalidad se visualiza en la Figura 89.

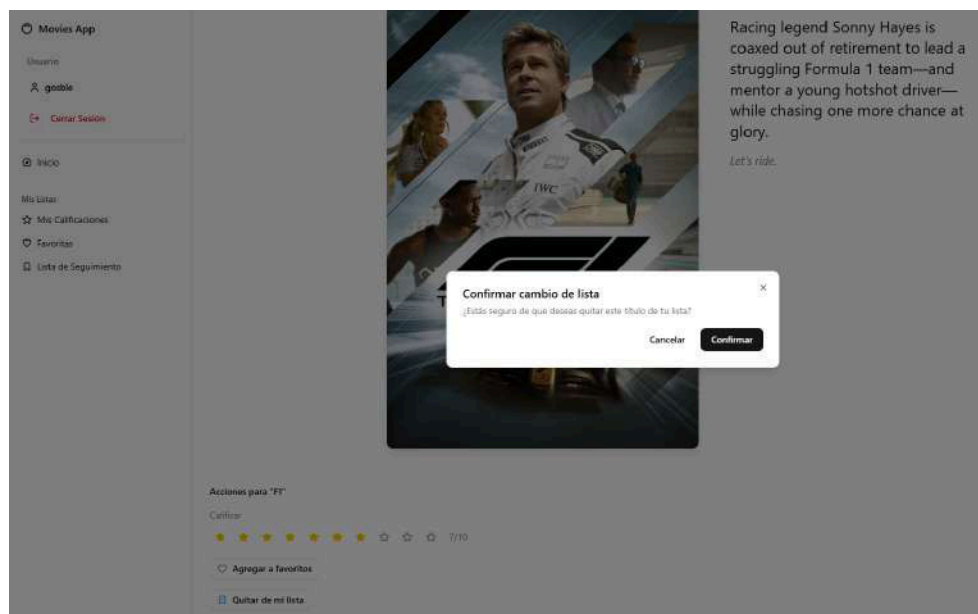


***Figura 89** Eliminar películas de listado de seguimiento*

F. Desarrollo de la HU06 - Gestionar la accesibilidad

Para el desarrollo de la sexta historia de usuario se tomaron los siguientes requerimientos.

- **FR011: Visualizar el modo oscuro**

La modalidad de accesibilidad está dentro de las dependencias internas con el fin de ser modificado de manera independiente y ser reutilizado por otros proyectos que usan la misma tecnología base que el proyecto presente (React), antes algún cambio de negocio se actualizó en un módulo independiente aislada del sistema y reutilizable por otros sistemas nuevos. El usuario puede cambiar al modo oscuro al presionar el icono de sol o de luna como lo muestra la Figura 90.

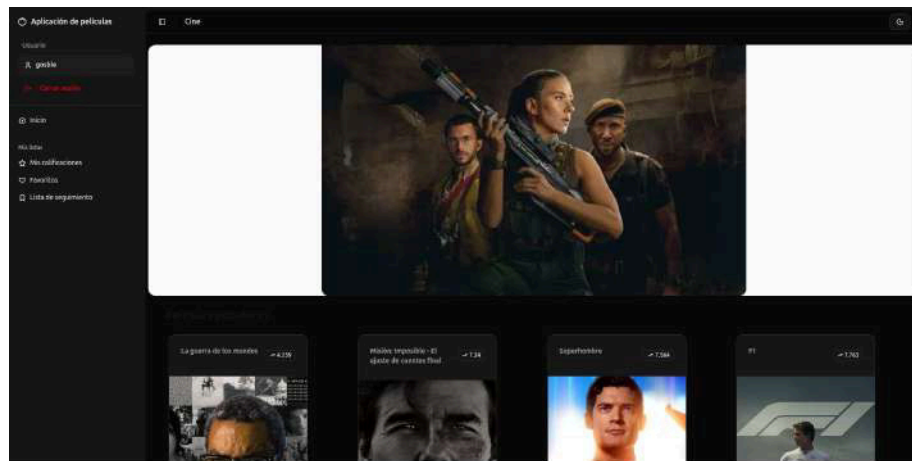


Figura 90 Modo oscuro

El usuario cuando ingresa a una película china puede obtener el título original como texto para el detalle de la página, así como lo indica la Figura 91.

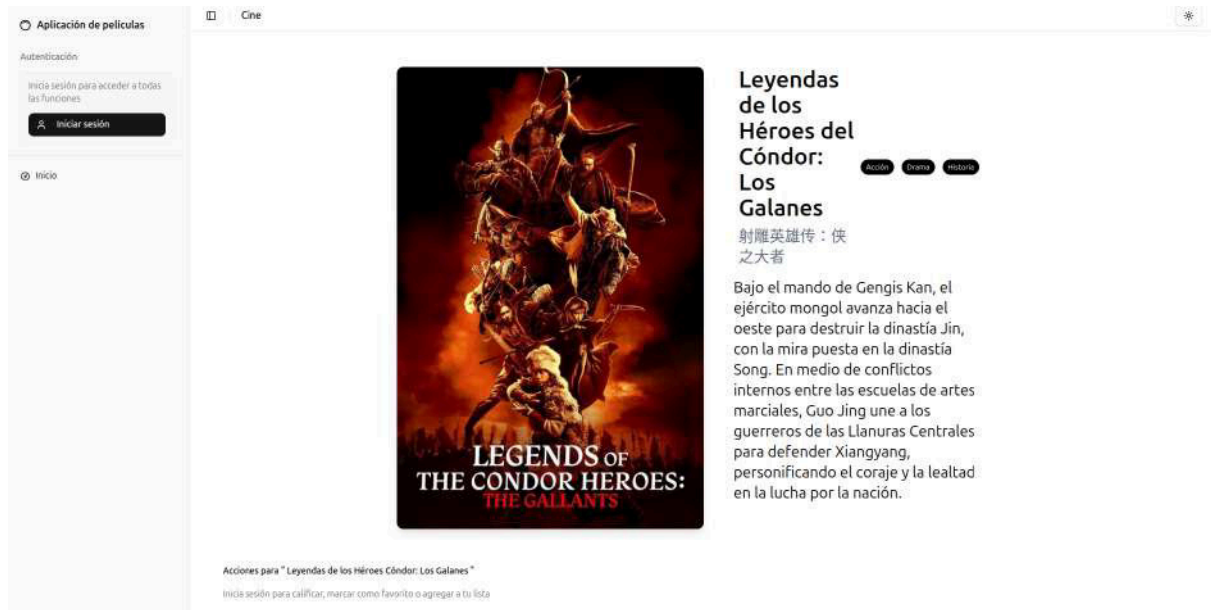


Figura 91 Detalle de una película china

4.2.3.5. Evaluación de funcionalidades del sistema

En la Tabla 12 se presentaron las funcionalidades del Sistema que fueron implementadas.

Tabla 12 Funcionalidades del sistema

Requerimiento	Efecto deseado	Estado	Observación
FR001	El sistema valida la existencia del usuario	Ok	Ninguna
FR002	El sistema logra mostrar la lista de películas y su detalle	Ok	Ninguna
FR003	El usuario se desliza por el carrusel y observa las películas	Ok	Solo existe 3 películas en el carrusel
FR004	El sistema proporciona la función de agregar películas Favoritas	Ok	El usuario debe estar logueado y ser miembro
FR005	El sistema permite eliminar películas Favoritas	Ok	El usuario debe estar logueado y ser miembro
FR006	El sistema proporciona la función de calificar una	Ok	El usuario debe estar logueado y ser

Requerimiento	Efecto deseado	Estado	Observación
	película		miembro
FR007	El sistema proporciona la función de actualizar la calificación	Ok	El usuario debe estar logueado y ser miembro
FR008	El sistema permite eliminar la calificación de la película	Ok	El usuario debe estar logueado y ser miembro
FR009	El sistema proporciona la función de agregar a su lista de seguimiento	Ok	El usuario debe estar logueado y ser miembro o visitante
FR010	El sistema permite eliminar una película de su lista de seguimiento.	Ok	El usuario debe estar logueado y ser miembro o visitante
FR011	El sistema permite usar el modo oscuro con el fin de cuidar la visión del usuario	Ok	Cualquier usuario puede hacer uso de esta funcionalidad

4.4. Validación de la propuesta

En esta base a las propuestas de (Mena & Santorum, 2021) se validó el sistema usando la norma ISO/IEC 25010 y la ISO/IEC 25022, en su artículo de investigación donde se identificó las subcaracterísticas, métricas y ecuaciones, mediante un enfoque cuantitativo que nos permitiera medir la mantenibilidad que tiene un sistema así como lo indicó la Figura 92 el cual muestra el proceso seguido para la medición de los sistemas. La evaluación fue realizada por dos desarrolladores con 3 años de experiencia en Frontend Engineering, quienes participaron activamente en el análisis y validación del sistema, las métricas de mantenibilidad, dichas métricas nos permitirían medir este proyecto diseñado con la Arquitectura propuesta basada en Clean Architecture de 3 capas, para luego ser comparado con otro de 4 capas mediante un enfoque de pruebas por escenarios y la evaluación comparativa del análisis estáticos de los datos del repositorio en definición.

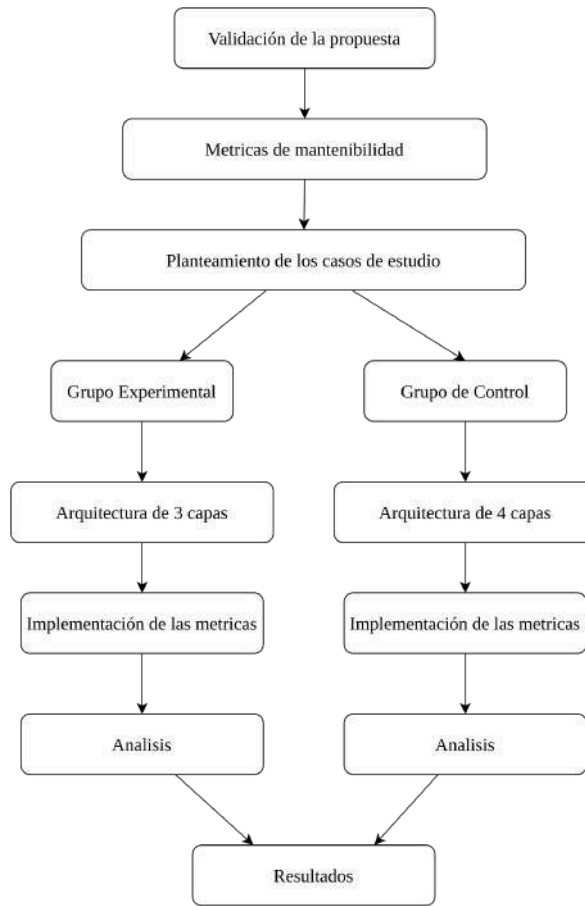


Figura 92 Proceso de Validación

4.4.1. Métricas de mantenibilidad

4.4.1.1. Métrica de la norma ISO/IEC 25010

La norma ISO/IEC 25010 está conformada por 8 características de calidad: adecuación funcional, eficiencia de desempeño, compatibilidad, capacidad de interacción, fiabilidad, seguridad, mantenibilidad y flexibilidad, así como lo indica (Cai et al., 2025).

La validación tiene como objetivo evaluar la mantenibilidad de la arquitectura con 3 capas propuesta usando la norma ISO 25010, donde se define la mantenibilidad que representa la capacidad de software para ser efectivo y se subdivide, como se visualiza la Tabla 13, la evaluación de las métricas ISO/IEC 25010 se usó las métricas descritas en el estándar ISO/IEC 25022 como lo indica (Quiña-Mera et al., 2022), con el fin de obtener las ecuaciones necesarias para evaluar la ISO/IEC 25010.

Tabla 13 Mantenibilidad y subcaracterísticas

ID	Definición	Sub característica
MT1	La capacidad de un producto para evitar que alteren los cambios a un componente	Modularidad
MT2	La capacidad de ser usado en más de una construcción de software	Reusabilidad
MT3	Es la facilidad con la que se puede medir el impacto sobre el software, diagnosticando posibles fallos	Analizabilidad
MT4	Es la capacidad del sistema de ser modificado de forma eficiente sin generar defecto de calidad	Capacidad para ser modificado
MT5	Es la facilidad de generar criterio de prueba para un sistema	Capacidad para ser probado

Tabla 14 Análisis de mantenibilidad del proyecto

Identificador	Definición	Estado
A	Datos de clasificación A	Excelente
B	Datos de clasificación B	Bueno
C	Datos de clasificación C	Normal
D	Datos de clasificación D	Malo
E	Datos de clasificación E	Paupérrimo

En la Tabla 14, se describe la clasificación de la mantenibilidad en un proyecto de manera gráfica, en un rango de A-E, tomando las mejores condiciones de mantenibilidad en la clasificación A y la peor condición en la clasificación E.

4.4.1.2. Métricas de la norma la ISO/IEC 25022

Según (Mena & Santorum, 2021), la evaluación de este modelo se definió a través de las siguientes métricas aplicadas en la Tabla 15 que fue parte de los criterios de analizabilidad, cambiabilidad, estabilidad, testabilidad, formando las métricas de mantenibilidad del sistema.

Tabla 15 Métricas de mantenibilidad

Métrica y ecuación	Descripción
MT1M1- Capacidad de condensación $X=A/B$	A = Número de componentes que no se ven afectado por cambio de otros componente B = Total de número de componentes específicos donde: $B > 0$. $0 \leq X \leq 1$; lo más cerca a 0, es mejor.

Métrica y ecuación	Descripción
MT1M2- Clases de acoplamiento $X=A$	A = Número de relaciones que tiene una función respecto a otras clases. $1 \leq X \leq 4$; Lo más cercano a 1, es mejor.
MT2M1- Ejecución de reutilización $X = A/B$	A = Número de elementos reutilizados; B = Total número de ítems en la librería reutilizable; donde: $B > 0$ $0 \leq X \leq 1$; Lo más cercano a 1, es el mejor. $0 \leq X \leq 1$; Lo más cercano a 1 es mejor.
MT3M1- Capacidad de seguimiento de auditoría $X = A/B$	A = Número de datos realmente registrados durante operación; B = Número de datos esperados a ser grabado para controlar el estado del sistema durante los operarios. Lo más cercano a 1, es mejor.
MT3M2-Diagnóstico de funciones suficientes $X = A/B$	A = Número de funciones diagnosticadas implementadas ; B = Número de funciones de diagnóstico requerido en la especificación de requisitos ($B > 0$). $0 \leq X \leq 1$; Lo más cercano a 1, es el mejor.
MT4M1- Complejidad ciclomática $X = A + 1$	A = Número de instrucciones condicionales que una función tiene $1 \leq X < 15$; Lo más cercano a 1, es el mejor.
MT4M2- Profundidad de herencia $X=A$	A = Número de jerarquías utilizadas para un determinada función $0 \leq X \leq 4$; Lo más cercano a 0, es el mejor.
MT4M3- Localización de corrección de impacto grado $X = A/B$	A = Número de fallos que aparecen después de un fallo ha sido resuelto; B = Número de averías solucionadas. ($B > 0$) $0 \leq X \leq 1$; Lo más cercano a 0, es el mejor.
MT4M4- Modification complexity $X = A/T$	A = Número de modificaciones; B = tiempo de trabajo eso le lleva al desarrollador modificarlo. ($T > 0$) $X = A/T$ Lo más alejado de 0/t es el mejor
MT4M5- Tasa de éxito de la modificación $X = A/B$	A = Número de problemas dentro de un período determinado antes del mantenimiento B = Número de problemas en el mismo período después del mantenimiento ($B > 0$)
MT5M1- Functional complexity of test functions $X = A/B$	$X \leq 1$; Lo más cercano a 0, es el mejor. A = Número de funciones de prueba implementadas; B = Número de funciones de prueba requeridas. ($B > 0$) $0 \leq X \leq 1$; Lo más cercano a 1, es el mejor.
MT5M2- Capacidad de prueba autónoma $X = A/B$	A = Número de pruebas que dependen de otros sistemas; B = Número total de pruebas dependiendo de otros sistemas. ($B > 0$) $0 \leq X \leq 1$; Lo más cercano a 0, es el mejor.
MT5M3- Capacidad de reinicio de prueba $X = A/B$	A = Número de casos en los que el mantenedor puede pausar y restaurar la prueba; B = Número de pausa casos en las pruebas de ejecución. ($B > 0$)

Las siguientes métricas se pueden medir según las 12 ecuaciones descritas:

Capacidad de condensación: Es el número de componentes que no se ven afectados entre el número total de componentes específicos:

$$(1) \text{ Capacidad de condensación} = \frac{\text{Número de componentes que no se ven afectados por cambio de otros componente}}{\text{Total de número de componentes específicos}}$$

Clases de acoplamiento: Es el número de relaciones que tiene una función respecto a otras clases.

$$(2) \text{ Clases de acoplamiento} = \text{Número de relaciones que tiene una función respecto a otras clases}$$

Ejecución de reutilización: Es el número de elementos reutilizados entre el total de ítems en la librería reutilizable.

$$(3) \text{ Ejecución de reutilización} = \frac{\text{Número de elementos reutilizados}}{\text{total de ítems en la librería reutilizable}}$$

Capacidad de seguimiento de auditoría: Es el número de datos realmente registrados durante operación entre el número de datos esperados a ser grabado para controlar el estado del sistema durante los operarios.

$$(4) \text{ Capacidad de Seguimiento} = \frac{\text{Número de datos realmente registrados durante operación}}{\text{Número de datos esperados a ser grabado para controlar el estado del sistema durante los operarios}}$$

Diagnóstico de funciones suficiente: Es el número de funciones diagnosticadas implementadas entre el número de funciones de diagnóstico requerido en la especificación de requisitos.

$$(5) \text{ Diagnóstico de funciones suficiente} = \frac{\text{Número de funciones diagnosticadas implementadas}}{\text{Número de funciones de diagnóstico requerido en la especificación de requisitos.}}$$

Complejidad Ciclomática: Es el número de instrucciones condicionales que una función tiene más una unidad.

$$(6) \text{ Complejidad Ciclomática} = \text{Número de instrucciones condicionales} + 1$$

$$(6.1) \text{ Número de instrucciones condicionales} = \frac{\text{Suma de la complejidad ciclomática de cada archivo}}{\text{Número total de funciones}}$$

Profundidad de herencia: Es el número de jerarquía utilizada para una determinada función.

$$(7) \text{ Profundidad de herencia} = \text{Número de jerarquía utilizada para una determinada función.}$$

Localización de corrección de impacto grado: Es el número de fallos que aparecen después de un fallo ha sido resuelto entre el número de averías solucionadas.

$$(8) \text{ Localización de corrección de impacto grado} = \frac{\text{Número de fallos que aparecen después de un fallo ha sido resuelto}}{\text{Número de averías solucionadas.}}$$

Modificación de la complejidad: Es el número de modificaciones entre el tiempo de trabajo que le lleva al desarrollador modificarlo.

$$(9) \text{ Modificación de la complejidad} = \frac{\text{Número de modificaciones}}{\text{Tiempo de trabajo que le lleva al desarrollador modificarlo.}}$$

Tasa de éxito de la modificación: Es el número de problemas dentro de un período determinado antes del mantenimiento entre el número de problemas en el mismo periodo después del mantenimiento.

$$(10) \text{ Tasa de éxito de la modificación} = \frac{\text{Número de problemas dentro de un período determinado antes del mantenimiento}}{\text{Número de problemas en el mismo periodo después del mantenimiento..}}$$

Complejidad funcional de las funciones de testeo: Es el número de funciones de prueba implementadas entre el número de funciones de prueba requeridas.

$$(10) \text{ Complejidad funcional de testeo} = \frac{\text{Número de funciones de prueba implementadas}}{\text{Número de funciones de prueba requeridas.}}$$

Capacidad de prueba autónoma: Es el número de pruebas que dependen de otros sistemas entre el número total de pruebas dependiendo de otros sistemas.

$$(11) \text{ Capacidad de prueba autónoma} = \frac{\text{Número de pruebas que dependen de otros sistemas}}{\text{Número total de pruebas dependiendo de otros sistemas.}}$$

Capacidad de reinicio de prueba: Es el número de casos en los que el mantenedor puede pausar y restaurar la prueba entre el número de pausa de casos en las pruebas de ejecución.

$$(12) \text{ Capacidad de reinicio de prueba} = \frac{\text{Número de casos en los que el mantenedor puede pausar y restaurar la prueba}}{\text{Número de pausa de casos en las pruebas de ejecución.}}$$

4.4.2. Planteamiento de los de casos de estudio

Para la validación de la arquitectura, se realizó la comparación entre los grupos experimental y de control usando Clean Architecture basada en 3 capas y 4 capas respectivamente.

4.4.2.1. Dataset

Para la validación de la arquitectura, se realizó un análisis de código estático a partir del repositorio <https://github.com/tesis-SanMarcos/MOVIES>, el uso de este repositorio y no uno de código abierto externo es imprescindible, ya que esta implementación es la evidencia empírica de la aplicación de la propuesta arquitectónica tal como señala García et al. (2021), la recuperación y el análisis de la arquitectura desde su implementación son pasos esenciales en la fase de mantenimiento para detectar el deterioro arquitectónico. La Figura 93 indica las características del repositorio.

```
github.com/AlDanial/cloc v 1.98 T=0.15 s (945.7 files/s, 181523.3 lines/s)
```

Language	files	blank	comment	code
JSON	9	4	0	12775
TypeScript	113	1466	150	11855
JavaScript	10	163	249	656
CSS	2	7	0	132
HTML	1	0	0	13
Text	1	0	0	6
Markdown	1	1	0	2
SVG	2	0	0	2
SUM:	139	1641	399	24641

Figura 93 Características del repositorio

4.4.2.1. Estructura de los registros

El dataset mantiene la siguiente estructura dentro del repositorio:

- Files: Número de archivos del proyecto
- Blank: Número de líneas en blanco
- Líneas: Número de líneas de comentarios
- Archivos: Número de líneas de código usadas

4.4.2.2. Obtención y filtrado de datos

Para la obtención de datos se usa la url proporcionada que contiene la información del proyecto y su data estáticos, para aligerar el preprocesamiento no se analizaron el coverage y code smells para la capa más externa, debido a que compartimos la misma idea de construcción con (Cecil, R., 2017), y también solo se realizó el análisis para la carpeta source ya que aquí se encuentra el código de calidad dentro de nuestro repositorio. La Figura 94 y 95 representa el preprocesamiento de los datos del repositorio

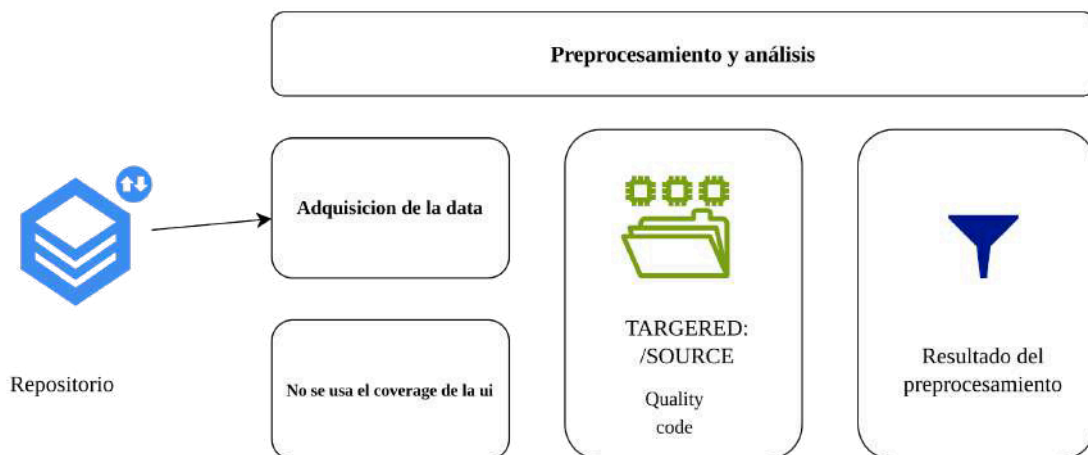


Figura 94 Diagrama de preprocesamiento de los datos estáticos

```
github.com/AlDanial/cloc v 1.98 T=0.10 s (864.7 files/s, 80998.0 lines/s)
```

Language	files	blank	comment	code
TypeScript	81	669	24	7035
CSS	2	7	0	132
SVG	1	0	0	1
SUM:	84	676	24	7168

Figura 95 Característica del repositorio luego del filtro

4.4.2.3. Transformación de los datos de entrada

Para la obtención de datos es necesario un tratamiento con un transformador que depende de cada uso, para ello usamos SonarQube que lee reportes de código estático, Dependency-cruiser generar un json que nos permitió analizar en forma de grafo los módulos de nuestro proyecto y en caso se necesite un análisis estático específico se creó también un JSON usando un análisis de Script automatizado así como se muestra en la Tabla 16 y en la selección en la Figura 96.

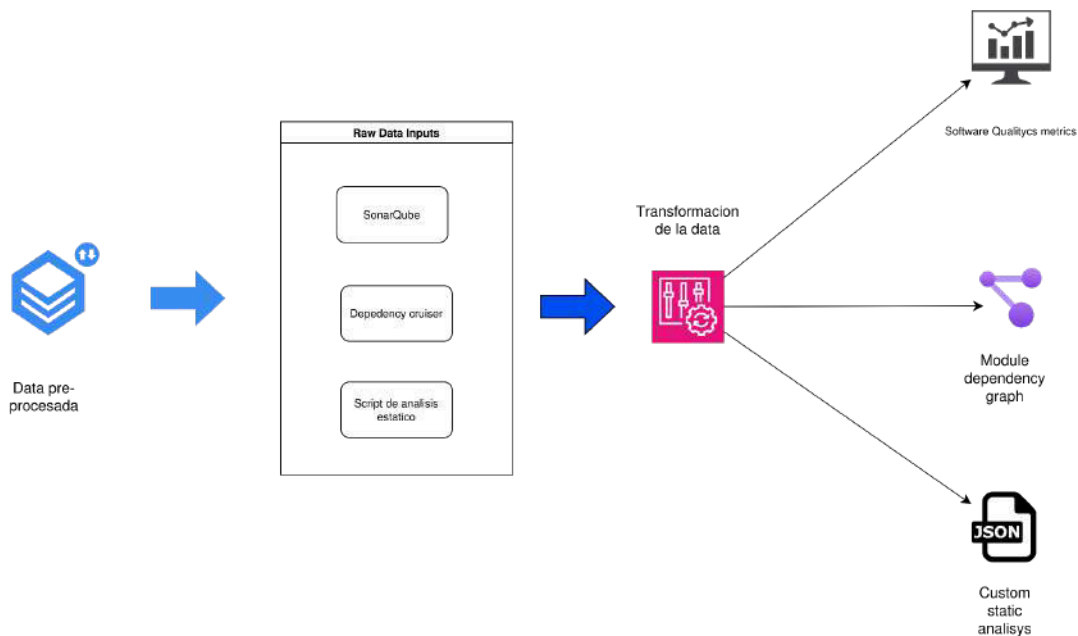


Figura 96 Diagrama de la transformación de los datos estáticos

Tabla 16 Análisis de las herramientas de código estático

Aspectos	SonarQube	Dependency-cruiser	Análisis de Script Automático
Importancia	Garantiza calidad usando estándares de codificación y ayuda mantener deuda técnica a lo largo del tiempo	Análisis de dependencia en la arquitectura de un proyecto, verifica las capas y dependencias circulares	Permite revisar código como commit o pull, lo que permite encontrar errores o fallos que se puede necesitar antes casos más complejos
Salida	Panel de control y puntajes de calidad	Gráficos de dependencias y reportes e arquitectura de proyecto	Reportes de problemas de patrones específicos
Cuando usarlo	En el ciclo de integración de desarrollo al finalizar un cambio	Durante la fase de diseño de arquitectura y para validar que no se rompan reglas estructurales	Se puede cuando es necesarios pruebas específicas o cambios nuevos no vistos por otros desarrolladores

4.4.2.4. Casos de estudios

La experimentación se basó sobre 4 estados evolutivos de la base de código, estos estados buscan aislar el impacto de 2 componentes claves como se indica en la Tabla 17

- **Módulo de abstracción de peticiones HTTP**

Gestiona la complejidad de la comunicación externa mediante patrones de inyección de dependencias

- **Módulo de accesibilidad**

Gestiona la complejidad del Front-end y la experiencia de usuario incluye temas

La combinación de estos módulos genera una matriz de 4 casos de estudio que permiten medir la complejidad estructural incremental.

Tabla 17 Descripción de los casos de estudio

Caso	Módulo de Abstracción	Módulo de accesibilidad	Descripción
Caso 1	No	No	Caso mínimo o línea base
Caso 2	Si	No	Impacto de Abstracción en los método HTTP
Caso 3	No	Si	Impacto de accesibilidad
Caso 4	Si	Si	Caso Completo, representa la complejidad final y total del sistema

4.4.2.4.1. Grupo experimental

Se define por la aplicación de la propuesta en la tesis, la cual se usó el principio de dependencia unidireccional de capas externas a las internas: Capa UI, Capa de casos de uso y Capa entidad.

- **Caso de estudio 1**

Este es el caso más simple donde no existe el módulo de puertos por que no hubo abstracción para este caso, de esta manera solo se implementa la arquitectura de 3 capas y la carga que debería recibir por las entidades o otras constante son esparcidas por sus capas de interés como la entidad y sub módulo de código estático como *lib* así como se indica en la Figura 97.

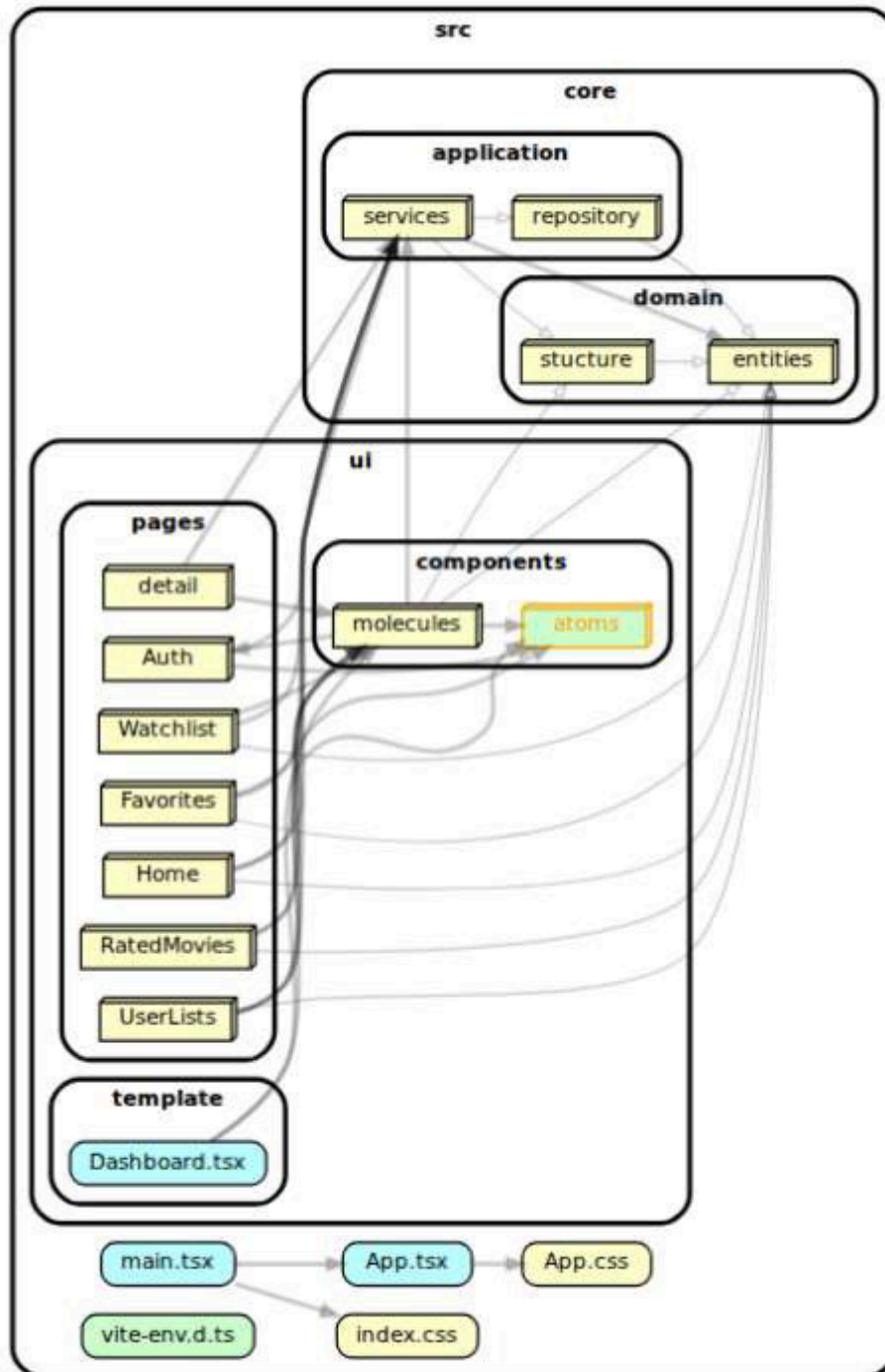


Figura 97 Diagrama del caso 1 del grupo experimental

- **Caso de estudio 2**

En este caso se usa el módulo de puerto llevando la abstracción del protocolo de comunicación HTTP como parte pública debido a la librería Axios que es independiente a nuestro desarrollo, por tal motivo, en caso de usar otro dependencia de comunicación como Ki se puede modificar desde su feature independiente sin

afectación al proyecto pero eso dependerá de las necesidades de desarrollo cual dependencias es mejor para el proyecto como se indica en la Figura 98.

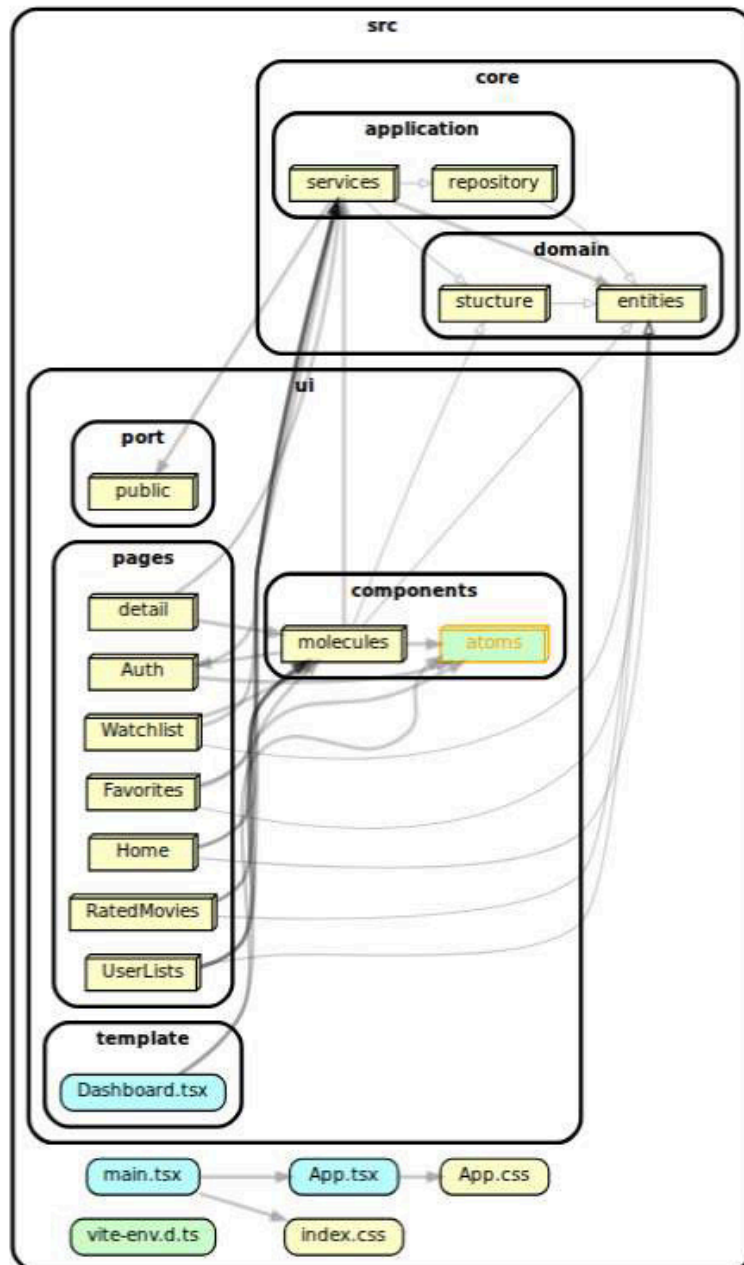


Figura 98 Diagrama del caso 2 del grupo experimental

- **Caso de estudio 3**

En este caso se usa el módulo de puerto llevando la abstracción de la accesibilidad, siendo mas específico la accesibilidad, este caso conlleva más complejidad, por este caso se decide llevarlo como parte privada debido que usamos uso de la tecnología

base el Framework o librería (React) usando el Hooks logramos crear nuestra feature de temas exportable a cualquier proyecto esto gracias al patrón Bridge como se indica en la Figura 99.

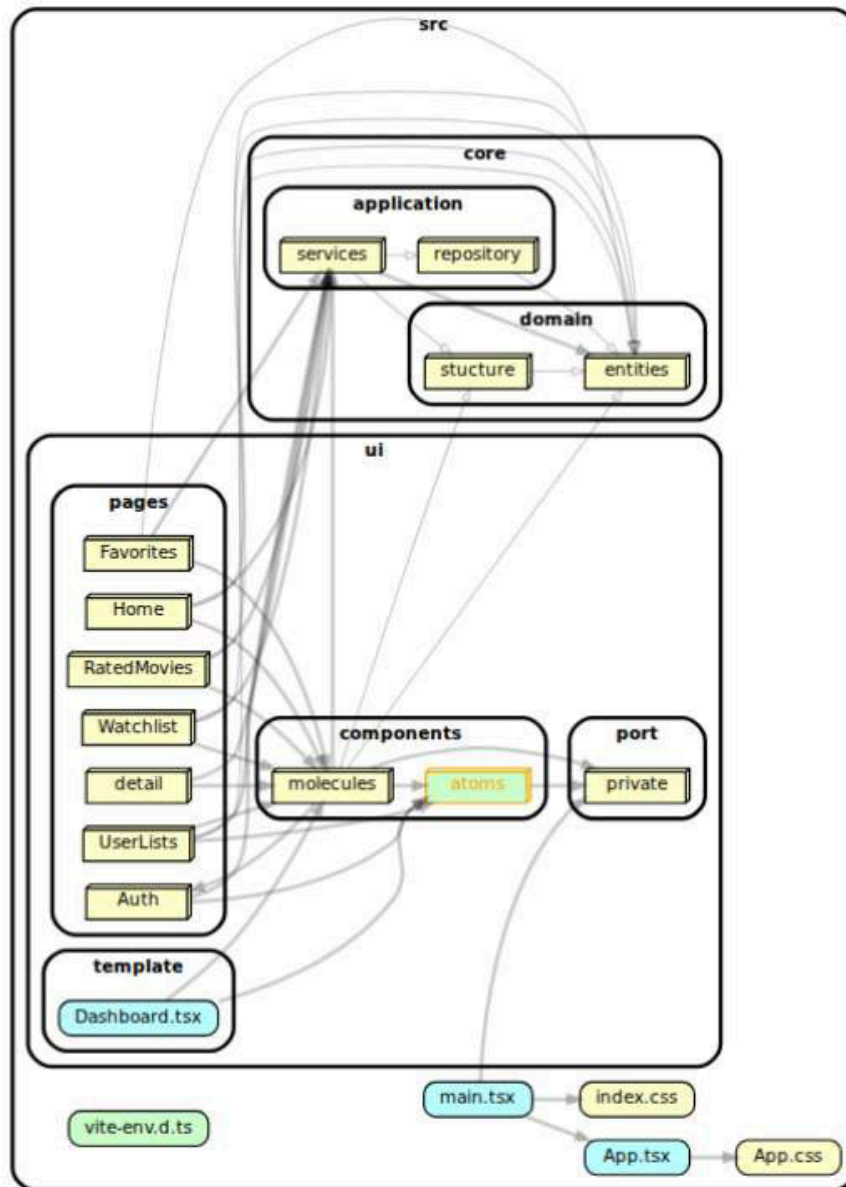


Figura 99 Diagrama del caso 3 del grupo experimental

- Caso de estudio 4

Este caso es el más complejo, ya que implementa los 2 modelos mencionados usando el uso de puertos tanto público como privado esto representa la abstracción total de los casos mencionados dentro del sistema así como se muestra en la Figura 100.

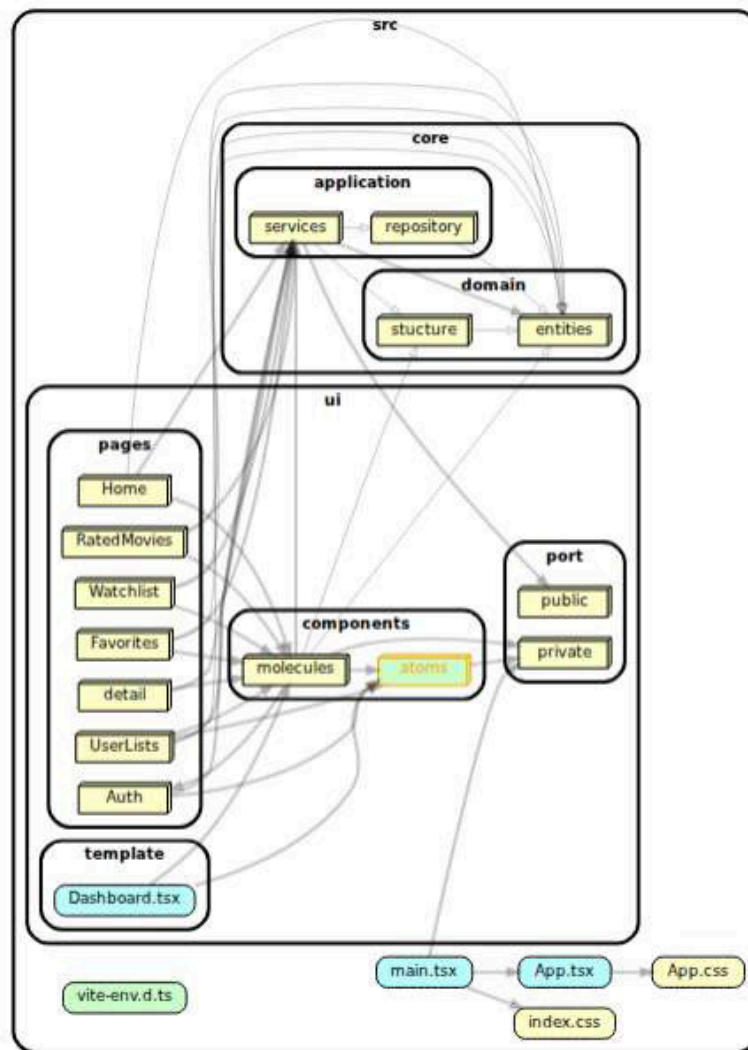


Figura 100 Diagrama del caso 4 del grupo experimental

4.4.2.4.2. Grupo de control

Este diseño se define por el diseño clásico de la Clean Architecture estructurado en cuatro capas concéntricas que garantiza la lógica de negocio: Capa de UI, Capa Adaptadora, Capa de Casos de Uso y la Capa Entidad.

- **Caso de estudio 1**

Este es el caso más simple donde la Capa Adaptadora la encargada de cualquier adaptación no tiene carga por parte de módulos de accesibilidad ni la del protocolo de comunicación, la carga de constantes como código estático o adaptación de entidades esto debido a su arquitectura de 4 capas como se indica en la Figura 101.

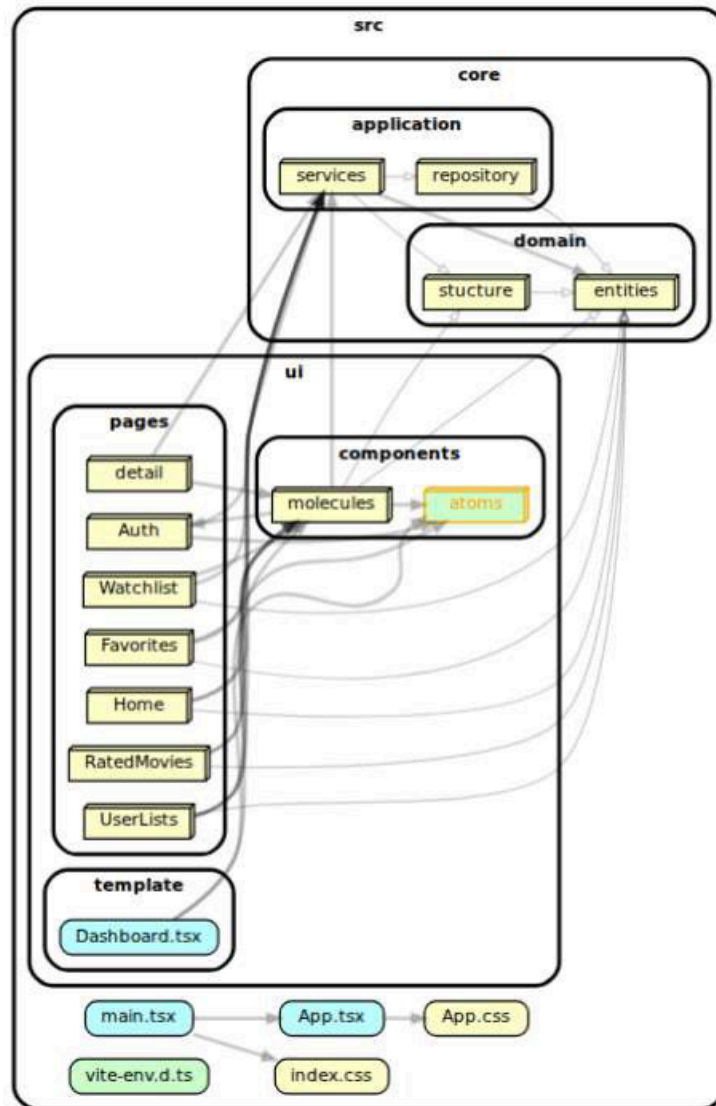


Figura 101 Diagrama del caso 1 del grupo control

- **Caso de estudio 2**

En este caso se usa el módulo abstracción del protocolo de comunicación HTTP como parte de la Capa Adaptadora, por tal motivo, en caso de usar otro dependencia de comunicación como Ki se puede modificar desde la capa de adaptación pero esta funcionalidad siempre dependera en este proyecto y no podría ser usada en otros proyecto aumentando duplicación de código así como se muestra la Figura 102.

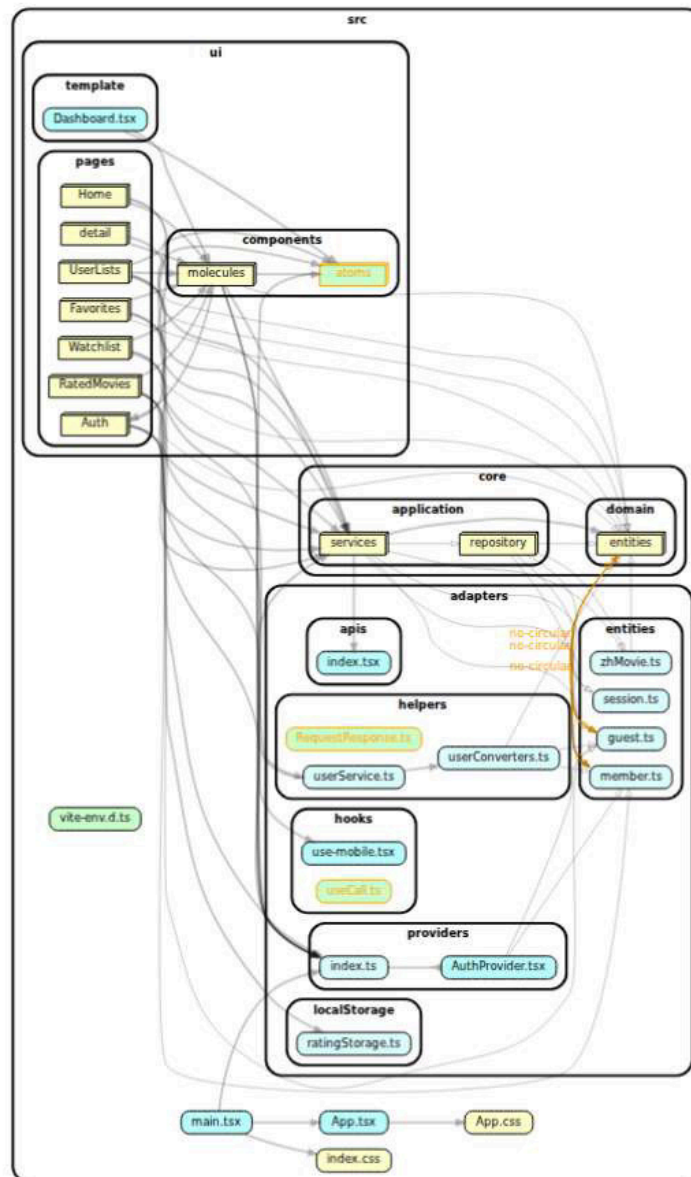


Figura 102 Diagrama del caso 2 del grupo control

- **Caso de estudio 3**

En este caso se usa el módulo de abstracción de la accesibilidad, siendo más específico los temas, este caso conlleva más complejidad, por ello se usó Hooks para su implementación y se encuentra en la capa de adaptación así como se muestra en la Figura 103

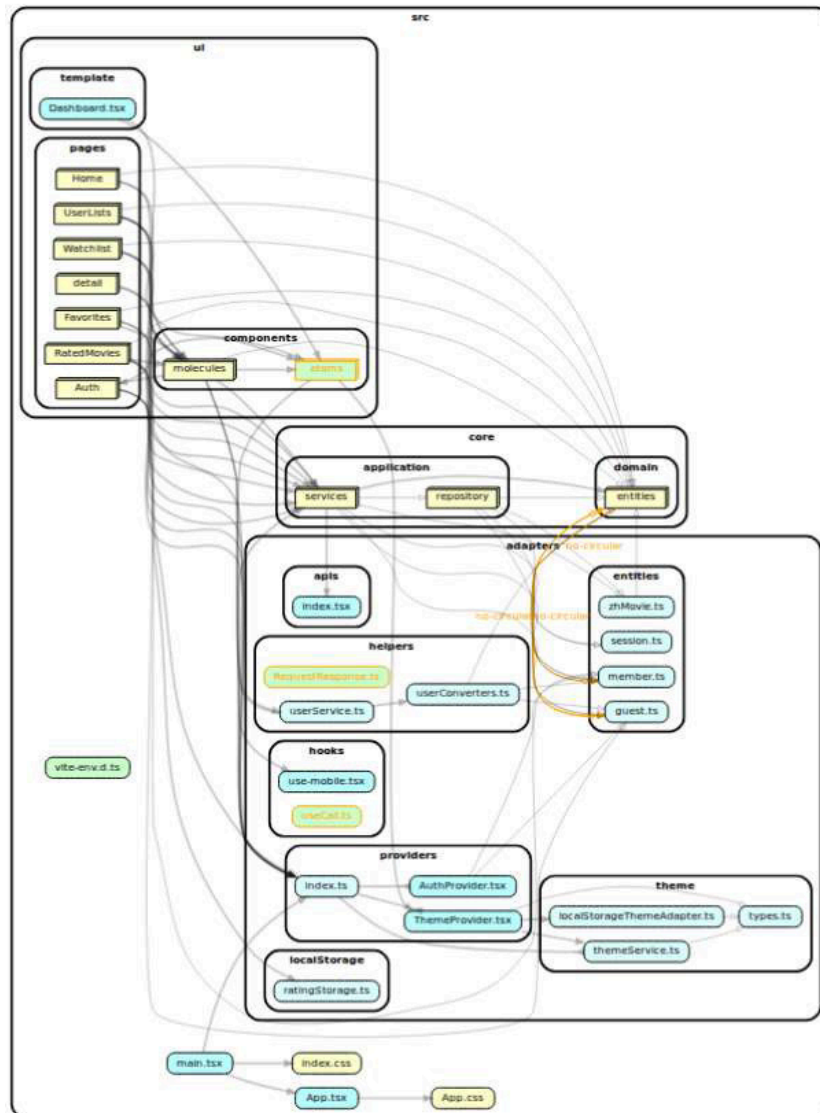


Figura 103 Diagrama del caso 3 del grupo control

- **Caso de estudio 4**

Este caso es el más complejo, ya que implementa los 2 modelos mencionados usando el uso de la Capa Adaptadora esto representa la abstracción total de los casos mencionados dentro del sistema en la arquitectura de 4 capas así como se indica en la Figura 104.

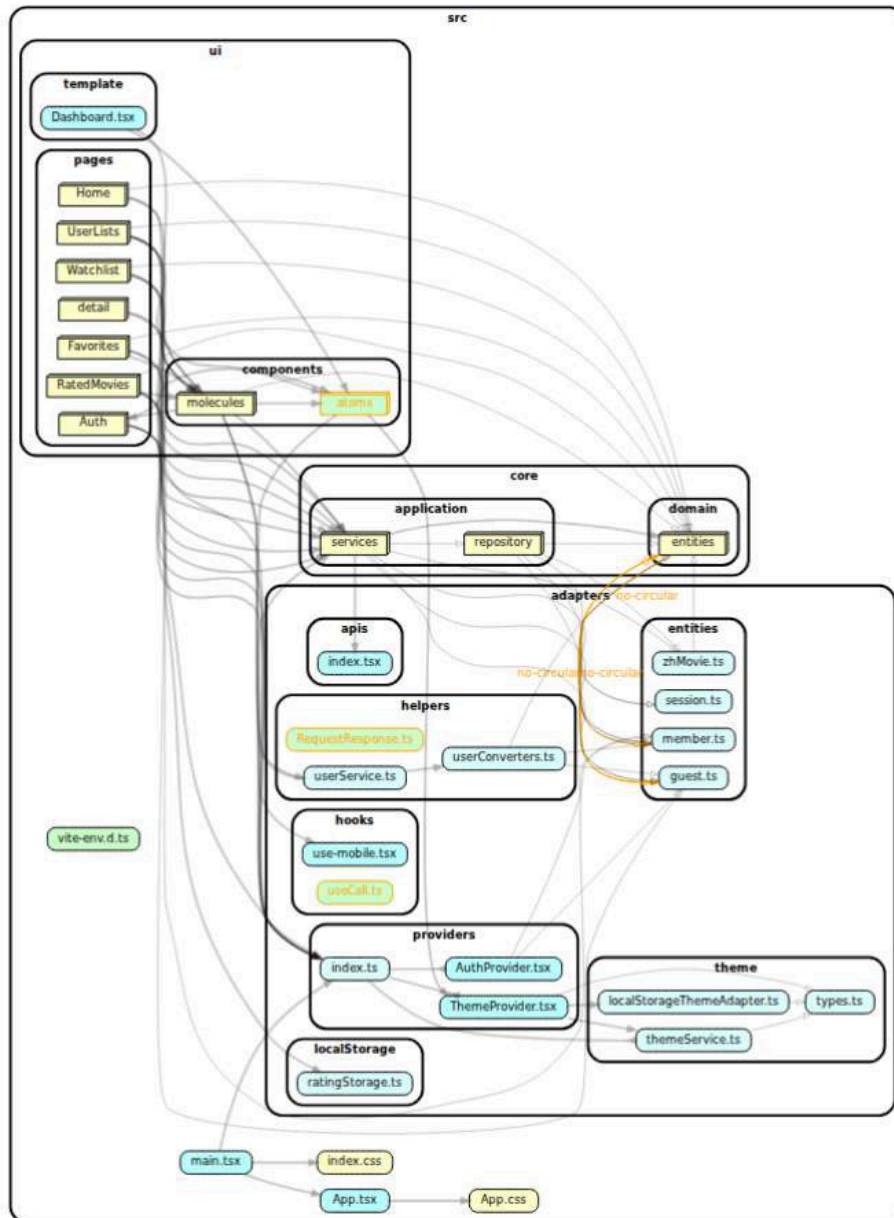


Figura 104 Diagrama del caso 4 del grupo control

4.4.2.5. Limitación

En el presente trabajo se decidió no abordar las métricas MT4M3 debido a que es necesario una herramienta que mida constantemente el desarrollo del app cuando ocurra fallos, MT4M5, se necesitaría un desarrollo previo para medir el número de code smells antes de la mantenibilidad y la MT5M2 y MT5M3 tener prueba externa u de tipo servidor y comenzar en donde se detuvo.

Además solo se planteó 4 casos por grupo de estudio con 2 módulos de abstracción, debido solo por fines prácticos y de investigación.

4.4.3. Implementación de las Métricas

Se ha usado la herramienta de SonarQube, Script Análisis estático, análisis estático a través de grafos de dependencias como también lo sugiere (Falco, Scott & Robiolo, 2020) para un análisis de código estático y poder facilitar la evaluación de la ISO/IEC 25010 y la ISO/IEC 25022 ya que nos permitir un análisis automatizado de calidad, además de inspeccionar el código y vulnerabilidades en nuestro sistema, el proceso de análisis se muestra en la Figura 105 lo que representa la alimentación del código estático a través de un repositorio y el análisis mediante las herramientas previamente descritas y por último el cálculo de las métricas para la mantenibilidad del Software.

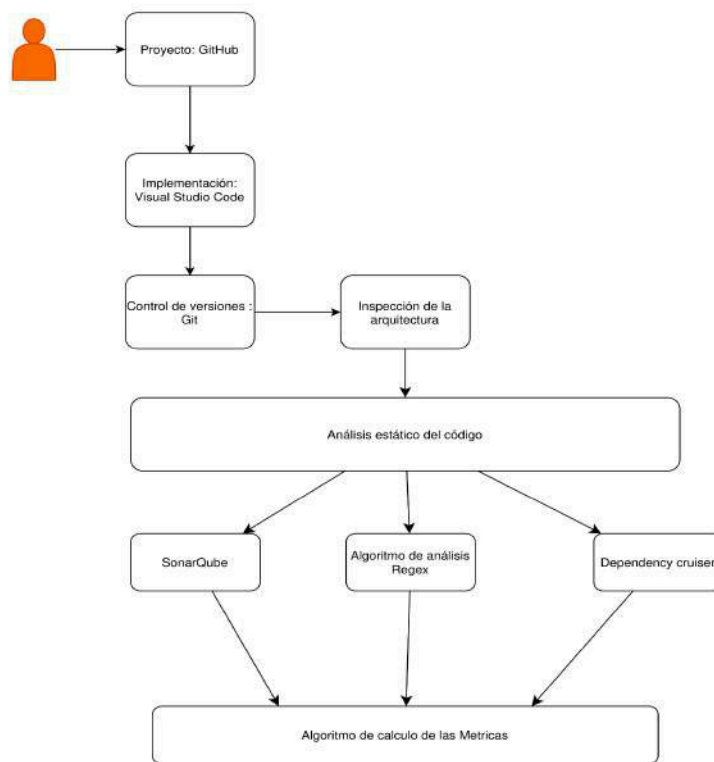


Figura 105 Proceso de análisis usando SonarQube

En la siguiente Tabla 18 se explica la relación de las métricas con las herramientas de código estático para su posterior análisis.

Tabla 18 Transformación de los datos de entrada para su posterior análisis

Métricas	SonarQube	Dependency-cruiser	Análisis Automático manual	Descripción
MT1M1	No	Si	No	Se debe medir la condensación del proyecto por lo que va relacionado a la capa estructural del proyecto.
MT1M2	No	Si	No	Se hace un análisis a los módulos y acoplamiento de en las relaciones por ello se usó grafos
MT2M3	No	Si	No	Se define los módulo de reutilización dentro de la estructura del proyecto por lo que se decide seguir usando grafos
MT3M1	No	Si	No	Se define los los módulos de auditoría y sus palabras claves, como se define temas estructurales se decide seguir usando grafos.
MT3M2	No	Si	No	Se define las funciones a diagnosticar y los módulos donde se realiza lógica de diagnóstico por lo que representa una parte estructural por eso se decir usar grafos
MT4M1	Si	No	No	Debido que Sonar analiza la complejidad se decidió usar los métodos de <i>complexity</i> y obtener la complejidad en nuestro proyecto
MT4M2	No	No	Si	Debido a que nuestro proyecto fue creado usando TypeScript las herramientas de análisis no estuvieron preparadas por eso se

				decidió usar Regex mediante un script para analizar las clases del proyecto debido que es un caso específico.
MT4M4	No	No	Si	Como se debía implementar las modificaciones se creó un Script para obtener nuestro historial de cambios de Github ya que es un caso específico.
MT5M1	Si	No	No	Se decidió usar el reporte del coverage y usar el SonarQube para el análisis estático de nuestro reporte

- **SonarQube**

Estas herramientas nos ayudaron a inspeccionar nuestro código fuente con el fin de detectar los indicadores de problemas de mantenibilidad como se indica en la Figura 106.

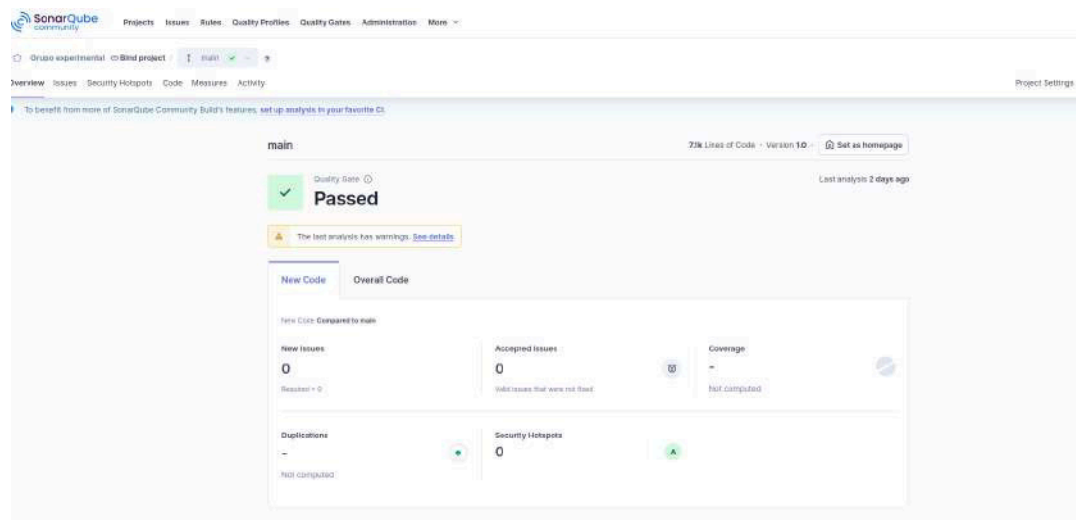


Figura 106 Analizador de código estático SonarQube

- **Dependency cruiser**

La implementación de Dependency cruiser asegura que la métrica se extrae del código fuente, eliminando todo rastro de subjetividad, el proceso depende de obtener

el grafo de dependencias *deps.json* como en la Tabla 19. Para su implementación se instaló: `npx depcruise --include-only "^src" --output-type json src > deps.json`

Tabla 19 Desglose de los parámetros de entrada para *dependency cruiser*

Características	Descripción
depcruise	Se selecciona Dependency Cruiser por su análisis estático identificando import y requiere para construir un grafo dirigido.
--include-only "src"	Se filtra el directorio src para asegurar que el análisis se centre en los componentes del sistema bajo estudio, excluyendo librerías de terceros.
src	Directorio raíz
--output-type json	Es el resultado del análisis de dependencias y se exporta en formato JSON, posteriormente consumida por el cálculo de la ecuación 1; $X=A/B$
> deps.json	Redirige la salida JSON al archivo deps.json y se convierte en los datos verificables

- **Análisis de Script automatico**

La implementación de Script de análisis se da sólo en casos específicos como en las métricas MT4M2 y MT4M4 debido a que las otras herramientas carecieron de soporte para typescript del caso específico mencionado como la evaluación de clases en TypeScript y el historial de commits respectivamente.

4.4.3.1. Capacidad de condensación MT1M1

La figura representa la métrica MT1M1, definida como $X=A/B$, mediante un script de Node.js como en la Figura 107 que procesa el grafo de dependencias obtenido del análisis estático *deps.json*. La variable B representa el total de componentes como *modules.length*, representando el universo del sistema la variable A componentes sin afectación representa los módulos que no aparecen como destino de ninguna dependencia como *!depsTo.include(m)*, son los módulos que no son usados por otros y por consiguiente no son afectados

```

import fs from 'fs';

const raw = fs.readFileSync('./deps.json', 'utf-8');
const deps = JSON.parse(raw);

const modules = deps.modules.map(m => m.source);
const depsTo = deps.modules.flatMap(m => m.dependencies.map(d => d.resolved));

const B = modules.length;
const A = modules.filter(m => !depsTo.includes(m)).length;
const X = B > 0 ? +(A / B).toFixed(3) : 0;

console.log('Variable A:', A);
console.log('Variable B:', B);

console.log('Reporte generado con X =', X);

```

Figura 107 Script de la métrica MT1M1

4.4.3.2. Acoplamiento MT1M2

La Tabla 19 nos da detalle del comando usado, luego se usó el siguiente Script como se observa en la Figura 108 que representa la métrica MT1M2, definida como $X=A$, donde A fue representado como el total de relaciones *dependencies* entre el total de módulos *modules.length*, tomando en referencia a la símil que representa los módulos con las clases en este caso por usar la tecnología React con typescript. Para este caso el script lee el file *deeps.json* donde guardamos en una Variable el universo de componentes utilizados como *deps.modules* y en otra variable las relaciones totales, es decir se usa un *flatMap* de *deps.modules* donde toma las lista de dependencias individuales y las junta en un única arreglo contando luego su longitud del arreglo.

```

import fs from 'fs';

const raw = fs.readFileSync('./deps.json', 'utf-8');
const deps = JSON.parse(raw);

// B = Número total de Clases/Módulos (Total de funciones principales a analizar)
const B = deps.modules.length;

// Se obtiene el número total de relaciones/dependencias en todo el sistema.
const totalRelations = deps.modules.flatMap(m => m.dependencies).length;

// A = Acoplamiento Promedio: Total de relaciones / Total de módulos
// Este es el promedio de relaciones que tiene una unidad (módulo/función) con otras clases.
const A = B > 0 ? (totalRelations / B) : 0;

// X es igual a A (la métrica)
const X = +A.toFixed(3);

console.log('Acoplamiento de Clases (X) basado en promedio por módulo:', X);

```

Figura 108 Script de la métrica MT1M2

4.4.3.3. Ejecución de Reutilización MT2M1

La Tabla 19 nos da detalle del comando usado, luego se usó el siguiente Script como se observa en la Figura 109 que representa la métrica MT2M1, definida como $X=A/B$, donde A fue representado como los ítems realmente reutilizados y B como el total de ítems reutilizables. En primer lugar se definió REUSABLE_LIBRARY_PATHS con la lista de directorios reutilizables, luego cargamos el grafo *deeps.json*, el script iterara los modelos del proyecto y verificó si su ruta *modulo.source* comienza con la lista de directorios reutilizables, entonces si coincide con alguna de las rutas se añade al conjunto de *allReusablePaths* para evitar conteos duplicados dándonos como resultado el conjunto de la variable B.

Para la variable A, el script genera un gran listado de módulos del proyecto que representa todos los destinos dentro del sistema esto se logra aplanados la lista de dependencias de cada modulo *deps.modules.flatMap(...)*. Si la ruta de módulos aparece en la lista de destinos, significa que tiene al menos una dependencia entrante, por lo que se considera un elemento reutilizado y se añade al conjunto de *reusedElements* dando como resultado el conjunto de la variable A.

```

import fs from 'fs';

const REUSABLE_LIBRARY_PATHS = [
  'src/adapters',
  'src/lib/'
  // AGREGAR O ELIMINAR RUTAS SEGÚN LA ESTRUCTURA REAL DE TU PROYECTO
];

// LECTURA DIRECTA DE DATOS (Requiere que 'deps.json' exista)
const raw = fs.readFileSync('./deps.json', 'utf-8');
const deps = JSON.parse(raw);

// --- LÓGICA DE CÁLCULO MS2MI (Reutilización) ---

// 1. OBTENER B: Total de elementos en las rutas reutilizables
const allReusablePaths = new Set();

deps.modules.forEach(module => {
  // Verificamos si la ruta de este módulo comienza con CUALQUIERA de las rutas candidatas
  const isCandidate = REUSABLE_LIBRARY_PATHS.some(reusablePath =>
    module.source.startsWith(reusablePath)
  );

  if (isCandidate) {
    allReusablePaths.add(module.source);
  }
});

const reusableModules = Array.from(allReusablePaths);
const B = reusableModules.length; // B = Total de Módulos Candidatos

// 2. OBTENER A: Número de elementos que son realmente reutilizados (tienen dependencias entrantes).

// A) Recolectamos todos los destinos de dependencias de todo el proyecto.
const allDependencies = deps.modules.flatMap(m => m.dependencies.map(d => d.resolved));
const reusedElements = new Set();

reusableModules.forEach(modulePath => {
  // B) Verificamos si el módulo es importado en algún lugar.
  if (allDependencies.includes(modulePath)) {
    reusedElements.add(modulePath);
  }
});

const A = reusedElements.size; // A = Módulos reutilizados al menos 1 vez

// 3. CÁLCULO FINAL X = A/B
const X = B > 0 ? (A / B) : 0;
const X_formatted = +X.toFixed(3);

// --- RESULTADOS EN CONSOLA ---
console.log(`\n--- MS2MI: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---`);
console.log(`RUTAS ANALIZADAS (Universo B): ${REUSABLE_LIBRARY_PATHS.join(', ')}`);
console.log('-----');
console.log(`B (Total de Módulos Candidatos a Reutilizar): ${B}`);
console.log(`A (Módulos Reutilizados en el Proyecto): ${A}`);
console.log('-----');
console.log(`MS2MI (X): ${X_formatted} (El valor más cercano a 1 es el mejor)`);

```

Figura 109 Script de la métrica MT2M1

4.4.3.4. Capacidad de auditoría de seguimiento MT3M1

La Tabla nos da detalle del comando usado, posteriormente se usó el siguiente Script como se observa en la Figura que representa la métrica MT3M1, definida como $X=A/B$, donde A fue representado como los datos realmente registrados durante una operación y B los datos esperados a ser grabados para controlar el estado del sistema. En primer lugar se define las variables *CRITICAL_PATH* y *LOGGING KEYWORDS*, lógica crítica en el estado del sistema y palabras claves que se buscó en el script. Luego se cargó los datos por *deeps.json*, el script iteró sobre todos los módulos *deeps.modules.forEach* y verificar la ruta con *modules.source* con la asignación de *CRITICAL_PATH* los módulos que cumple el criterio se almacena en *criticalModulePaths* y *criticalModules* se asigna a la variable B como el número de datos esperados a registrar.

```
import fs from 'fs';
import path from 'path';

// PASO 1: Define las rutas donde deben ocurrir los eventos de auditoria (UNIVERSO B).
const CRITICAL_PATHS = [
  'src/core/application/', // Donde ocurre la lógica de negocio
  'src/ui/port/', //
];

// PASO 2: Palabras clave que indican que se está realizando una auditoria (CONTADOR A).
const LOGGING_KEYWORDS = ['logger', 'audit', 'log', 'saveLog', 'console.error'];
const KEYWORDS_REGEX = new RegExp(`${LOGGING_KEYWORDS.join('|')}`, 'i');

// --- LÓGICA DE CÁLCULO MT3M1 ---

const raw = fs.readFileSync('./deeps.json', 'utf-8');
const deps = JSON.parse(raw);

// 1. OBTENER B: Total de módulos en las rutas críticas.
let criticalModules = [];
const criticalModulePaths = new Set();

deps.modules.forEach(module => {
  const isCritical = CRITICAL_PATHS.some(criticalPath =>
    module.source.startsWith(criticalPath)
  );

  if (isCritical) {
    criticalModulePaths.add(module.source);
    // Almacenamos el objeto completo del módulo para usar su ruta luego.
    criticalModules.push(module);
  }
});

// B = Número de expected data (total de archivos en las rutas críticas)
const B = criticalModulePaths.size;

// 2. OBTENER A: Módulos que realmente contienen lógica de auditoria.
let auditedCount = 0;
const auditedModules = [];

for (const module of criticalModules) {
  const filePath = path.resolve(module.source);

  // Intentar leer el contenido del archivo
  const fileContent = fs.readFileSync(filePath, 'utf-8');

  // Verificar si el contenido del archivo contiene alguna de las palabras clave de logging
  if (KEYWORDS_REGEX.test(fileContent)) {
    auditedCount++;
    auditedModules.push(module.source);
  }
}

// A = Number of data actually recorded (número de archivos críticos que tienen logging)
const A = auditedCount;

// 3. CÁLCULO FINAL X = A/B
const X = B > 0 ? (A / B) : 0;
const X_formatted = +X.toFixed(3);

// --- RESULTADOS EN CONSOLA ---
console.log(`\n- MT3M1: CAPACIDAD DE LA PISTA DE AUDITORIA (X = A/B) -`);
console.log(`RUTAS CRÍTICAS ANALIZADAS (Universo B): ${CRITICAL_PATHS.join(', ')}`);
console.log(`KEYWORDS DE AUDITORIA BUSCADAS (Define A): ${LOGGING_KEYWORDS.join(', ')}`);
console.log(`-----`);
console.log(`B (Total de Módulos Críticos Esperados): ${B}`);
console.log(`A (Módulos Críticos que Contienen Auditoria): ${A}`);
console.log(`-----`);
console.log(`MT3M1 (X): ${X_formatted} (El valor más cercano a 1 es el mejor)`);
```

Figura 110 Script de la métrica MT3M1

4.4.3.5. Diagnóstico de funciones MT3M2

La Tabla 19 nos da detalle del comando usado, posteriormente se usó el siguiente Script como se observa en la Figura 111 que representa la métrica MT3M2, definida como $X=A/B$, donde A fue representado como el número de funciones diagnosticadas implementadas y B como el número de funciones de diagnóstico requerido en la especificación.

```
import fs from 'fs';
import path from 'path';

// PASO 1: Define las rutas donde deben estar los HEALTH CHECKS y funciones de DIAGNÓSTICO (UNIVERSO B).
const DIAGNOSTIC_PATHS = [
  'src/core/application/repository/',
  'src/core/application/services/',
  'src/ui/pages',
  'src/ui/ports'
];

// PASO 2: Palabras clave que indican que la función de diagnóstico está implementada (CONTADOR A).
const DIAGNOSTIC_KEYWORDS = ['ping', 'health', 'status', 'await', 'throw', 'try', 'console', 'logger', 'isReady'];
const KEYWORDS_REGEX = new RegExp(`${DIAGNOSTIC_KEYWORDS.join('|')}`, 'i');

// --- LÓGICA DE CÁLCULO MT3M2 ---
const raw = fs.readFileSync('./deps.json', 'utf-8');
const deps = JSON.parse(raw);

// 1. OBTENER B: Total de módulos en las rutas de diagnóstico esperadas.
let diagnosticModules = [];
const diagnosticModulePaths = new Set();

deps.modules.forEach(module => {
  const isDiagnosticCandidate = DIAGNOSTIC_PATHS.some(diagPath =>
    module.source.startsWith(diagPath)
  );

  if (isDiagnosticCandidate) {
    diagnosticModulePaths.add(module.source);
    diagnosticModules.push(module);
  }
});

// B = Number of diagnosis functions required in the specification (Archivos en la capa de diagnóstico)
const B = diagnosticModulePaths.size;

// 2. OBTENER A: Módulos que realmente contienen lógica de diagnóstico.
let implementedCount = 0;
const implementedModules = [];

// Iteramos sobre el universo B y leemos el código
for (const module of diagnosticModules) {
  const filePath = path.resolve(module.source);

  // La lectura del contenido nos permite verificar la IMPLEMENTACIÓN real
  const fileContent = fs.readFileSync(filePath, 'utf-8');

  // Verificamos si el contenido del archivo contiene alguna de las palabras clave de diagnóstico
  if (KEYWORDS_REGEX.test(fileContent)) {
    implementedCount++;
    implementedModules.push(module.source);
  }
}

// A = Number of diagnostic implemented functions
const A = implementedCount;

// 3. CÁLCULO FINAL X = A/B
const X = B > 0 ? (A / B) : 0;
const X_formatted = X.toFixed(3);

// --- RESULTADOS EN CONSOLA ---
console.log('\n--- MT3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---');
console.log('RUTAS ANALIZADAS (Universo B - Requeridas):', `${DIAGNOSTIC_PATHS.join(', ')}`);
console.log('KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A):', `${DIAGNOSTIC_KEYWORDS.join(', ')}`);
console.log('-----');
console.log('B (Total de Módulos de Diagnóstico Requeridos):', `${B}`);
console.log('A (Módulos de Diagnóstico Implementados):', `${A}`);
console.log('-----');
console.log('MT3M2 (X):', `${X_formatted}`);
console.log('El valor más cercano a 1 es el mejor');
```

Figura 111 Script de la métrica MT3M2

En primer lugar se define la lógica de las rutas luego se itera sobre el mapa de módulos *deps.modules* y se filtra aquellos cuyas rutas *modules.source* coinciden con alguna carpeta listada de *DIAGNOSTIC_PATHS* entonces la variable B se establece como el conteo total de estos archivos críticos.

En el caso de la variable A se utilizó la iteración del script en cada archivo del universo B dentro de un proceso de lectura y búsqueda sobre el contenido de cada archivo usando la expresión regular *KEYWORDS_REGEX* para buscar la presencia de las palabras claves el conteo de archivos que tuvieron coincidencia se establece como la Variable A .

4.4.3.6. Complejidad Ciclomática MT4M1

La métrica MT4M1, definida como $X=A + 1$, donde A es el número de funciones implementadas.

El script que indica la Figura 112 se basa en primer lugar la configuración del acceso al endpoint usada de la herramienta SonarQube mostrada en la Figura estos consta de *SONAR_URL*, *PROJECT_KEY*, *SONAR_TOKEN* y *METRICS*, obteniendo el valor final en un *API_ENDPOINT*, con la información brindada se hace una petición de tipo GET guardando la información es una variable data.

Luego el script itera en *data.component.measures* para localizar y asignar valores devueltos por el SonarQube dando como resultado que la variable *totalComplexitySum* reciba el valor de la complejidad ciclomática de todos los archivos del proyecto en forma de suma y la variable B obtiene el valor de total de funciones dentro del proyecto.

Finalmente se realizó una división y obtuvimos el valor de A la complejidad ciclomática según el número de funciones que tiene nuestro proyecto.

```

import fetch from 'node-fetch';
const SONAR_URL = 'http://localhost:9000';
const PROJECT_KEY = 'Grupo-experimental';
const SONAR_TOKEN = 'TOKEN';
// complexity (EX) y functions (B)
const METRICS = 'complexity,functions';
const API_ENDPOINT = `${SONAR_URL}/api/measures/component?component=${PROJECT_KEY}&metricKeys=${METRICS}`;

/**
 * Función que realiza la llamada a la API de SonarQube, extrae los datos y calcula MS4M1.
 */
async function getSonarMetricsAndCalculateMS4M1() {
  console.log(`\n--- MS4M1: OBTENIENDO DATOS DE SONARQUBE ---`);
  console.log(`URL de la API: ${API_ENDPOINT}`);

  const options = {
    method: 'GET',
    headers: {
      'Authorization': 'Basic ${Buffer.from(SONAR_TOKEN + ':').toString('base64')}'
    }
  };

  try {
    // 1. Realizar la llamada a la API
    const response = await fetch(API_ENDPOINT, options);
    const data = await response.json();
    // 2. Extraer los valores de las métricas
    let totalComplexitySum = 0; // EX
    let B = 0; // Total de Funciones

    for (const measure of data.component.measures) {
      if (measure.metric === 'complexity') {
        totalComplexitySum = parseInt(measure.value, 10);
      } else if (measure.metric === 'functions') {
        B = parseInt(measure.value, 10);
      }
    }

    // 3. Validar y calcular
    if (totalComplexitySum === 0 || B === 0) {
      throw new Error('Datos faltantes. Complejidad Total: ${totalComplexitySum}, Funciones Totales: ${B}');
    }

    const X_average = totalComplexitySum / B;
    const X_formatted = +X_average.toFixed(3);

    // 4. Mostrar Resultados
    console.log('-----');
    console.log('DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:');
    console.log(`EX (Suma Total de Complejidad): ${totalComplexitySum}`);
    console.log(`B (Total de Funciones Analizadas): ${B}`);
    console.log('-----');
    console.log(`MS4M1 (X Promedio del Proyecto): ${X_formatted}`);
    console.log(`Interpretación: Valor ideal cercano a 1 (Limite de Riesgo: 15).`);
    console.log('-----');
  } catch (error) {

```

Figura 112 Script de la métrica MT4M1

4.4.3.7. Profundidad de la herencia MT4M2

En esta sección se detalló el procedimiento para obtener la profundidad del Árbol de herencias (DIT) en el proyecto, está midió la complicitad de la estructura vertical de las clases. En esta métrica no se pudo obtener resultados de DIT por SonarQube y tampoco se pudo usar la herramienta Dependency cruiser ya que lo analiza a nivel de módulos como en la Figura 113 y 114 .

```

import { promises as fs } from 'fs';
import path from 'path';

// --- CONFIGURACIÓN ---
const SOURCE_DIR = 'src';
const OUTPUT_FILE = 'complexity_mt4m2_data.json';
const MAX_DIT = 4; // Límite máximo de DIT para la métrica (0 <= X <= 4)

// --- REGEX PARA DETECTAR EXTENDS ---
const EXTENDS_REGEX = /(?:abstract\s+)?class\s+(\w+)(?:<[>{}]+>)?\s+extends\s+([\w\.\s]+)(?:<[>{}]+>)?/g;

// --- REGEX PARA DETECTAR CUALQUIER CLASE ---
const CLASS_REGEX = /(?:abstract\s+)?class\s+(\w+)(?:<[>{}]+>)?\s*(?:implements\s+[\^{}]+)?\s*\//g;

// Estructuras de datos para el mapeo
const inheritanceMap = new Map(); // Hijo: Padre
const allClassesList = new Set(); // Lista de todas las clases detectadas en el proyecto
const fileMap = new Map(); // Clase: Ruta del archivo

// Función de utilidad para obtener archivos
async function getSourceFiles(dir) {
  let fileList = [];
  const items = await fs.readdir(dir, { withFileTypes: true });
  for (const item of items) {
    const fullPath = path.join(dir, item.name);
    if (item.isDirectory()) {
      if (!fullPath.includes('node_modules') && !item.name.startsWith('.')) {
        fileList = fileList.concat(await getSourceFiles(fullPath));
      }
    } else if (item.isFile()) && (fullPath.endsWith('.ts') || fullPath.endsWith('.tsx')) {
      fileList.push(fullPath);
    }
  }
  return fileList;
}

```

Figura 113 Primer parte del Script de la métrica MT4M2 de análisis

Primero se inició el Script de análisis *MT4M2.js* el cual consta de escanear archivos desde *getSourceFiles* donde buscó los archivos *.ts* y *.tsx* del proyecto, segundo paso el mapeo de herencia desde el *analyzeFile* utiliza la expresión regular *EXTENDS_REGEX* para identificar todas las relaciones **hijo extiende Padre** (extends y implements) esta información fue almacenada en *inheritanceMap*, tercer paso la identificación de clases Base mediante el *analyzeFile* (*filePath*), cuarto paso calcular el DIT(x) *calcularDepth* (*className*), para cada clase detectada, esta función se llama recursivamente: a) Si la clase tiene un padre en *inheritanceMap*, suma 1 y llama a *calculateDepth* sobre el padre. b) Si la clase no tiene un padre en el mapa (es la raíz o hereda de una librería externa), devuelve 0. Finalmente, la generación del reporte mediante el archivo *complexity_MT4M2_data.json*.

El siguiente Script carga los datos del json *complexity_MT4M2_data.json*, luego realizó una suma de los *results* mediante un *forEach* y finalmente obtuvo la métrica género la suma de profundidad de herencia entre las clases encontradas, es decir, obtuvo el valor de $X = A$, donde A es el número así como indica el Script de la Figura 115.

```

// .....
// Función de Análisis
// .....
async function analyzeFile(filePath) {
  const content = await fs.readFile(filePath, 'utf-8');

  // 1. Buscar clases con Herencia (EXTENDS)
  EXTENDS_REGEX.lastIndex = 0;
  let match;
  while ((match = EXTENDS_REGEX.exec(content)) !== null) {
    const className = match[1];
    const parentName = match[2];

    inheritanceMap.set(className, parentName);
    allClassesList.add(className);
    fileMap.set(className, filePath);

    // Agregar el padre al set para asegurar el rastreo DIT completo
    allClassesList.add(parentName);
  }

  // 2. Buscar clases BASE (que no usen 'extends')
  CLASS_REGEX.lastIndex = 0;
  while ((match = CLASS_REGEX.exec(content)) !== null) {
    const className = match[1];
    if (!inheritanceMap.has(className)) {
      allClassesList.add(className);
      fileMap.set(className, filePath);
    }
  }
}
}

```

Figura 114 Segunda parte del Script de la métrica MT4M2 de análisis

```

MT4M2_inaja> ...
1 import { promises as fs } from 'fs';
2
3 const INPUT_FILE = 'complexity_mt4m2_data.json';
4 const MAX_DIT = 4; // Asegurate de que este valor coincida con el del analizador
5
6 async function calculateAverageMT4M2() {
7   console.log('\n--- CÁLCULO DEL PROMEDIO MT4M2 (DIT) ---');
8
9   const data = await fs.readFile(INPUT_FILE, 'utf-8');
10  const results = JSON.parse(data);
11
12  if (results.length === 0) {
13    console.log('No hay datos en el archivo de entrada. Ejecuta el analizador primero.');

```

Figura 115 Script de la métrica MT4M2 de cálculo.

4.4.3.8. Complejidad de Modificación MT4M4

En esta métrica usamos Git Bash para obtener el historial de commits realizados en las ramas del proyecto y usamos las 8 horas de trabajo de horario laboral diario como base de tiempo.

Primero en el script se asumió las 8 horas en la variable `ASSUMED_TIME_T` luego se usó la regla de filtro `COMMIT_FILTER_REGEX` para obtener tanto feat cómo fix.

```
MT4M4.js calculateMT4M4
import { exec } from 'child_process';
import { promisify } from 'util';
const execPromise = promisify(exec);
const ASSUMED_TIME_T = 8.0;
// Regexp para filtrar commits (debes usar 'feat' o 'fix' al inicio del mensaje)
const COMMIT_FILTER_REGEX = /^(feat|fix):/i;

// Función principal para calcular la métrica MT4M4
async function calculateMT4M4() {

  // 1. Obtener la lista de commits filtrados y sus hashes
  const { stdout: logOutput } = await execPromise('git log --pretty=format:"%H %s"');
  const allCommits = logOutput.trim().split('\n').filter(line => line.trim() !== '');

  // Mapear y filtrar los commits válidos
  const filteredCommitData = allCommits
    .map(line => {
      const parts = line.split(' ');
      const hash = parts[0];
      const message = parts.slice(1).join(' ');
      return { hash, message };
    })
    .filter(data => COMMIT_FILTER_REGEX.test(data.message));

  let totalModificationsLOC_A1 = 0; // LOC (Líneas de Código) modificadas
  let totalCommits_A2 = 0; // Conteo de commits

  // 2. Iterar sobre los commits para sumar las líneas (A1) y contarlos (A2)
  console.log(`\nProcesando ${filteredCommitData.length} commits con prefijo 'feat' o 'fix'...`);

  for (const data of filteredCommitData) {

    // Obtener estadísticas de líneas añadidas/eliminadas para A1
    const { stdout: statOutput } = await execPromise('git diff-tree --numstat ${data.hash} --');

    const stats = statOutput.trim().split('\n').filter(line => line.trim() !== '');
    let commitModificationsLOC = 0;

    stats.forEach(statLine => {
      const parts = statLine.split('\t');
      const added = parseInt(parts[0]) || 0;
      const deleted = parseInt(parts[1]) || 0;
      commitModificationsLOC += (added + deleted);
    });

    if (commitModificationsLOC > 0) {
      totalModificationsLOC_A1 += commitModificationsLOC; // Sumar líneas de código (A1)
      totalCommits_A2++; // Contar el commit (A2)
    }
  }

  // 3. Cálculo de MT4M4 (X = A2 / T)
  const MT4M4_X = totalCommits_A2 / ASSUMED_TIME_T;

  console.log(`\n=====`);
  console.log(`Análisis de Productividad y Densidad Completado!`);

  // Resultados Detallados
  console.log(`\n## MÉTRICAS CALCULADAS`);
  console.log(`  A1 (Líneas de Código Modificadas): **${totalModificationsLOC_A1} líneas**`);
  console.log(`  A2 (Total de Commits Válidos): **${totalCommits_A2} commits**`);
  console.log(`  T (Tiempo Asumido en horas laborales): **${ASSUMED_TIME_T} horas**`);
  console.log(`  =====`);
  console.log(`  **MT4M4 (X) (X) = A2 / T = ${MT4M4_X.toFixed(3)} commit modificado / hora**`);
  console.log(`  ** ${totalModificationsLOC_A1.toFixed(3)} / (ASSUMED_TIME_T * 60 * 60) líneas de código`);
  console.log(`  =====`);
}
```

Figura 116 Script de la métrica M4TM4

Segundo paso se extrajo los datos con el comando `git log --pretty=format:"%H %s"` para recuperar todos los commits y posteriormente con el `filteredCommitData` calculamos las líneas modificadas. Finalmente, la métrica se obtendrá como el contenido de commit / las 8 horas laborales. Así como lo indica la Figura 116

4.4.3.9. Complejidad Funcional del test MT5M1

Para esta métrica se analizó el código estático a través de la herramienta de SonarQube a través del comando `npm run test:coverage` que nos da como resultado el coverage del proyecto en un formato xml, que fue identificado por nuestro sonarQube e interpretado a nivel de un diagrama y el porcentaje total de coverage del proyecto como en la Figura 117.

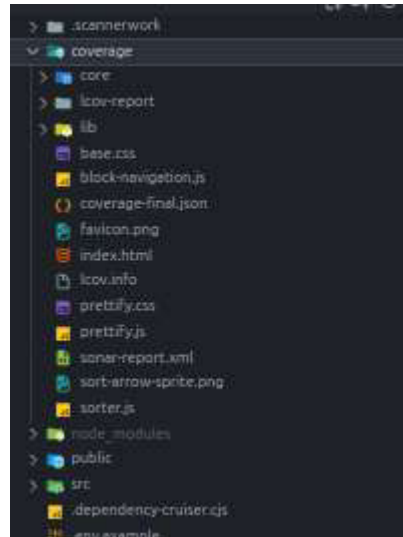


Figura 117 Coverage de la métrica MT5M1

4.4.4. Análisis de la Arquitectura de 3 capas basada en Clean Architecture

En este caso, el proyecto fue analizado y clasificado en grado de mantenibilidad, los resultados indicaron que el proyecto se encuentra en buenas condiciones, tal como se muestra en la Figura 118.

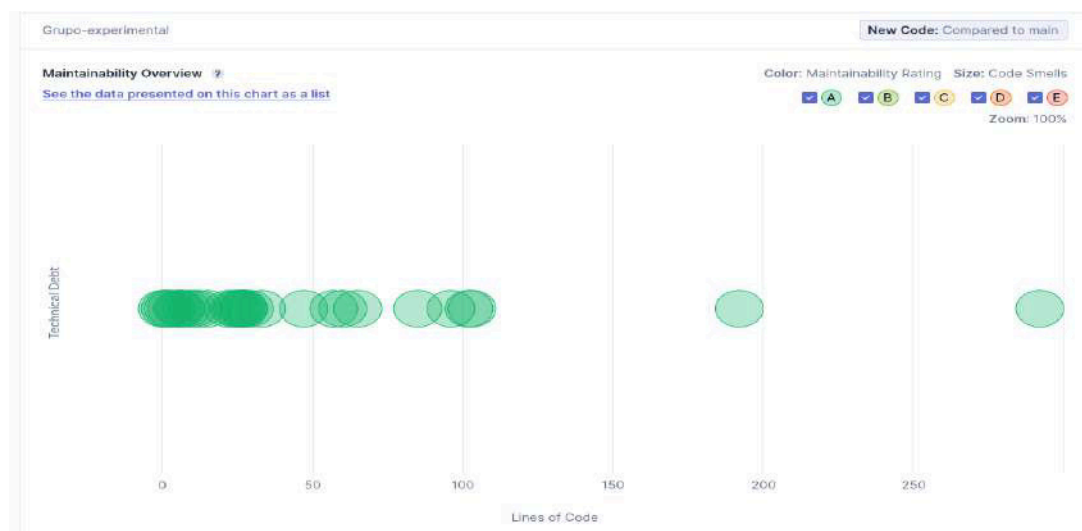


Figura 118 Análisis de mantenibilidad del grupo experimental

Para la validación del sistema se utilizaron los datos obtenidos a partir del análisis realizado en SonarQube. En la Tabla 20 se presentan las características correspondientes al sistema de tres capas, y en la Tabla 21 los datos procesados del Grupo experimental

Tabla 20 Datos del grupo experimental

Grupo experimental	Caso 1	Caso 2	Caso 3	Caso4	Caso promedio
Archivos	120	121	121	122	121
Total de líneas de código	21211	21289	21745	21367	21403

Tabla 21 Datos preprocesados del grupo experimental

Grupo experimental	Caso 1	Caso 2	Caso 3	Caso4	Caso promedio
Archivos	77	78	78	79	78
Total de líneas de código	6994	6915	6993	6927	6957.25

4.4.4.1. Capacidad de condensación - MTIM1

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La capacidad de condensación, definida por la Ecuación (1) como $X=A/B$

- **Caso de estudio 1**

Entonces el valor de X es determinado utilizando los datos estructurales obtenidos, es decir la variable A como el número de componentes sin afectación es 23 y B como el número total de componentes específicos es 91. Al aplicar el cociente, se obtiene un valor de capacidad de condensación de 0.253 como lo indica la Figura 119

```
ricse@RicseDev:~/tesis/carpet/MOVIEST npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIEST node MTIM1.js
Variable A : 23
Variable B: 91
Reporte generado con X = 0.253
```

Figura 119 Capacidad de condensación del grupo experimental caso 1

- **Caso de estudio 2**

Entonces el valor de X es determinado utilizando los datos estructurales obtenidos, es decir la variable A como el número de componentes sin afectación es 23 y B como el número total de componentes específicos es 92. Al aplicar el cociente, se obtiene un valor de capacidad de condensación de 0.25 como lo indica la Figura 120

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MTIM1.js
Variable A : 23
Variable B: 92
Reporte generado con X = 0.25
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 120 Capacidad de condensación del grupo experimental caso 2

- **Caso de estudio 3**

Entonces el valor de X es determinado utilizando los datos estructurales obtenidos, es decir la variable A como el número de componentes sin afectación es 23 y B como el número total de componentes específicos es 92. Al aplicar el cociente, se obtiene un valor de capacidad de condensación de 0.25 como lo indica la Figura 121

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MTIM1.js
Variable A : 23
Variable B: 92
Reporte generado con X = 0.25
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 121 Capacidad de condensación del grupo experimental caso 3

- **Caso de estudio 4**

Entonces el valor de X es determinado utilizando los datos estructurales obtenidos, es decir la variable A como el número de componentes sin afectación es 32 y B como el número total de componentes específicos es 103. Al aplicar el cociente, se obtiene un valor de capacidad de condensación de 0.311 como lo indica la Figura 122

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node mtlm1.js
Variable A : 32
Variable B: 103
Reporte generado con X = 0.311
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 122 Capacidad de condensación del grupo experimental caso 4

4.4.4.2. Acoplamiento - MTIM2

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La capacidad de acoplamiento, definida por la Ecuación (2) como $X=A$, como el número de relaciones que tiene una función sobre otras clases o módulos.

- **Caso de estudio 1**

El valor de X se define como el valor de A, es decir un valor de 2.143 como lo indica la Figura 123

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src$" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MTIM2.js
Total de Relaciones: 195
Total de Clases (B): 91
Acoplamiento de Clases (X): 2.143
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 123 Acoplamiento del grupo experimental caso 1

- **Caso de estudio 2**

El valor de X se define como el valor de A, es decir un valor de 2.228 como lo indica la Figura 124.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src$" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MTIM2.js
Total de Relaciones: 285
Total de Clases (B): 92
Acoplamiento de Clases (X): 2.228
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 124 Acoplamiento del grupo experimental caso 2

- **Caso de estudio 3**

El valor de X se define como el valor de A, es decir un valor de 2.13 como lo indica la Figura 125.

```

Acoplamiento de Clases (X): 2.13
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MTM2.js
Total de Relaciones: 196
Total de Clases (B): 92
Acoplamiento de Clases (X): 2.13
ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 125 Acoplamiento del grupo experimental caso 3

- **Caso de estudio 4**

El valor de X se define como el valor de A, es decir un valor de 2.107 como lo indica la Figura 126.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MTM2.js
Total de Relaciones: 217
Total de Clases (B): 103
Acoplamiento de Clases (X): 2.107
ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 126 Acoplamiento del grupo experimental caso 4

4.4.4.3. Ejecución de reutilización - MT2M1

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La ejecución de reutilización es definida por la Ecuación (3) como $X=A/B$, donde A es el número de ítems reutilizados y B el total de ítems reutilizables, siendo el caso de estudio experimental las carpetas utils y ports.

- **Caso de estudio 1**

El valor de X se define $A = 26$ entre $B = 28$, dando como resultado 0.929 como lo indica la Figura 127.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT2M1.js
--- MS2M1: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---
RUTAS ANALIZADAS (Universe B): src/ui/ports/, src/core, src/lib/
-----
B (Total de Módulos Candidatos a Reutilizar): 28
A (Módulos Reutilizados en el Proyecto): 26
-----
MS2M1 (X): 0.929 (El valor más cercano a 1 es el mejor)
-----

```

Figura 127 Ejecución de reutilización del grupo experimental caso 1

- **Caso de estudio 2**

El valor de X se define $A = 27$ entre $B = 29$, dando como resultado 0.931 como lo indica la Figura 128.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT2M1.js
--- MS2M1: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---
RUTAS ANALIZADAS (Universo B): src/ui/port, src/core, src/lib/
-----
B (Total de Módulos Candidatos a Reutilizar): 29
A (Módulos Reutilizados en el Proyecto): 27
-----
MS2M1 (X): 0.931 (El valor más cercano a 1 es el mejor)
-----
```

Figura 128 Ejecución de reutilización del grupo experimental caso 2

- **Caso de estudio 3**

El valor de X se define $A = 27$ entre $B = 29$, dando como resultado 0.931 como lo indica la Figura 129.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT2M1.js
--- MS2M1: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---
RUTAS ANALIZADAS (Universo B): src/ui/port, src/core, src/lib/
-----
B (Total de Módulos Candidatos a Reutilizar): 29
A (Módulos Reutilizados en el Proyecto): 27
-----
MS2M1 (X): 0.931 (El valor más cercano a 1 es el mejor)
-----
```

Figura 129 Ejecución de reutilización del grupo experimental caso 3

- **Caso de estudio 4**

El valor de X se define $A = 28$ entre $B = 30$, dando como resultado 0.933 como lo indica la Figura 130.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT2M1.js
--- MS2M1: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---
RUTAS ANALIZADAS (Universo B): src/ui/port, src/core, src/lib/
-----
B (Total de Módulos Candidatos a Reutilizar): 30
A (Módulos Reutilizados en el Proyecto): 28
-----
MS2M1 (X): 0.933 (El valor más cercano a 1 es el mejor)
-----
```

Figura 130 Ejecución de reutilización del grupo experimental caso 4

4.4.4.4. Capacidad de seguimiento de auditoría - MT3M1

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La capacidad de seguimiento de auditoría es definida por la Ecuación (4) como $X=A/B$, donde A es el número de datos realmente registrados y B el número de datos esperados a ser grabados para controlar el estado del sistema, siendo el caso de estudio experimental las carpetas utils y ports.

- **Caso de estudio 1**

El valor de X se define A = 7 entre B = 11 , dando como resultado 0.636 como lo indica la Figura 131.

```
ricsegRicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "**src*" --output-type json
src > deps.json
ricsegRicseDev:~/tesis/carpet/MOVIES$ node MT3M1.js

--- MS3M1: CAPACIDAD DE LA PISTA DE AUDITORÍA (X = A/B) ---
RUTAS CRÍTICAS ANALIZADAS (Universo B): src/core/application/, src/ui/port/
KEYWORDS DE AUDITORÍA BUSCADAS (Define A): logger, audit, log, saveLog, console.error
-----
B (Total de Módulos Críticos Esperados): 11
A (Módulos Críticos que Contienen Auditoría): 7
-----
MS3M1 (X): 0.636 (El valor más cercano a 1 es el mejor)
```

Figura 131 Capacidad de seguimiento de auditoría del grupo experimental caso 1

- **Caso de estudio 2**

El valor de X se define A = 7 entre B = 12 , dando como resultado 0.583 como lo indica la Figura 132

```
ricsegRicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "**src*" --output-type json
src > deps.json
ricsegRicseDev:~/tesis/carpet/MOVIES$ node MT3M1.js

--- MS3M1: CAPACIDAD DE LA PISTA DE AUDITORÍA (X = A/B) ---
RUTAS CRÍTICAS ANALIZADAS (Universo B): src/core/application/, src/ui/port/
KEYWORDS DE AUDITORÍA BUSCADAS (Define A): logger, audit, log, saveLog, console.error
-----
B (Total de Módulos Críticos Esperados): 12
A (Módulos Críticos que Contienen Auditoría): 7
-----
MS3M1 (X): 0.583 (El valor más cercano a 1 es el mejor)
```

Figura 132 Capacidad de seguimiento de auditoría del grupo experimental caso 2

- **Caso de estudio 3**

El valor de X se define $A = 7$ entre $B = 12$, dando como resultado 0.583 como lo indica la Figura 133

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M1.js

--- MS3M1: CAPACIDAD DE LA PISTA DE AUDITORIA (X = A/B) ---
RUTAS CRÍTICAS ANALIZADAS (Universe B): src/core/application/, src/ui/port/
KEYWORDS DE AUDITORIA BUSCADAS (Define A): logger, audit, log, saveLog, console.error
-----
B (Total de Módulos Críticos Esperados): 12
A (Módulos Críticos que Contienen Auditoria): 7
-----
MS3M1 (X): 0.583 (El valor más cercano a 1 es el mejor)
```

Figura 133 Capacidad de seguimiento de auditoría del grupo experimental caso 3

- **Caso de estudio 4**

El valor de X se define $A = 7$ entre $B = 11$, dando como resultado 0.636 como lo indica la Figura 134.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M1.js

--- MS3M1: CAPACIDAD DE LA PISTA DE AUDITORIA (X = A/B) ---
RUTAS CRÍTICAS ANALIZADAS (Universe B): src/core/application/, src/ui/port/
KEYWORDS DE AUDITORIA BUSCADAS (Define A): logger, audit, log, saveLog, console.error
-----
B (Total de Módulos Críticos Esperados): 11
A (Módulos Críticos que Contienen Auditoria): 7
-----
MS3M1 (X): 0.636 (El valor más cercano a 1 es el mejor)
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 134 Capacidad de seguimiento de auditoría del grupo experimental caso 4

4.4.4.5. Diagnóstico de funciones- MT3M2

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La capacidad de seguimiento de auditoría es definida por la Ecuación (5) como $X=A/B$, donde A es el número de funciones diagnosticadas e implementadas y B número de diagnóstico requerido, siendo el caso de estudio experimental las carpetas *utils* y *ports*.

- **Caso de estudio 1**

El valor de X se define $A = 13$ entre $B = 18$, dando como resultado 0.722 como lo indica la Figura 135.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type js
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M2.js

--- MS3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---
RUTAS ANALIZADAS (Universo B - Requeridas): src/core/application/repository/, src/core/app
ation/services/, src/ui/pages, src/ui/ports
KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A): ping, health, status, await, throw, try, c
ole, logger, isReady
-----
B (Total de Módulos de Diagnóstico Requeridos): 18
A (Módulos de Diagnóstico Implementados): 13
-----
MS3M2 (X): 0.722 (El valor más cercano a 1 es el mejor)
```

Figura 135 Diagnóstico de funciones del grupo experimental caso 1

- **Caso de estudio 2**

El valor de X se define $A = 13$ entre $B = 18$, dando como resultado 0.722 como lo indica la Figura 136.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type js
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M2.js

--- MS3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---
RUTAS ANALIZADAS (Universo B - Requeridas): src/core/application/repository/, src/core/app
ation/services/, src/ui/pages, src/ui/ports
KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A): ping, health, status, await, throw, try, c
ole, logger, isReady
-----
B (Total de Módulos de Diagnóstico Requeridos): 18
A (Módulos de Diagnóstico Implementados): 13
-----
MS3M2 (X): 0.722 (El valor más cercano a 1 es el mejor)
```

Figura 136 Diagnóstico de funciones del grupo experimental caso 2

- **Caso de estudio 3**

El valor de X se define $A = 13$ entre $B = 18$, dando como resultado 0.722 como lo indica la Figura 137.

```

• ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type js
src > deps.json
• ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M2.js

--- MS3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---
RUTAS ANALIZADAS (Universo B - Requeridas): src/core/application/repository/, src/core/app
ation/services/, src/ui/pages, src/ui/ports
KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A): ping, health, status, await, throw, try, c
ole, logger, isReady
-----
B (Total de Módulos de Diagnóstico Requeridos): 18
A (Módulos de Diagnóstico Implementados): 13
-----
MS3M2 (X): 0.722 (El valor más cercano a 1 es el mejor)

```

Figura 137 Diagnóstico de funciones del grupo experimental caso 3

- **Caso de estudio 4**

El valor de X se define $A = 13$ entre $B = 18$, dando como resultado 0.722 como lo indica la Figura 138

```

• ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M2.js

--- MS3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---
RUTAS ANALIZADAS (Universo B - Requeridas): src/core/application/repository/, src/core/application/services/, src/ui/pages, src/ui/ports
KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A): ping, health, status, await, throw, try, console, logger, isReady
-----
B (Total de Módulos de Diagnóstico Requeridos): 18
A (Módulos de Diagnóstico Implementados): 13
-----
MS3M2 (X): 0.722 (El valor más cercano a 1 es el mejor)

```

Figura 138 Diagnóstico de funciones del grupo experimental caso 4

4.4.4.6. Complejidad Ciclomática - MT4M1

Tras la generación del análisis estático en SonarQube, se ejecutó el Script para obtener los datos y tener el número de funciones condicionales respecto a la complejidad ciclomática. La complejidad ciclomática por la Ecuación (6) como $X=A+1$, donde A es el número de funciones condicionales respecto a la complejidad ciclomática definida por la ecuación (6.1).

- **Caso de estudio 1**

Donde el valor de A es 1.69 y ejecutando la Ecuación (6) obtuvimos $X=1.69+1 = 2.69$ como lo indica la Figura 139, 140 y 1414

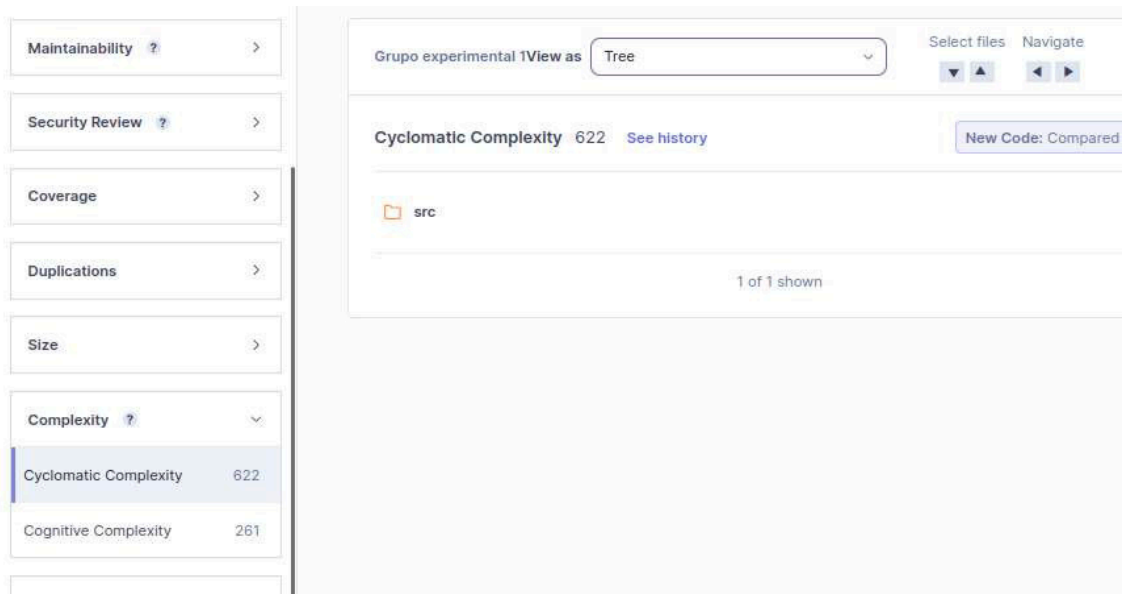


Figura 139 Complejidad Ciclomática desde el SonarQube caso 1

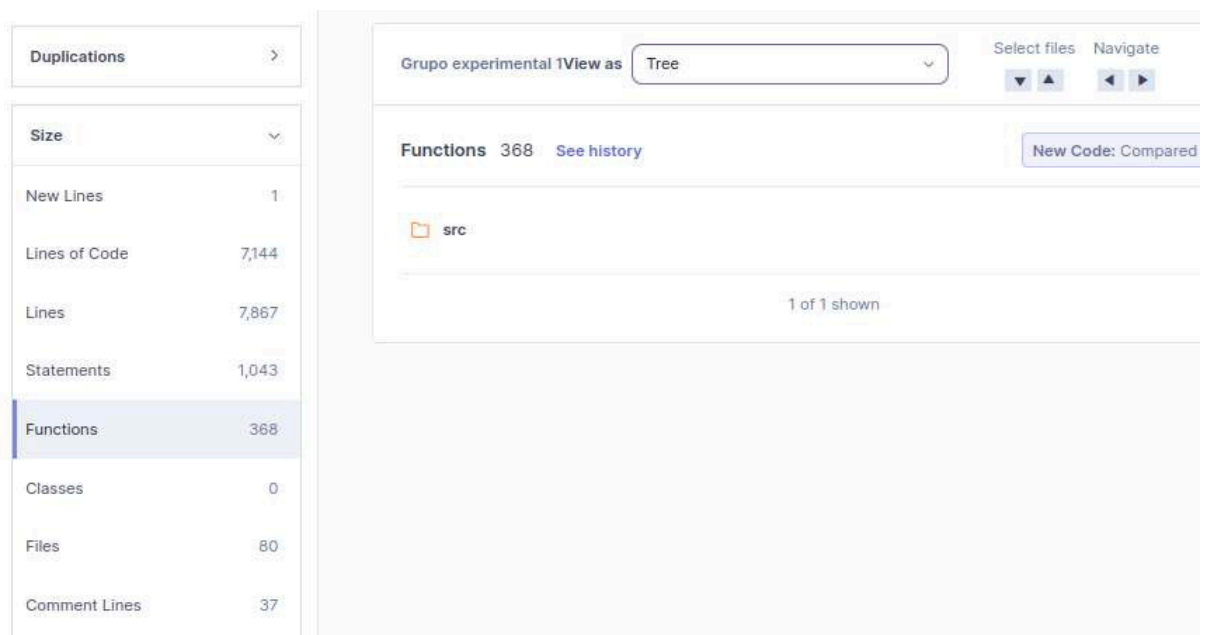


Figura 140 Número total de funciones desde el SonarQube caso 1

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M1.js
--- MS4M1: OBTENIENDO DATOS DE SONARQUBE ---
URL de la API: http://localhost:9000/api/measures/component?component=Grupo-exp-1&metric=complexity,functions
-----
DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:
EX (Suma Total de Complejidad): 622
B (Total de Funciones Analizadas): 368
-----
MS4M1 (X Promedio del Proyecto): 1.69
Interpretación: Valor ideal cercano a 1 (Límite de Riesgo: 15).
```

Figura 141 Número total de funciones desde el SonarQube del grupo experimental caso 1

- **Caso de estudio 2**

Donde el valor de A es 1.688 y ejecutando la Ecuación (6) obtuvimos $X=1.688+1 = 2.688$ como lo indica la Figura 142, 143 y 144.

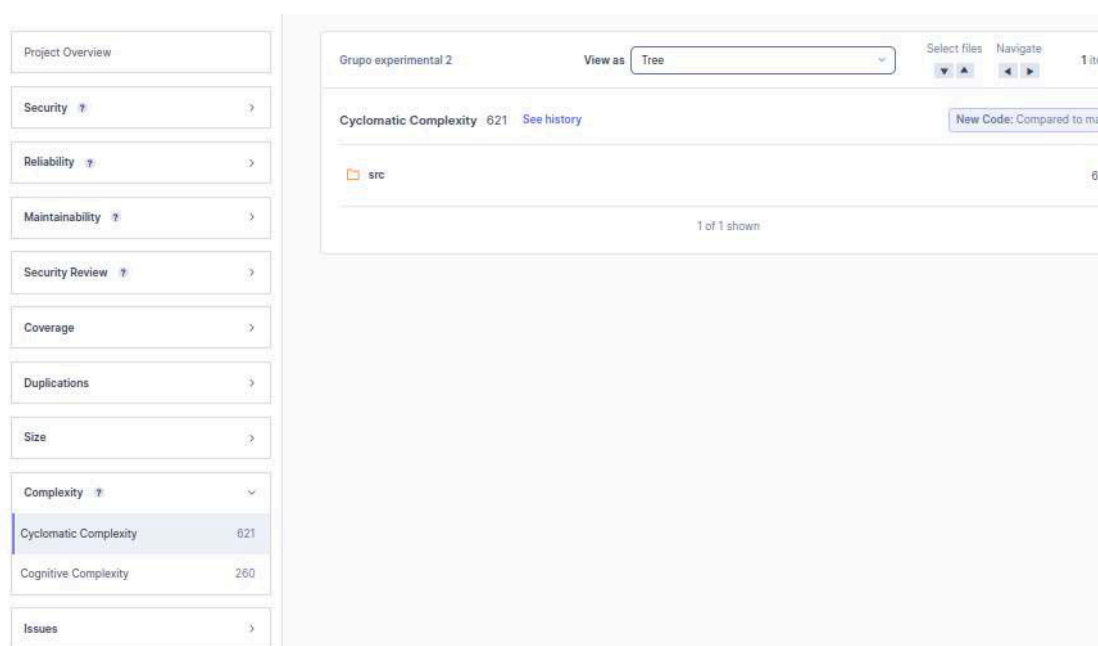


Figura 142 Complejidad Ciclomática desde el SonarQube caso 2

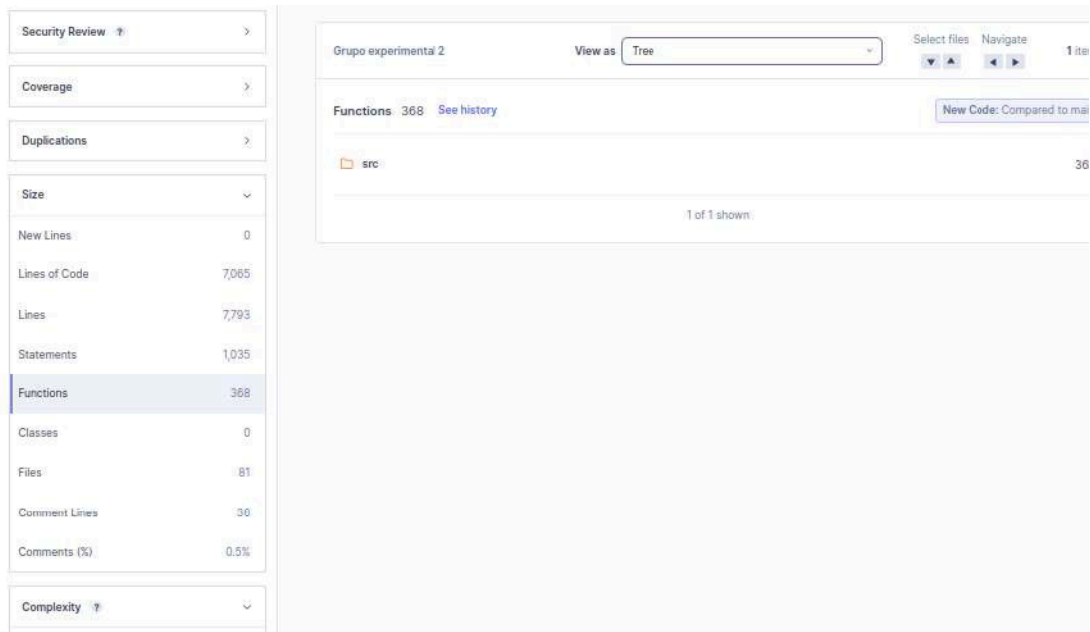


Figura 143 Número total de funciones desde el SonarQube caso 2

```
INFO: ---
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M1.js

--- MS4M1: OBTENIENDO DATOS DE SONARQUBE ---
URL de la API: http://localhost:9000/api/measures/component?component=Grupo-
rickeys=complexity,functions

-----
DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:
EX (Suma Total de Complejidad): 621
B (Total de Funciones Analizadas): 368
-----
MS4M1 (X Promedio del Proyecto): 1.688
Interpretación: Valor ideal cercano a 1 (Límite de Riesgo: 15).
-----
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 144 Número total de funciones desde el SonarQube del grupo experimental caso 2

- **Caso de estudio 3**

Donde el valor de A es 1.692 y ejecutando la Ecuación (6) obtuvimos $X=1.692+1 = 2.692$ como lo indica la Figura 145, 146 y 147.

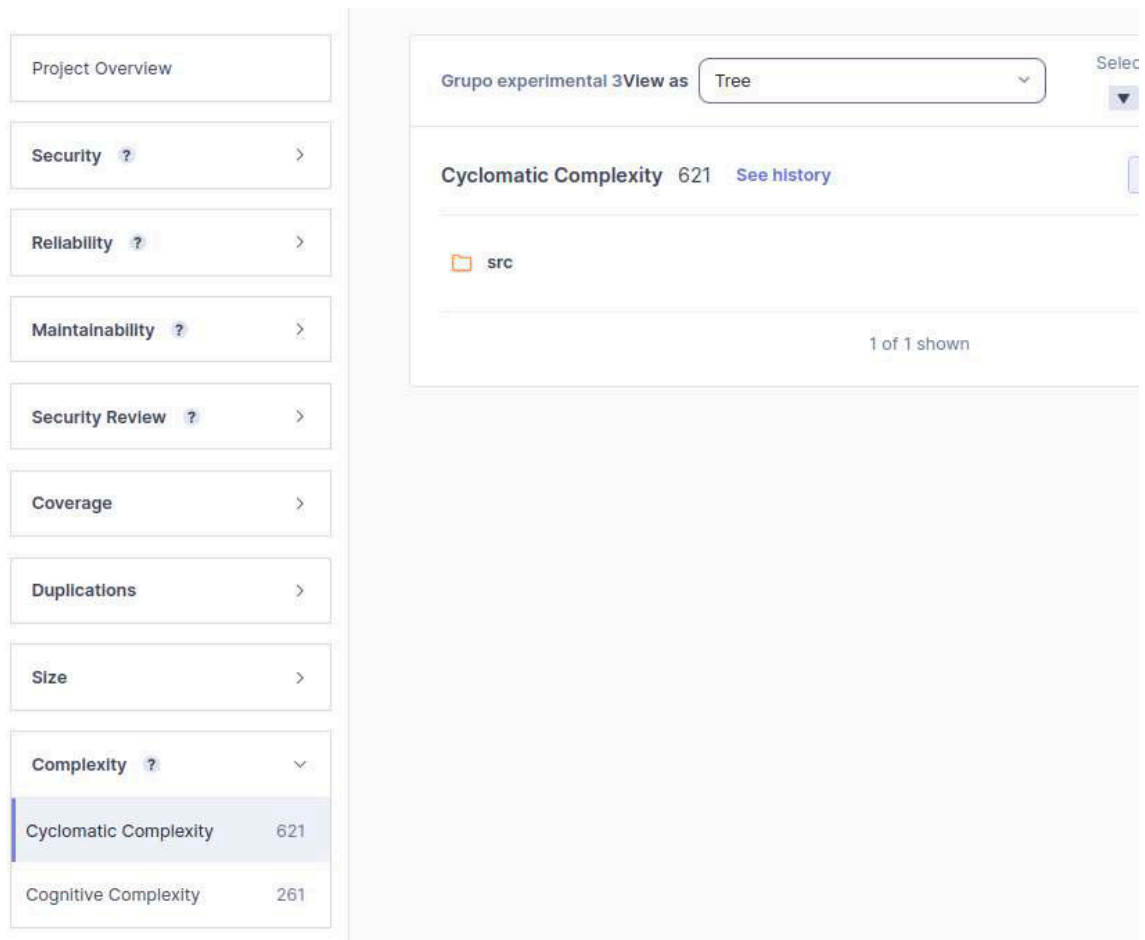


Figura 145 Complejidad Ciclomática desde el SonarQube caso 3

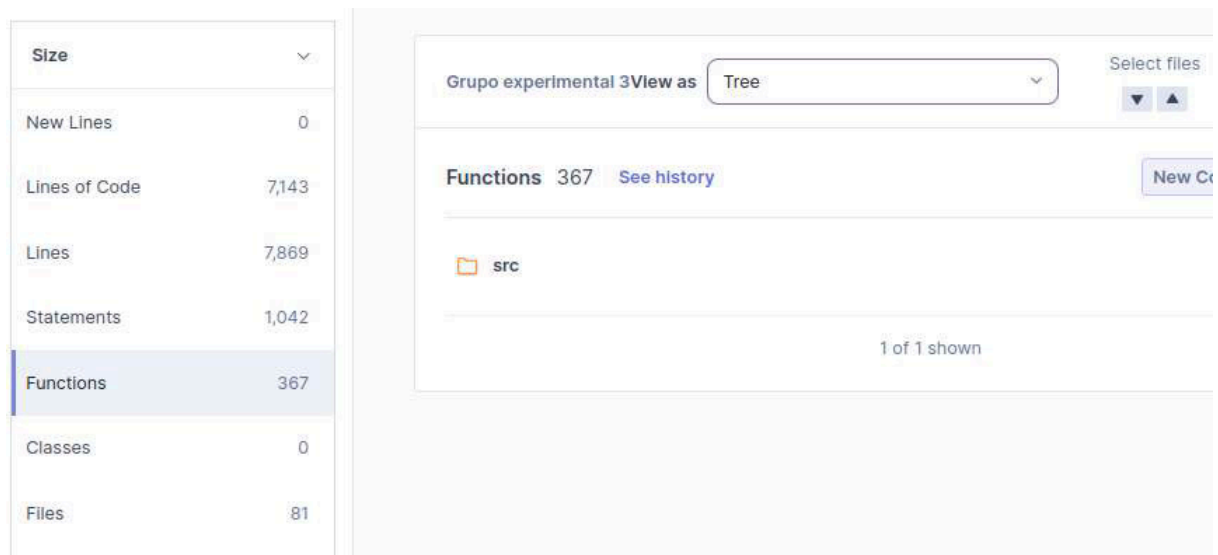


Figura 146 Número total de funciones desde el SonarQube caso 3

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M1.js
--- MS4M1: OBTE Seguir vinculo \(ctrl + clic\) JBE ---
URL de la API: http://localhost:9080/api/measures/component?component=Grupo-experimental&metricKeys=complexity.functions
-----
DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:
CX (Suma Total de Complejidad): 621
B (Total de Funciones Analizadas): 367
-----
MS4M1 (X Promedio del Proyecto): 1.692
Interpretación: Valor ideal cercano a 1 (Límite de Riesgo: 15).
-----
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 147 Número total de funciones desde el SonarQube del grupo experimental

- **Caso de estudio 4**

Donde el valor de A es 1.692 y ejecutando la Ecuación (6) obtuvimos $X=1.692+1 = 2.692$ como lo indica la Figura 148, 149 y 150.

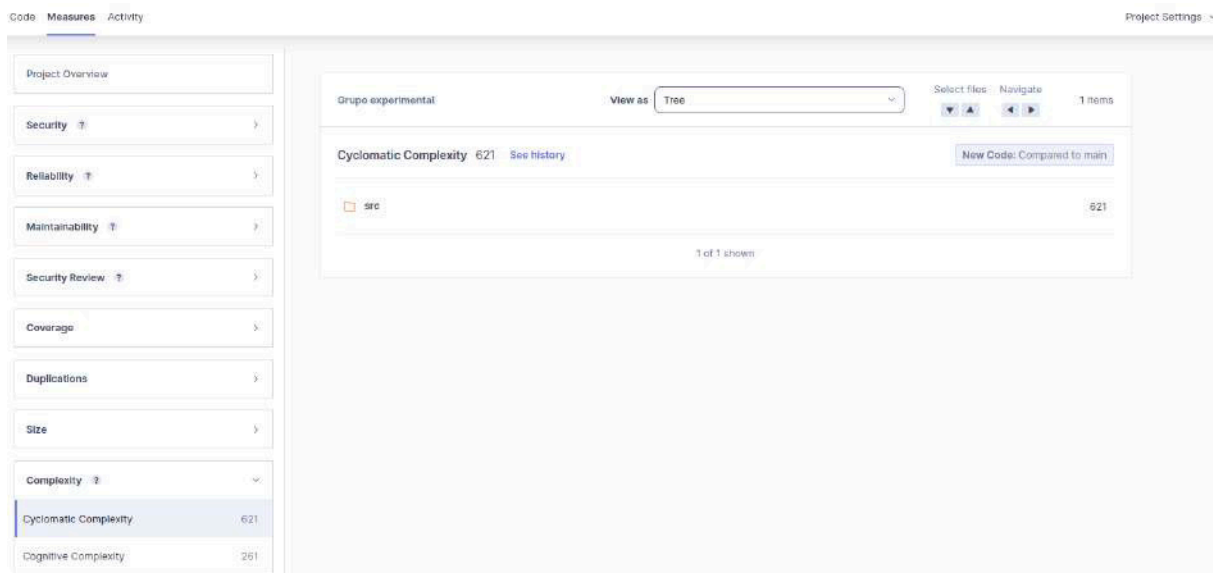


Figura 148 Complejidad Ciclomática desde el SonarQube

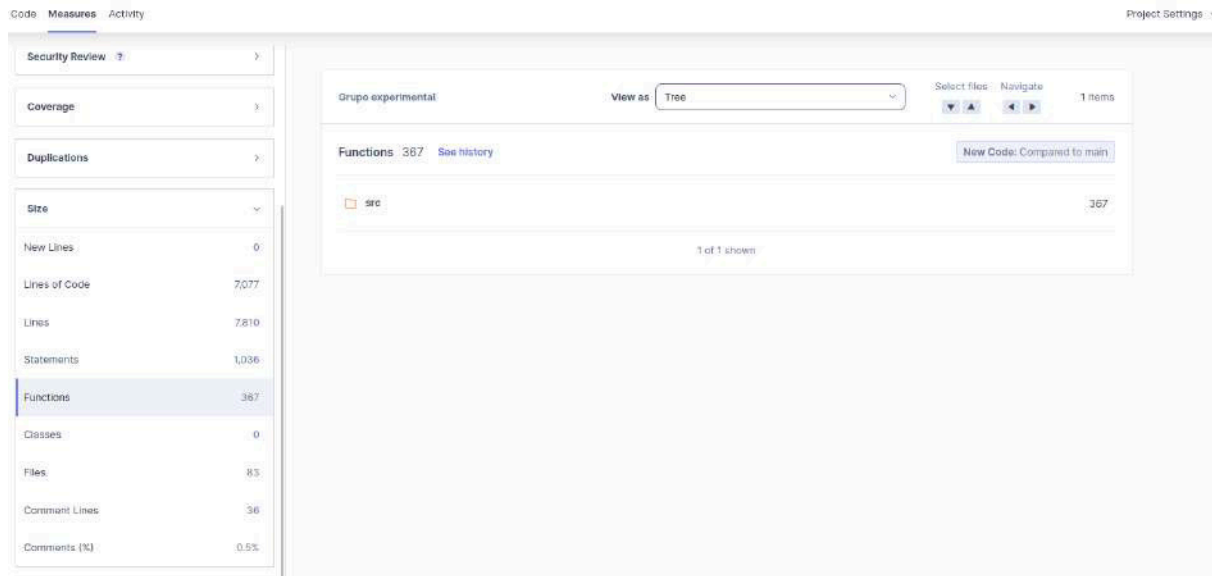


Figura 149 Número total de funciones desde el SonarQube caso 4

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M1.js
--- MS4M1: OBTE Seguir vinculo (ctrl + clic) JBE ---
URL de la API: http://localhost:9000/api/measures/component?component=Grupo-experimental&metricKeys=complexity.functions
-----
DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:
IX (Suma Total de Complejidad): 621
B (Total de Funciones Analizadas): 367
-----
MS4M1 (X Promedio del Proyecto): 1.692
Interpretación: Valor ideal cercano a 1 (Limite de Riesgo: 15).
-----
ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 150 Número total de funciones desde el SonarQube del grupo experimental caso 4

4.4.4.7. Profundidad de herencia - MT4M2

Tras la generación del análisis estático mediante el Primer Script, se ejecutó el Script final para obtener los datos y tener la profundidad de herencia del proyecto. Por esta razón se usó la Ecuación (7) como $X=A$, donde A es el número de jerarquías utilizadas para una determinada función.

- **Caso de estudio 1**

Dado que no existen clases entonces se deduce que $X=A=0$, es decir no existe profundidad de herencia en el proyecto.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2.js

--- MT4M2: ANALIZADOR DE PROFUNDIDAD DE HERENCIA (DIT) ---
Calculando el valor X ( $0 \leq X \leq 4$ ) para cada clase.

=====
¡Análisis de la Profundidad X completado!
Total de CLASES analizadas: 0
Reporte de valores X guardado en: **complexity_mt4m2_data.json**
=====

*** SIGUIENTE PASO ***
Ejecuta el script **MT4M2_final** para obtener el PROMEDIO MT4M2.
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2_final.js

--- CÁLCULO DEL PROMEDIO MT4M2 (DIT) ---
No hay datos en el archivo de entrada. Ejecuta el analizador primero.
```

Figura 151 Profundidad de herencia en el grupo de experimental caso 1

El archivo Json almacena las clases existentes y la profundidad que tiene, en este debido a que esta adaptación está dentro de una puerto privado es mantenida mediante una feature y ya no es necesario tenerlo en el proyecto sino que se instala como dependencias y extraerse a diferentes proyecto por esta razón no adquirimos un archivo json que es diferente al grupo de control como lo indica la Figura 151 y 152.

```
complexity_mt4m2_data.json
1
```

Figura 152 Archivo json de herencia en el grupo de experimental caso 1

- **Caso de estudio 2**

Dado que no existen clases entonces se deduce que $X=A=0$, es decir no existe profundidad de herencia en el proyecto así como lo indica la Figura 153 y 154.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2.js

--- MT4M2: ANALIZADOR DE PROFUNDIDAD DE HERENCIA (DIT) ---
Calculando el valor X ( $0 \leq X \leq 4$ ) para cada clase.

=====
¡Análisis de la Profundidad X completado!
Total de CLASES analizadas: 0
Reporte de valores X guardado en: **complexity_mt4m2_data.json**
=====

*** SIGUIENTE PASO ***
Ejecuta el script **MT4M2_final** para obtener el PROMEDIO MT4M2.
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2_final.js

--- CÁLCULO DEL PROMEDIO MT4M2 (DIT) ---
No hay datos en el archivo de entrada. Ejecuta el analizador primero.
ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 153 Profundidad de herencia en el grupo de experimental caso 2

```

complexity_mt4m2_data.json
1

```

Figura 154 Archivo json de herencia en el grupo de experimental caso 2

- **Caso de estudio 3**

Dado que no existen clases entonces se deduce que $X=A=0$, es decir no existe profundidad de herencia en el proyecto así como lo indica la Figura 155 y 156

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2.js

--- MT4M2: ANALIZADOR DE PROFUNDIDAD DE HERENCIA (DIT) ---
Calculando el valor X ( $0 \leq X \leq 4$ ) para cada clase.

=====
¡Análisis de la Profundidad X completado!
Total de CLASES analizadas: 0
Reporte de valores X guardado en: **complexity_mt4m2_data.json**
=====

*** SIGUIENTE PASO ***
Ejecuta el script **MT4M2_final** para obtener el PROMEDIO MT4M2.
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2_final.js

--- CÁLCULO DEL PROMEDIO MT4M2 (DIT) ---
No hay datos en el archivo de entrada. Ejecuta el analizador primero.
ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 155 Profundidad de herencia en el grupo de experimental caso 3

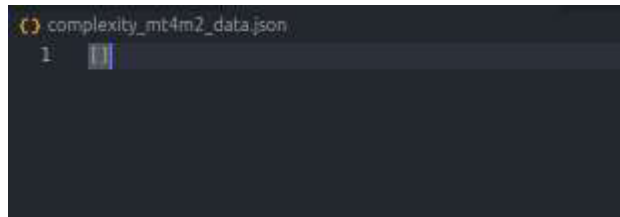


Figura 156 Archivo json de herencia en el grupo de experimental caso 3

- **Caso de estudio 4**

Dado que no existen clases entonces se deduce que $X=A=0$, es decir no existe profundidad de herencia en el proyecto así como lo indica la Figura 157 y 158.

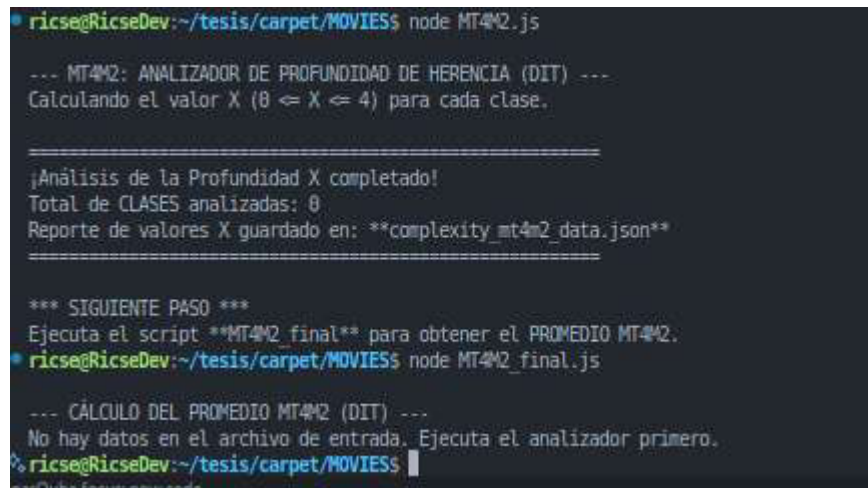


Figura 157 Profundidad de herencia en el grupo de experimental caso 4

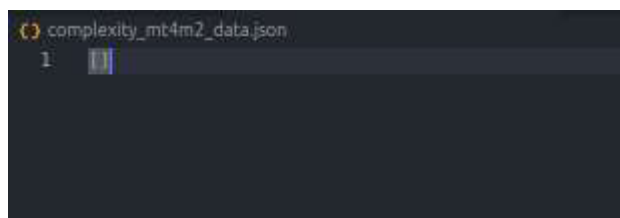


Figura 158 Archivo json de herencia en el grupo de experimental caso 4

4.4.4.8. Complejidad de modificación - MT4M4

Tras la generación del análisis del historial de commit realizados en Github para el grupo de control se obtuvo el Valor de A el número de modificaciones realizadas optamos por un tiempo promedio para fines prácticos de 8 horas que corresponden a un día laboral, se ejecutó

el Script final para obtener los datos y tener la complejidad de modificación. Por esta razón se usó la Ecuación (9) como $X=A/T$

- **Caso de estudio 1**

Dado que el valor de A es 32 y T = 8, entonces un valor de $X = 4$, así como lo indica la Figura 159.

```
ricsegRicseDev:~/tesis/carpet/MOVIES$ node MT4M.js

Procesando 33 commits con prefijo 'feat' o 'fix'...

=====
¡Análisis de Productividad y Densidad Completado!
=====

### MÉTRICAS CALCULADAS
- A1 (Líneas de Código Modificadas): **44186481 líneas**
- A2 (Total de Commits Válidos): **32 commits**
- T (Tiempo Asumido en horas laborales): **8 horas**

=====
**MT4M (X) (X) = A2 / T = 4.00 comit modificado /hora**
** 1534.2528125 líneas de código de comit modificado /segundos**
=====
```

Figura 159 Deuda técnica del grupo experimental caso 1

- **Caso de estudio 2**

Dado que el valor de A es 29 y T = 8, es decir $X = 3.63$, así como lo indica la Figura 160.

```
ricsegRicseDev:~/tesis/carpet/MOVIES$ node MT4M.js

Procesando 30 commits con prefijo 'feat' o 'fix'...

=====
¡Análisis de Productividad y Densidad Completado!
=====

### MÉTRICAS CALCULADAS
- A1 (Líneas de Código Modificadas): **44184823 líneas**
- A2 (Total de Commits Válidos): **29 commits**
- T (Tiempo Asumido en horas laborales): **8 horas**

=====
**MT4M (X) (X) = A2 / T = 3.63 comit modificado /hora**
** 1534.195243855555 líneas de código de comit modificado /segundos**
=====
```

Figura 160 Deuda técnica del grupo experimental caso 2

- **Caso de estudio 3**

Dado que el valor de A es 30 y T = 8, es decir $X = 3.75$, así como lo indica la Figura 161

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M.js

Procesando 31 commits con prefijo 'feat' o 'fix'...

=====
¡Análisis de Productividad y Densidad Completado!

### MÉTRICAS CALCULADAS
- A1 (Líneas de Código Modificadas): **44185825 líneas**
- A2 (Total de Commits Válidos): **30 commits**
- T (Tiempo Asumido en horas laborales): **8 horas**

=====
**MT4M (X) (X) = A2 / T = 3.75 comit modificado /hora**
** 1534.2388347222222 líneas de código de comit modificado /segundos**
=====

```

Figura 161 Deuda técnica del grupo experimental caso 3

- **Caso de estudio 4**

Dado que el valor de A es 25 y T = 8, es decir X = 3.13, así como lo indica la Figura 162.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M.js

Procesando 26 commits con prefijo 'feat' o 'fix'...

=====
¡Análisis de Productividad y Densidad Completado!

### MÉTRICAS CALCULADAS
- A1 (Líneas de Código Modificadas): **44182248 líneas**
- A2 (Total de Commits Válidos): **25 commits**
- T (Tiempo Asumido en horas laborales): **8 horas**

=====
**MT4M (X) (X) = A2 / T = 3.13 comit modificado /segundos**
** 1534.1855555555556 líneas de código de comit modificado /segundos**
=====

```

Figura 162 Deuda técnica del grupo experimental

4.4.4.9. Complejidad funcional del test - MT5M1

- **Caso de estudio 1**

De acuerdo con el análisis en SonarQube, se registró un 36% de cumplimiento en la ejecución de pruebas unitarias sobre las funciones definidas en el proyecto, así como lo indica la Figura 163 y 164.

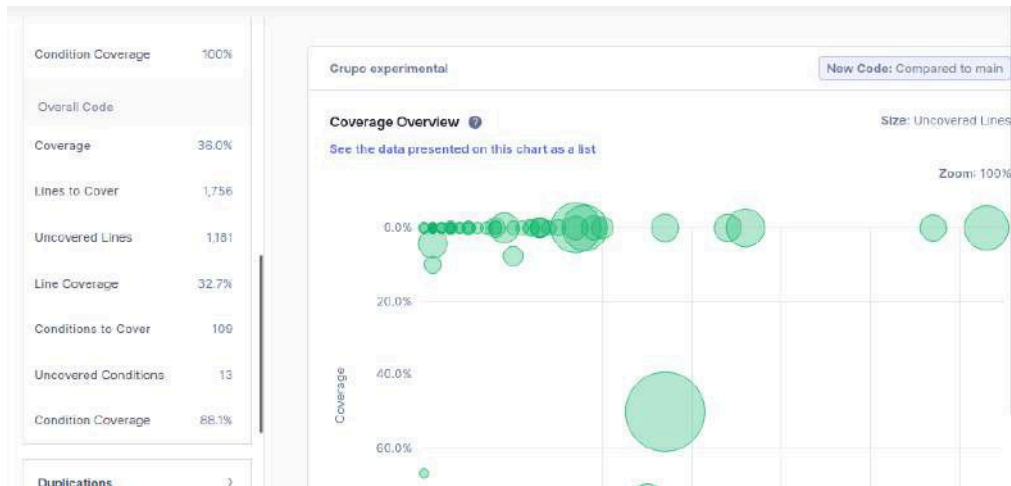


Figura 163 Cobertura del grupo experimental caso 1



Figura 164 Coverage resultante del grupo experimental caso 1

- Caso de estudio 2

De acuerdo con el análisis en SonarQube, se registró un 34.4% de cumplimiento en la ejecución de pruebas unitarias sobre las funciones definidas en el proyecto así como lo indica la Figura 165 y 166.



Figura 165 Cobertura del grupo experimental caso 2



Figura 166 Coverage resultante del grupo experimental caso 2

- **Caso de estudio 3**

De acuerdo con el análisis en SonarQube, se registró un 36% de cumplimiento en la ejecución de pruebas unitarias sobre las funciones definidas en el proyecto, así como lo indica la Figura 167 y 168.

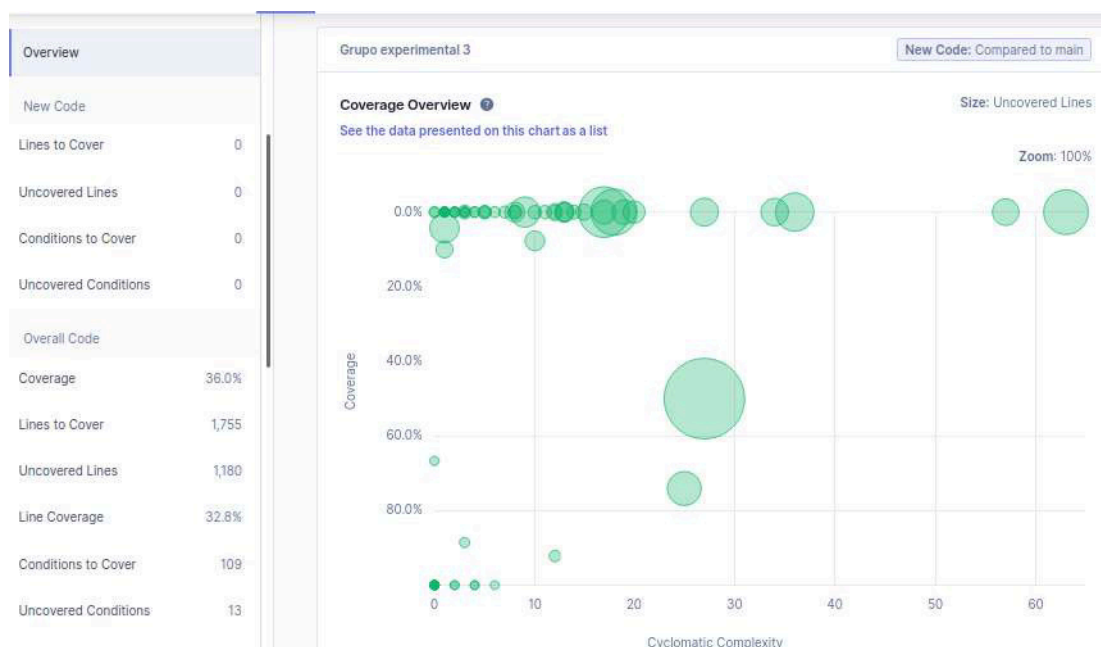


Figura 167 Cobertura del grupo experimental caso 3



Figura 168 Coverage resultante del grupo experimental caso 3

- **Caso de estudio 4**

De acuerdo con el análisis en SonarQube, se registró un 34.4% de cumplimiento en la ejecución de pruebas unitarias sobre las funciones definidas en el proyecto, así como lo indica la Figura 169 y 170..

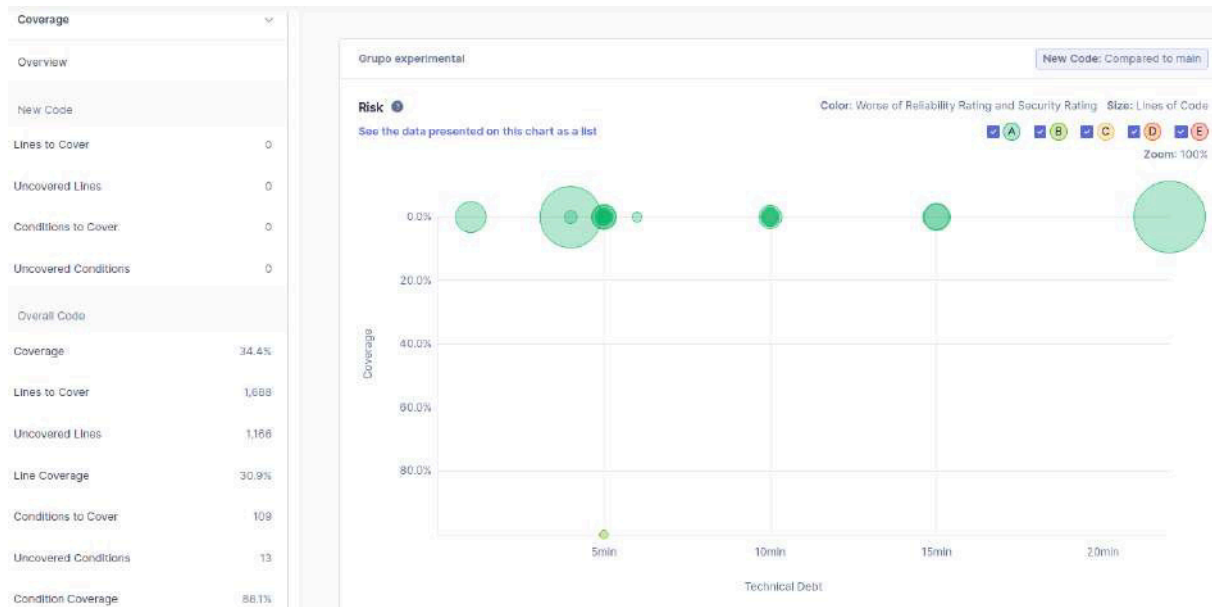


Figura 169 Cobertura del grupo experimental caso 4



Figura 170 Coverage resultante del grupo experimental caso 4

4.4.5. Análisis de la Arquitectura de 4 capas basada en Clean Architecture

En este caso, se analizó el proyecto y se clasificó según los grados de mantenibilidad del sistema. Los resultados indican que el proyecto se encuentra en buenas condiciones, como se muestra en la Figura 171.



Figura 171 Análisis de mantenibilidad del grupo de control

Para la validación del sistema se utilizó los datos obtenidos tras el análisis en SonarQube, como se indicó en la Tabla 22 y la Tabla 23.

Tabla 22 Datos del grupo control

Grupo de control	Caso 1	Caso 2	Caso 3	Caso 4	Caso promedio
Archivos	104	131	137	139	127.75
Total de líneas de código	18011	23491	24646	24641	22697.25

Tabla 23 Datos preprocesados del grupo control

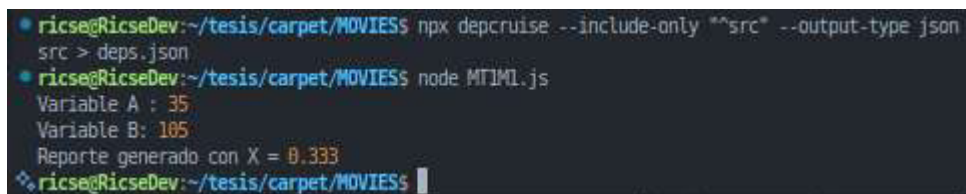
Grupo de control	Caso 1	Caso 2	Caso 3	Caso 4	Caso promedio
Archivos	80	80	83	84	81.75
Total de líneas de código	7062	7037	7143	7035	7069.25

4.4.5.1. Capacidad de condensación - MT1M1

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La capacidad de condensación, definida por la Ecuación (1) como $X=A/B$

- **Caso de estudio 1**

Entonces el valor de X es determinado utilizando los datos estructurales obtenidos, es decir la variable A como el número de componentes sin afectación es 35 y B como el número total de componentes específicos es 105. Al aplicar el cociente, se obtiene un valor de capacidad de condensación de 0.333, así como lo indica la Figura 172.

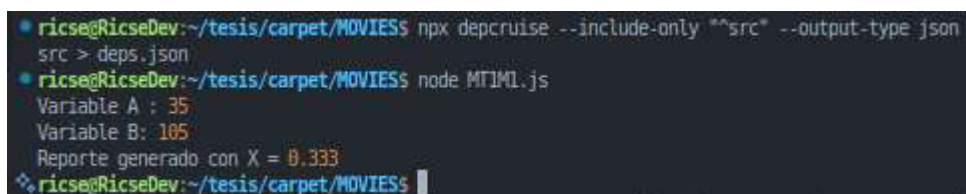


```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "**src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT1M1.js
Variable A : 35
Variable B: 105
Reporte generado con X = 0.333
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 172 Capacidad de condensación del grupo de control caso 1

- **Caso de estudio 2**

Entonces el valor de X es determinado utilizando los datos estructurales obtenidos, es decir la variable A como el número de componentes sin afectación es 35 y B como el número total de componentes específicos es 105. Al aplicar el cociente, se obtiene un valor de capacidad de condensación de 0.333, así como lo indica la Figura 173.



```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "**src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT1M1.js
Variable A : 35
Variable B: 105
Reporte generado con X = 0.333
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 173 Capacidad de condensación del grupo de control caso 2

- **Caso de estudio 3**

Entonces el valor de X es determinado utilizando los datos estructurales obtenidos, es decir la variable A como el número de componentes sin afectación es 38 y B como el número total de componentes específicos es 112. Al aplicar el cociente, se obtiene un valor de capacidad de condensación de 0.339, así como lo indica la Figura 174.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT1M1.js
Variable A : 38
Variable B: 112
Reporte generado con X = 0.339
ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 174 Capacidad de condensación del grupo de control caso 3

- **Caso de estudio 4**

Entonces el valor de X es determinado utilizando los datos estructurales obtenidos, es decir la variable A como el número de componentes sin afectación es 39 y B como el número total de componentes específicos es 114. Al aplicar el cociente, se obtiene un valor de capacidad de condensación de 0.342, así como lo indica la Figura 175

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node mt1m1.js
Variable A : 39
Variable B: 114
Reporte generado con X = 0.342
ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 175 Capacidad de condensación del grupo de control caso 4

4.4.5.2. Acoplamiento- MT1M2

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La capacidad de acoplamiento, definida por la Ecuación (2) como $X=A$, como el número de relaciones que tiene una función sobre otras clases o módulos.

- **Caso de estudio 1**

El valor de X se define como el valor de A, es decir un valor de 2.038, así como lo indica la Figura 176

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT1M2.js
Acoplamiento de Clases (X) basado en promedio por módulo: 2.038
ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 176 Acoplamiento del grupo de control caso 1

- **Caso de estudio 2**

El valor de X se define como el valor de A, es decir un valor de 2.113 , así como lo indica la Figura 177

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT1M2.js
Acoplamiento de Clases (X) basado en promedio por módulo: 2.113
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 177 Acoplamiento del grupo de control caso 2

- **Caso de estudio 3**

El valor de X se define como el valor de A, es decir un valor de 2.045, así como lo indica la Figura 178

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT1M2.js
Acoplamiento de Clases (X) basado en promedio por módulo: 2.045
ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 178 Acoplamiento del grupo de control caso 3

- **Caso de estudio 4**

El valor de X se define como el valor de A, es decir un valor de 2.132, así como lo indica la Figura 179.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src"
--output-type dot src > graph.dot
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT1M2.js
Acoplamiento de Clases (X) basado en promedio por módulo: 2.132
```

Figura 179 Acoplamiento del grupo de control caso 4

4.4.5.3. Ejecución de reutilización - MT2M1

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La ejecución de reutilización es definida por la Ecuación (3) como $X=A/B$, donde A es el número de ítems

reutilizados y B el total de ítems reutilizables, siendo el caso de estudio experimental las carpetas utils y adapters

- **Caso de estudio 1**

El valor de X se define $A = 15$ entre $B = 18$, dando como resultado 0.833, así como lo indica la Figura 180.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT2M1.js

--- MS2M1: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---
RUTAS ANALIZADAS (Universo B): src/adapters, src/lib/
-----
B (Total de Módulos Candidatos a Reutilizar): 18
A (Módulos Reutilizados en el Proyecto): 15
-----
MS2M1 (X): 0.833 (El valor más cercano a 1 es el mejor)
```

***Figura 180** Ejecución de reutilización del grupo de control caso 1*

- **Caso de estudio 2**

El valor de X se define $A = 29$ entre $B = 31$, dando como resultado 0.935, así como lo indica la Figura 181

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT2M1.js

--- MS2M1: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---
RUTAS ANALIZADAS (Universo B): src/adapters, src/core, src/lib/
-----
B (Total de Módulos Candidatos a Reutilizar): 31
A (Módulos Reutilizados en el Proyecto): 29
-----
MS2M1 (X): 0.935 (El valor más cercano a 1 es el mejor)
```

***Figura 181** Ejecución de reutilización del grupo de control caso 2*

- **Caso de estudio 3**

El valor de X se define $A = 32$ entre $B = 34$, dando como resultado 0.941, así como lo indica la Figura 182.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT2M1.js
--- MS2M1: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---
RUTAS ANALIZADAS (Universo B): src/adapters, src/core, src/lib/
-----
B (Total de Módulos Candidatos a Reutilizar): 34
A (Módulos Reutilizados en el Proyecto): 32
-----
MS2M1 (X): 0.941 (El valor más cercano a 1 es el mejor)
-----

```

Figura 182 Ejecución de reutilización del grupo de control caso 3

- **Caso de estudio 4**

El valor de X se define $A = 33$ entre $B = 35$, dando como resultado 0.943, así como lo indica la Figura 183.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT2M1.js
--- MS2M1: EJECUCIÓN DE LA REUTILIZACIÓN (X = A/B) ---
RUTAS ANALIZADAS (Universo B): src/adapters, src/core, src/lib/
-----
B (Total de Módulos Candidatos a Reutilizar): 35
A (Módulos Reutilizados en el Proyecto): 33
-----
MS2M1 (X): 0.943 (El valor más cercano a 1 es el mejor)
-----

```

Figura 183 Ejecución de reutilización del grupo de control caso 4

4.4.5.4. Capacidad de seguimiento de auditoría - MT3M1

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla, se procedió a ejecutar el script de cálculo de la métrica. La capacidad de seguimiento de auditoría es definida por la Ecuación (4) como $X=A/B$, donde A es el número de datos realmente registrados y B el número de datos esperados a ser grabados para controlar el estado del sistema

- **Caso de estudio 1**

El valor de X se define $A = 11$ entre $B = 24$, dando como resultado 0.458, así como lo indica la Figura 184.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M1.js

--- MS3M1: CAPACIDAD DE LA PISTA DE AUDITORIA (X = A/B) ---
RUTAS CRITICAS ANALIZADAS (Universo B): src/core/application/, src/adapters/
KEYWORDS DE AUDITORIA BUSCADAS (Define A): logger, audit, log, saveLog, console.error
-----
B (Total de Módulos Críticos Esperados): 24
A (Módulos Críticos que Contienen Auditoria): 11
-----
MS3M1 (X): 0.458 (El valor más cercano a 1 es el mejor)

```

Figura 184 Capacidad de seguimiento de auditoría del grupo control caso 1

- **Caso de estudio 2**

El valor de X se define $A = 11$ entre $B = 24$, dando como resultado 0.458, así como lo indica la Figura 185.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M1.js

--- MS3M1: CAPACIDAD DE LA PISTA DE AUDITORIA (X = A/B) ---
RUTAS CRITICAS ANALIZADAS (Universo B): src/core/application/, src/adapters/
KEYWORDS DE AUDITORIA BUSCADAS (Define A): logger, audit, log, saveLog, console.error
-----
B (Total de Módulos Críticos Esperados): 24
A (Módulos Críticos que Contienen Auditoria): 11
-----
MS3M1 (X): 0.458 (El valor más cercano a 1 es el mejor)

```

Figura 185 Capacidad de seguimiento de auditoría del grupo control caso 2

- **Caso de estudio 3**

El valor de X se define $A = 11$ entre $B = 27$, dando como resultado 0.407, así como lo indica la Figura 186

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "^src" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M1.js

--- MS3M1: CAPACIDAD DE LA PISTA DE AUDITORIA (X = A/B) ---
RUTAS CRITICAS ANALIZADAS (Universo B): src/core/application/, src/adapters/
KEYWORDS DE AUDITORIA BUSCADAS (Define A): logger, audit, log, saveLog, console.error
-----
B (Total de Módulos Críticos Esperados): 27
A (Módulos Críticos que Contienen Auditoria): 11
-----
MS3M1 (X): 0.407 (El valor más cercano a 1 es el mejor)

```

Figura 186 Capacidad de seguimiento de auditoría del grupo control caso 3

- **Caso de estudio 4**

El valor de X se define $A = 12$ entre $B = 28$, dando como resultado 0.429, así como lo indica la Figura 187.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "**src*" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M1.js

--- MS3M1: CAPACIDAD DE LA PISTA DE AUDITORÍA (X = A/B) ---
RUTAS CRÍTICAS ANALIZADAS (Universo B): src/core/application/, src/adapters/
KEYWORDS DE AUDITORÍA BUSCADAS (Define A): logger, audit, log, saveLog, console.error
-----
B (Total de Módulos Críticos Esperados): 28
A (Módulos Críticos que Contienen Auditoría): 12
-----
MS3M1 (X): 0.429 (El valor más cercano a 1 es el mejor)
```

Figura 187 Capacidad de seguimiento de auditoría del grupo control caso 4

4.4.5.5. Diagnóstico de funciones- MT3M2

Tras la generación del grafo de dependencias en el file *deps.json* mediante el comando detallado en la Tabla 19, se procedió a ejecutar el script de cálculo de la métrica. La capacidad de seguimiento de auditoría es definida por la Ecuación (5) como $X=A/B$, donde A es el número de funciones diagnosticadas e implementadas y B número de diagnóstico requerido, siendo el caso de estudio experimental las carpetas *utils* y *adapter*.

- **Caso de estudio 1**

El valor de X se define $A = 20$ entre $B = 33$, dando como resultado 0.606, así como lo indica la Figura 188.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "**src*" --output-type json
src > deps.json
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M2.js

--- MS3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---
RUTAS ANALIZADAS (Universo B - Requeridas): src/core/application/repository/, src/core/application/services/, src/core/domain/, src/ui/pages/, src/adapters
KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A): ping, health, status, await, throw, try, console, logger, isReady
-----
B (Total de Módulos de Diagnóstico Requeridos): 33
A (Módulos de Diagnóstico Implementados): 20
-----
MS3M2 (X): 0.606 (El valor más cercano a 1 es el mejor)
```

Figura 188 Diagnóstico de funciones del grupo de control caso 1

- **Caso de estudio 2**

El valor de X se define $A = 20$ entre $B = 33$, dando como resultado 0.606, así como lo indica la Figura 189

```
ricseg@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "**src*" --output-type json
src > deps.json
ricseg@RicseDev:~/tesis/carpet/MOVIES$ node MT3M2.js

--- MS3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---
RUTAS ANALIZADAS (Universo B - Requeridas): src/core/application/repository/, src/core/applic
ation/services/, src/core/domain/, src/ui/pages/, src/adapters
KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A): ping, health, status, await, throw, try, cons
ole, logger, isReady
-----
B (Total de Módulos de Diagnóstico Requeridos): 33
A (Módulos de Diagnóstico Implementados): 20
-----
MS3M2 (X): 0.606 (El valor más cercano a 1 es el mejor)
```

Figura 189 Diagnóstico de funciones del grupo de control caso 2

- **Caso de estudio 3**

El valor de X se define $A = 21$ entre $B = 36$, dando como resultado 0.583, así como lo indica la Figura 190.

```
ricseg@RicseDev:~/tesis/carpet/MOVIES$ npx depcruise --include-only "**src*" --output-type json
src > deps.json
ricseg@RicseDev:~/tesis/carpet/MOVIES$ node MT3M2.jsj

--- MS3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---
RUTAS ANALIZADAS (Universo B - Requeridas): src/core/application/repository/, src/core/applic
ation/services/, src/core/domain/, src/ui/pages/, src/adapters
KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A): ping, health, status, await, throw, try, cons
ole, logger, isReady
-----
B (Total de Módulos de Diagnóstico Requeridos): 36
A (Módulos de Diagnóstico Implementados): 21
-----
MS3M2 (X): 0.583 (El valor más cercano a 1 es el mejor)
```

Figura 190 Diagnóstico de funciones del grupo de control caso 3

- **Caso de estudio 4**

El valor de X se define $A = 13$ entre $B = 20$, dando como resultado 0.65, así como lo indica la Figura 191.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT3M2.js

--- MS3M2: DIAGNÓSTICO DE FUNCIONES SUFICIENTES (X = A/B) ---
RUTAS ANALIZADAS (Universo B - Requeridas): src/core/application/repository/, src/core/appli
cation/services/, src/core/domain/, src/ui/pages/, src/adapters
KEYWORDS DE IMPLEMENTACIÓN BUSCADAS (Define A): ping, health, status, await, throw, try, con
sole, logger, isReady
-----
B (Total de Módulos de Diagnóstico Requeridos): 20
A (Módulos de Diagnóstico Implementados): 13
-----
MS3M2 (X): 0.65 (El valor más cercano a 1 es el mejor)
-----

```

Figura 191 Diagnóstico de funciones del grupo de control caso 4

4.4.5.6. Complejidad ciclomática - MT4M1

Tras la generación del análisis estático en SonarQube, se ejecutó el Script para obtener los datos y tener el número de funciones condicionales respecto a la complejidad ciclomática. La complejidad ciclomática por la Ecuación (6) como $X=A+1$, donde A es el número de funciones condicionales respecto a la complejidad ciclomática definida por la ecuación (6.1)

- **Caso de estudio 1**

. Donde el valor de A es 1.352 y ejecutando la Ecuación (6) obtuvimos $X=1.352+1 = 2.352$, así como lo indica la Figura 192, 193 y 194.

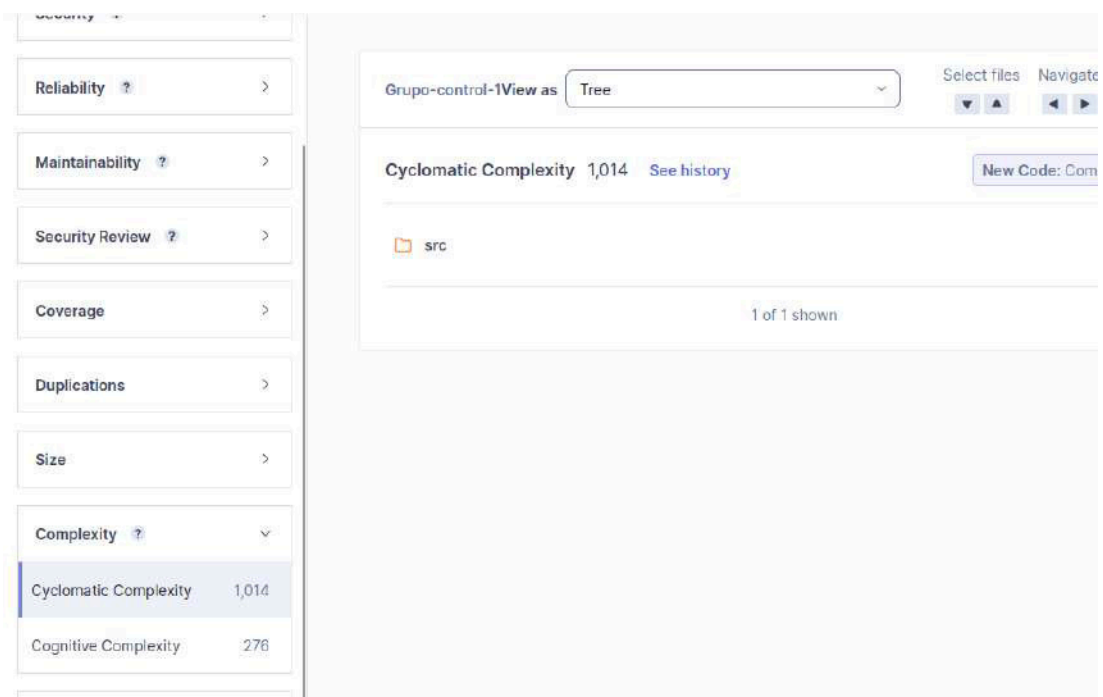


Figura 192 Complejidad Ciclomática desde el SonarQube del grupo de control caso 1

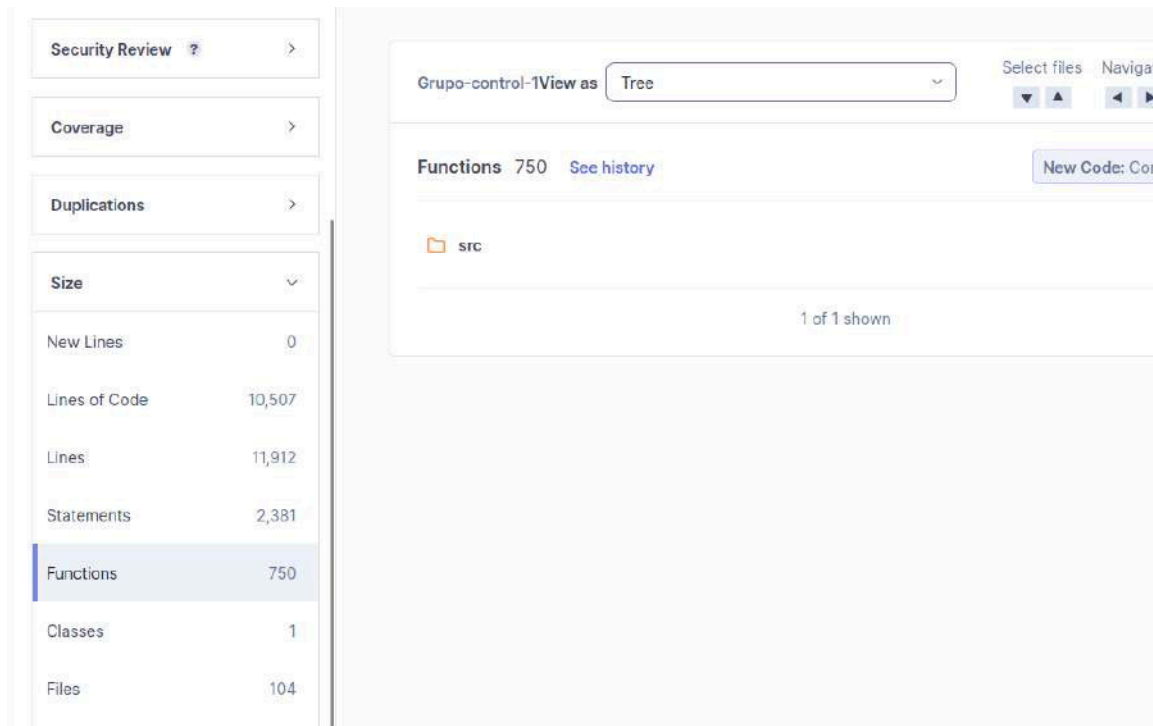


Figura 193 Número total de funciones desde el SonarQube del grupo de control caso 1

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M1.js
--- MS4M1: OBTENIENDO DATOS DE SONARQUBE ---
URL de la API: http://localhost:9000/api/measures/component?component=Grupo-control-1&metricKeys=complexity,functions
-----
DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:
CICX (Suma Total de Complejidad): 1014
CICB (Total de Funciones Analizadas): 750
-----
MS4M1 (X Promedio del Proyecto): 1.352
Interpretación: Valor ideal cercano a 1 (Limite de Riesgo: 15).
```

Figura 194 Complejidad ciclomática del grupo de control caso 1

- **Caso de estudio 2**

Donde el valor de A es 1.344 y ejecutando la Ecuación (6) obtuvimos $X=1.344+1 = 2.344$, así como lo indica la Figura 195, 196 y 197.

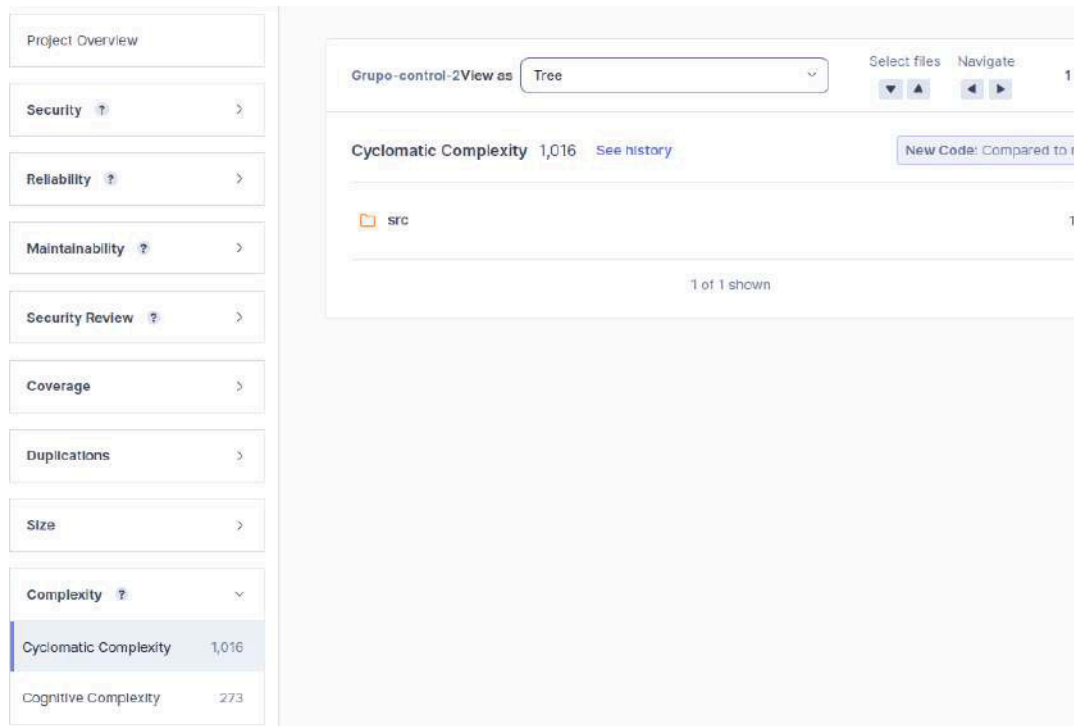


Figura 195 Complejidad Ciclomática desde el SonarQube del grupo de control caso 2

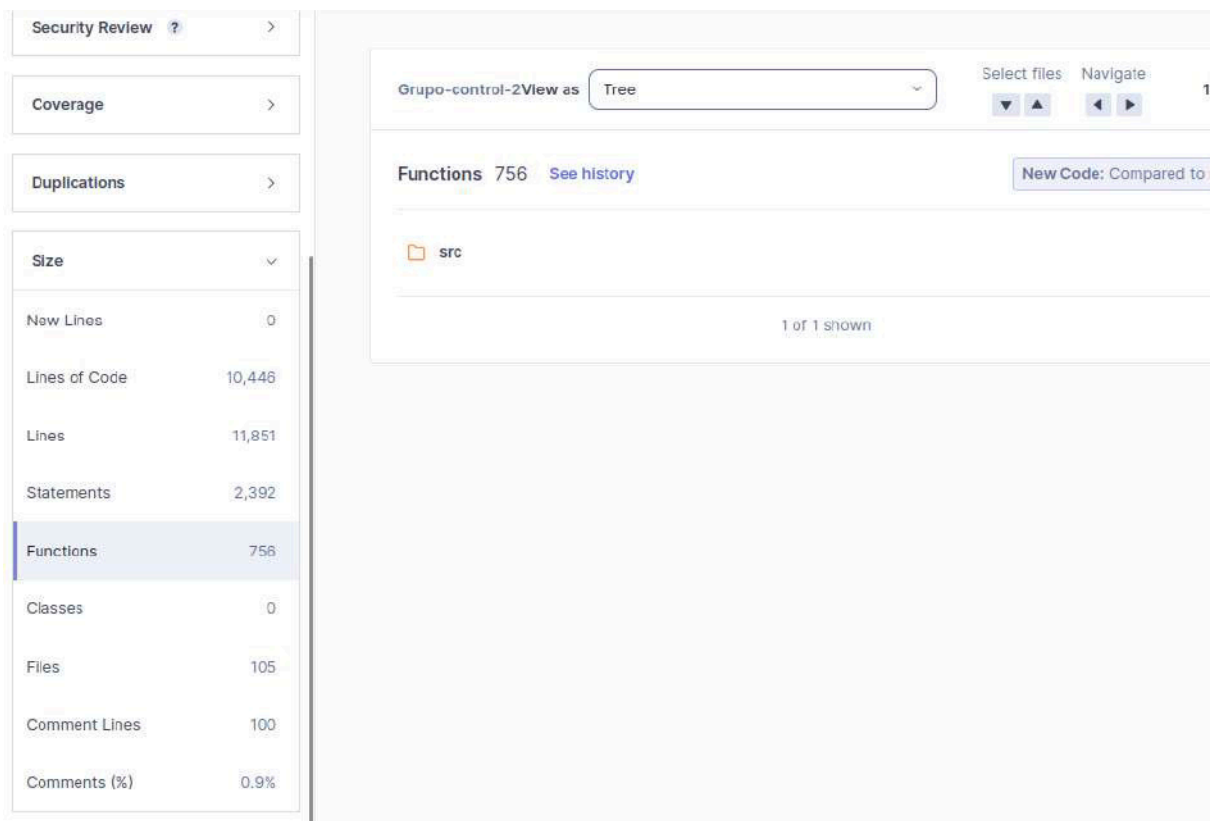


Figura 196 Número total de funciones desde el SonarQube del grupo de control caso 2

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M1.js

--- MS4M1: OBTENIENDO DATOS DE SONARQUBE ---
URL de la API: http://localhost:9000/api/measures/component?component=Grupo-control-2&metricKeys=complexity,functions
-----
DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:
ΣX (Suma Total de Complejidad): 1816
B (Total de Funciones Analizadas): 756
-----
MS4M1 (X Promedio del Proyecto): 1.344
Interpretación: Valor ideal cercano a 1 (Límite de Riesgo: 15).
```

Figura 197 Complejidad ciclomática del grupo de control caso 2

- **Caso de estudio 3**

Donde el valor de A es 1.298 y ejecutando la Ecuación (6) obtuvimos $X=1.298+1 = 2.298$, así como lo indica la Figura 198, 199 y 200.

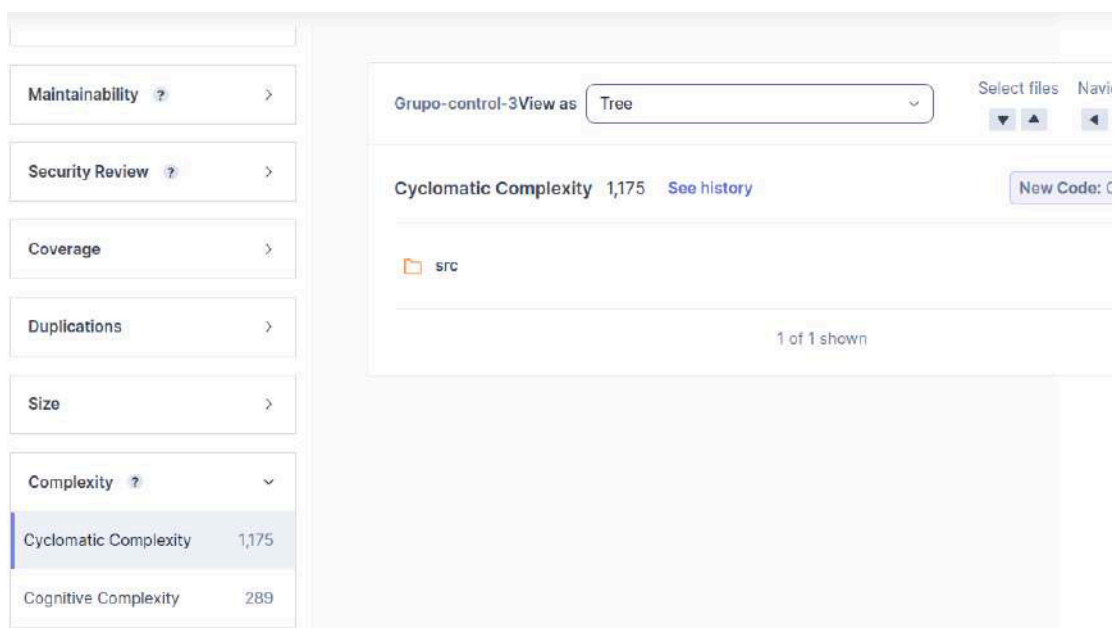


Figura 198 Complejidad Ciclomática desde el SonarQube del grupo de control caso 3

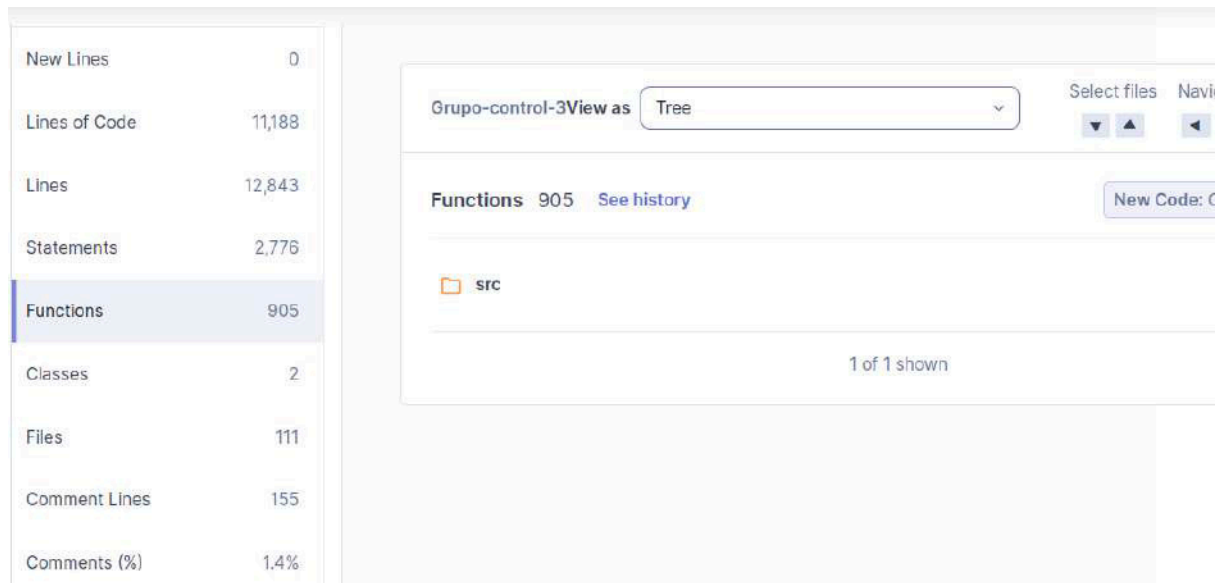


Figura 199 Número total de funciones desde el SonarQube del grupo de control caso 3

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M1.js
--- MS4M1: OBTENIENDO DATOS DE SONARQUBE ---
URL de la API: http://localhost:9000/api/measures/component?component=Grupo-control-3&met
eys=complexity,functions
-----
DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:
EX (Suma Total de Complejidad): 1175
B (Total de Funciones Analizadas): 905
-----
MS4M1 (X Promedio del Proyecto): 1.298
Interpretación: Valor ideal cercano a 1 (Límite de Riesgo: 15).
```

Figura 200 Complejidad ciclomática del grupo de control caso 3

- **Caso de estudio 4**

Donde el valor de A es 1.294 y ejecutando la Ecuación (6) obtuvimos $X=1.294+1 = 2.294$, así como lo indica la Figura 201, 202 y 203.

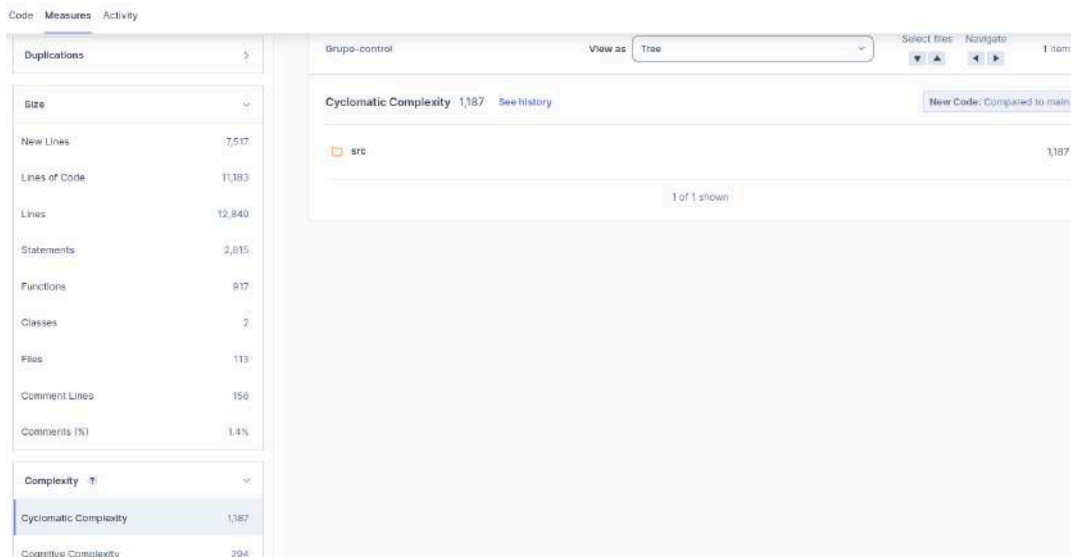


Figura 201 Complejidad Ciclomática desde el SonarQube del grupo de control caso 4

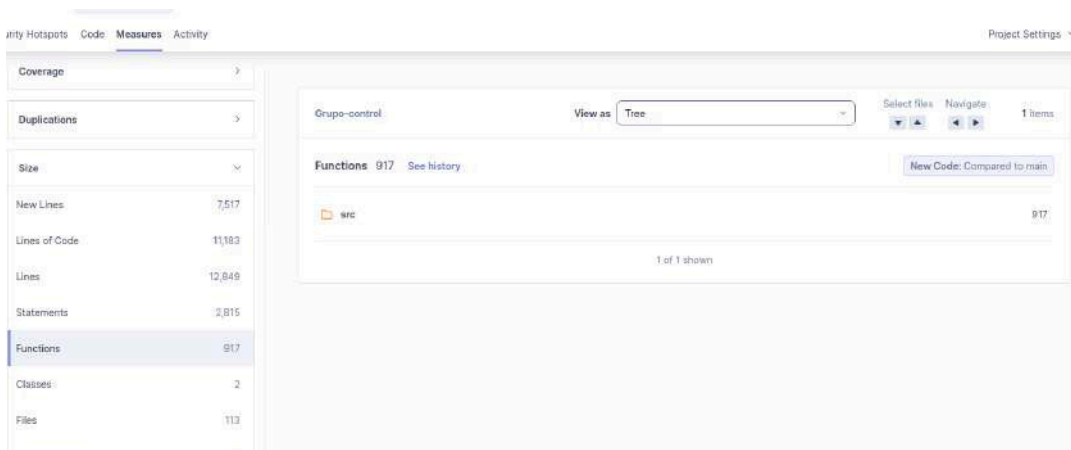


Figura 202 Número total de funciones desde el SonarQube del grupo de control caso 4

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M1.js

--- MS4M1: OBTENIENDO DATOS DE SONARQUBE ---
URL de la API: http://localhost:9000/api/measures/component?component=Grupo-control&metricKeys=complexity,functions

DATOS OBTENIDOS DEL REPORTE DE SONARQUBE:
EX (Suma Total de Complejidad): 1187
B (Total de Funciones Analizadas): 917

MS4M1 (X Promedio del Proyecto): 1.294
Interpretación: Valor ideal cercano a 1 (Límite de Riesgo: 15).

ricse@RicseDev:~/tesis/carpet/MOVIES$

```

Figura 203 Complejidad ciclomática del grupo de control caso 4

4.4.5.7. Profundidad de herencia - MT4M2

Tras la generación del análisis estático mediante el Primer Script, se ejecutó el Script final para obtener los datos y tener la profundidad de herencia del proyecto. Por esta razón se usó la Ecuación (7) como $X=A$, donde A es el número de jerarquías utilizadas para una determinada función.

- **Caso de estudio 1**

Dado que no existen clases entonces se deduce que $X=A=0$, es decir no existe profundidad de herencia en el proyecto, así como lo indica la Figura 204 y la Figura 205.

```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2.js
--- MT4M2: ANALIZADOR DE PROFUNDIDAD DE HERENCIA (DIT) ---
Calculando el valor X ( $0 \leq X \leq 4$ ) para cada clase.

=====
¡Análisis de la Profundidad X completado!
Total de CLASES analizadas: 1
Reporte de valores X guardado en: **complexity_mt4m2_data.json**
=====

*** SIGUIENTE PASO ***
Ejecuta el script **MT4M2_final** para obtener el PROMEDIO MT4M2.
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2_final.js
--- CÁLCULO DEL PROMEDIO MT4M2 (DIT) ---

-----
RESULTADOS
-----
Total de Clases (B): 1
Suma de Profundidades ( $\Sigma X$ ): 0
**Promedio MT4M2 (X): 0.00**
-----
```

Figura 204 Profundidad de herencia en el grupo de control caso 1

```
complexity_mt4m2_data.json > ...
{
  "file": "src/adapters/theme/localStorageThemeAdapter.ts",
  "className": "LocalStorageThemeAdapter",
  "inheritanceDepth": 0,
  "isInherited": false
}
```

Figura 205 Archivo json de herencia en el grupo de control caso 1

- **Caso de estudio 2**

Dado que no existen clases entonces se deduce que $X=A=0$, es decir no existe profundidad de herencia en el proyecto, así como lo indica la Figura 206 y la Figura 207.

```
ricsegRicseDev:~/tesis/carpet/MOVIES$ node MT4M2.js

--- MT4M2: ANALIZADOR DE PROFUNDIDAD DE HERENCIA (DIT) ---
Calculando el valor X ( $0 \leq X \leq 4$ ) para cada clase.

=====
¡Análisis de la Profundidad X completado!
Total de CLASES analizadas: 0
Reporte de valores X guardado en: **complexity_mt4m2_data.json**
=====

*** SIGUIENTE PASO ***
Ejecuta el script **MT4M2_final** para obtener el PROMEDIO MT4M2.
ricsegRicseDev:~/tesis/carpet/MOVIES$ node MT4M2_final.js

--- CÁLCULO DEL PROMEDIO MT4M2 (DIT) ---
No hay datos en el archivo de entrada. Ejecuta el analizador primero.
```

Figura 206 Profundidad de herencia en el grupo de control caso 2

```
complexity_mt4m2_data.json
1 | 11
```

Figura 207 Archivo json de herencia en el grupo de control caso 2

- **Caso de estudio 3**

Dado que no existen clases entonces se deduce que $X=A=0$, es decir no existe profundidad de herencia en el proyecto, así como lo indica la Figura 208 y la Figura 209.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2.js

--- MT4M2: ANALIZADOR DE PROFUNDIDAD DE HERENCIA (DIT) ---
Calculando el valor X ( $0 \leq X \leq 4$ ) para cada clase.

=====
¡Análisis de la Profundidad X completado!
Total de CLASES analizadas: 2
Reporte de valores X guardado en: **complexity_mt4m2_data.json**
=====

*** SIGUIENTE PASO ***
Ejecuta el script **MT4M2_final** para obtener el PROMEDIO MT4M2.
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2_final.js

--- CÁLCULO DEL PROMEDIO MT4M2 (DIT) ---

-----
RESULTADOS
-----
Total de Clases (B): 2
Suma de Profundidades ( $\sum X$ ): 0
**Promedio MT4M2 (X): 0.00**
-----

```

Figura 208 Profundidad de herencia en el grupo de control caso 3

```

complexity_mt4m2_data.json > ...
{
  "file": "src/adapters/theme/localStorageThemeAdapter.ts",
  "className": "LocalStorageThemeAdapter",
  "inheritanceDepth": 0,
  "isInherited": false
},
{
  "file": "src/adapters/theme/themeService.ts",
  "className": "ThemeService",
  "inheritanceDepth": 0,
  "isInherited": false
}

```

Figura 209 Archivo json de herencia en el grupo de control caso 3

- **Caso de estudio 4**

Dado que no existen clases entonces se deduce que $X=A=0$, es decir no existe profundidad de herencia en el proyecto, así como lo indica la Figura 210 y la Figura 211.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2.js

--- MT4M2: ANALIZADOR DE PROFUNDIDAD DE HERENCIA (DIT) ---
Calculando el valor X ( $0 \leq X \leq 4$ ) para cada clase.

=====
¡Análisis de la Profundidad X completado!
Total de CLASES analizadas: 2
Reporte de valores X guardado en: **complexity_mt4m2_data.json**
=====

*** SIGUIENTE PASO ***
Ejecuta el script **MT4M2_final** para obtener el PROMEDIO MT4M2.
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M2_final.js

--- CÁLCULO DEL PROMEDIO MT4M2 (DIT) ---

-----
RESULTADOS
-----

Total de Clases (B): 2
Suma de Profundidades ( $\Sigma X$ ): 0
**Promedio MT4M2 ( $\bar{X}$ ): 0.00**

```

Figura 210 Profundidad de herencia en el grupo de control caso 4

```

complexity_mt4m2_data.json
{
  [
    {
      "file": "src/adapters/theme/localStorageThemeAdapter.ts",
      "className": "LocalStorageThemeAdapter",
      "inheritanceDepth": 0,
      "isInherited": false
    },
    {
      "file": "src/adapters/theme/themeService.ts",
      "className": "ThemeService",
      "inheritanceDepth": 0,
      "isInherited": false
    }
  ]
}

```

Figura 211 Archivo json de herencia en el grupo de control caso 4

4.4.5.8. Complejidad de modificación - MT4M4

Tras la generación del análisis del historial de commit realizados en Github para el grupo de control se obtuvo el Valor de A el número de modificaciones realizadas optamos por un tiempo promedio para fines prácticos de 8 horas que corresponden a un día laboral, se ejecutó el Script final para obtener los datos y tener la complejidad de modificación. Por esta razón se usó la Ecuación (9) como $X=A/T$

- **Caso de estudio 1**

Dado que el valor de A es 17 y T = 8, es decir $X = 2.250$, así como lo indica la Figura 212.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M4.js

Procesando 19 commits con prefijo 'feat' o 'fix'...

=====
¡Análisis de Productividad y Densidad Completado!

### MÉTRICAS CALCULADAS
- A1 (Líneas de Código Modificadas): **1284775 líneas**
- A2 (Total de Commits Válidos): **18 commits**
- T (Tiempo Asumido en horas laborales): **8 horas**
=====
**MT4M4 (X) (X) = A2 / T = 2.250 comit modificado /hora**
** 44.61024305555556 lineas de codigo de comit modificado /segundos**
=====

```

Figura 212 Complejidad de modificación del grupo de control caso 1

- **Caso de estudio 2**

Dado que el valor de A es 17 y T = 8, es decir $X = 2.250$, así como lo indica la Figura 213.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M4.js

Procesando 19 commits con prefijo 'feat' o 'fix'...

=====
¡Análisis de Productividad y Densidad Completado!

### MÉTRICAS CALCULADAS
- A1 (Líneas de Código Modificadas): **1284748 líneas**
- A2 (Total de Commits Válidos): **18 commits**
- T (Tiempo Asumido en horas laborales): **8 horas**
=====
**MT4M4 (X) (X) = A2 / T = 2.250 comit modificado /hora**
** 44.60930555555556 lineas de codigo de comit modificado /segundos**
=====

```

Figura 213 Complejidad de modificación del grupo de control caso 2

- **Caso de estudio 3**

Dado que el valor de A es 17 y T = 8, es decir $X = 2.125$, así como lo indica la Figura 214.

```

ricse@RicseDev:~/tesis/carpet/MOVIES$ node MT4M4.js

Procesando 18 commits con prefijo 'feat' o 'fix'...

=====
¡Análisis de Productividad y Densidad Completado!

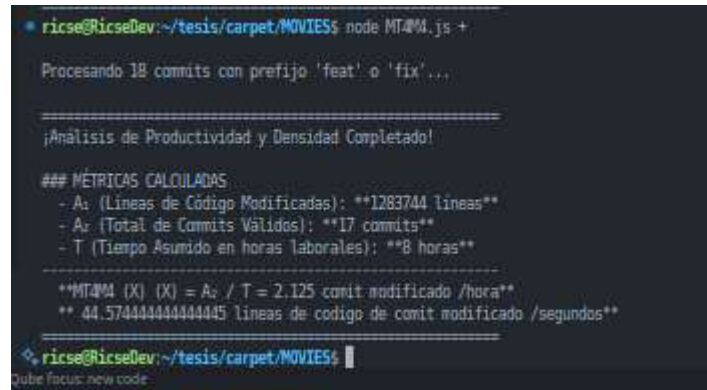
### MÉTRICAS CALCULADAS
- A1 (Líneas de Código Modificadas): **1283744 líneas**
- A2 (Total de Commits Válidos): **17 commits**
- T (Tiempo Asumido en horas laborales): **8 horas**
=====
**MT4M4 (X) (X) = A2 / T = 2.125 comit modificado /hora**
** 44.57444444444445 lineas de codigo de comit modificado /segundos**
=====
ricse@RicseDev:~/tesis/carpet/MOVIES$ git branch

```

Figura 214 Complejidad de modificación del grupo de control caso 3

- **Caso de estudio 4**

Dado que el valor de A es 17 y T = 8, es decir $X = 2.125$, así como lo indica la Figura 215.



```
ricse@RicseDev:~/tesis/carpet/MOVIES$ node MTQM.js +
Procesando 18 commits con prefijo 'feat' o 'fix'...

=====
¡Análisis de Productividad y Densidad Completado!
=====

## MÉTRICAS CALCULADAS
- A1 (Lineas de Código Modificadas): **1283744 líneas**
- A2 (Total de Commits Válidos): **17 commits**
- T (Tiempo Asumido en horas laborales): **8 horas**

=====

**MTQM (X) (X) = A2 / T = 2.125 commit modificado /hora**
** 44.574444444444445 líneas de código de comit modificado /segundos**
=====

ricse@RicseDev:~/tesis/carpet/MOVIES$
```

Figura 215 Complejidad de modificación del grupo de control caso 4

4.4.5.9. Complejidad funcional del test - MT5M1

- **Caso de estudio 1**

De acuerdo con el análisis en SonarQube, se registró un 21.6% de cumplimiento en la ejecución de pruebas unitarias sobre las funciones definidas en el proyecto, así como lo indica la Figura 216 y la Figura 217.

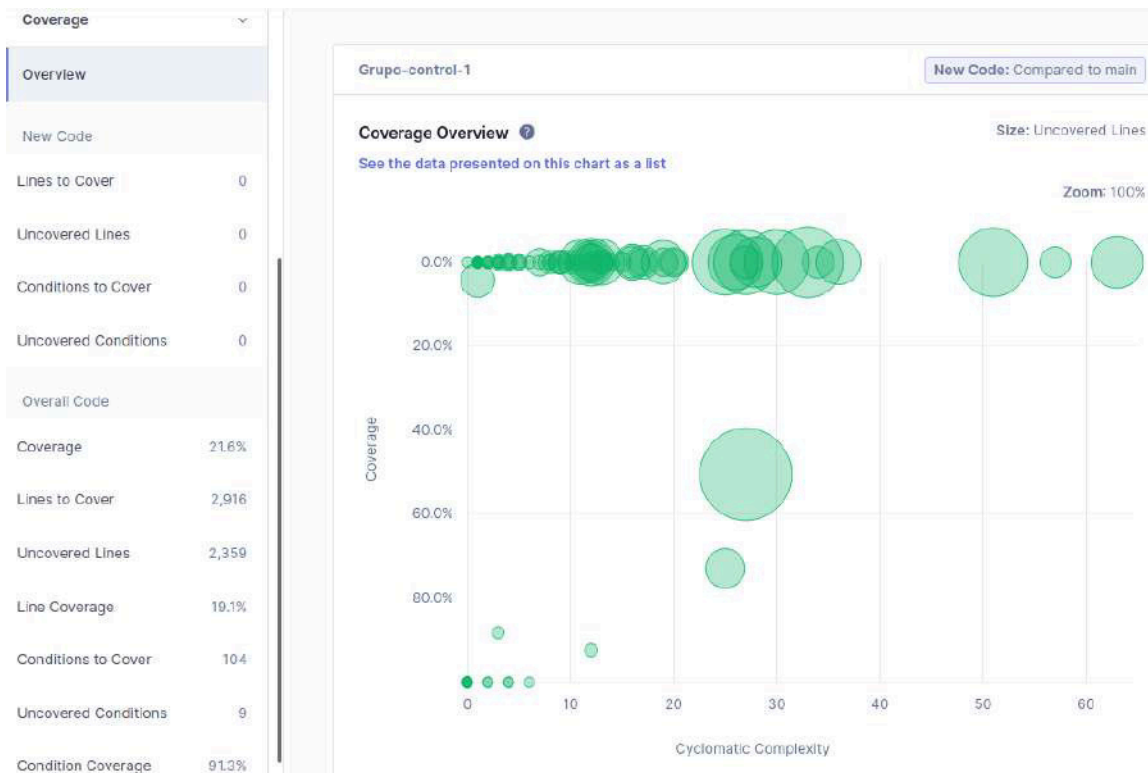


Figura 216 Diagrama Cobertura del grupo de control caso 1



Figura 217 Coverage del grupo de control caso 1

- **Caso de estudio 2**

De acuerdo con el análisis en SonarQube, se registró un 20.6% de cumplimiento en la ejecución de pruebas unitarias sobre las funciones definidas en el proyecto, así como lo indica la Figura 218 y la Figura 219

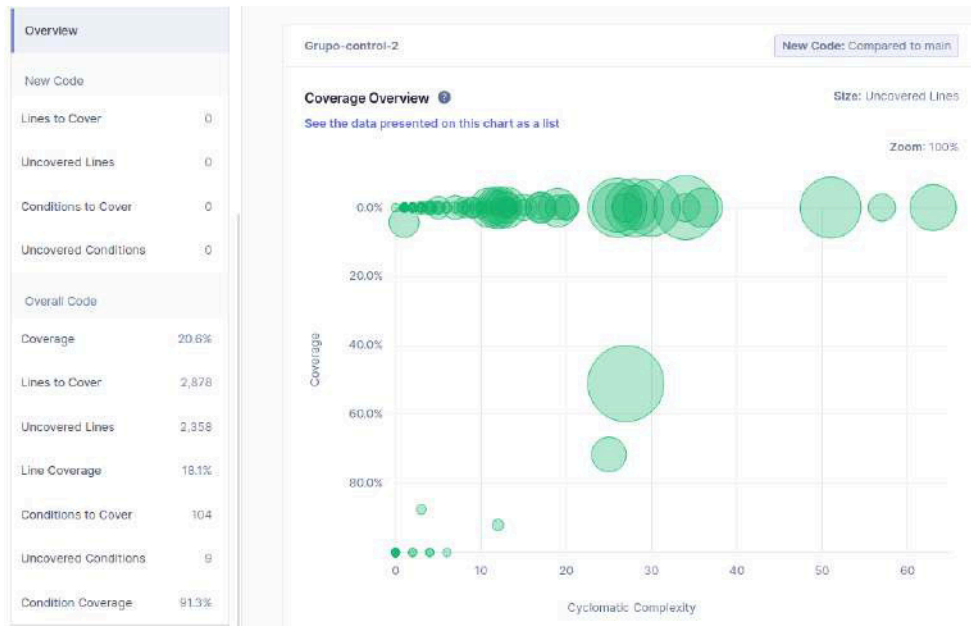


Figura 218 Diagrama Cobertura del grupo de control caso 2



Figura 219 Coverage del grupo de control caso 2

- **Caso de estudio 3**

De acuerdo con el análisis en SonarQube, se registró un 19.1% de cumplimiento en la ejecución de pruebas unitarias sobre las funciones definidas en el proyecto, así como lo indica la Figura 220 y 221.

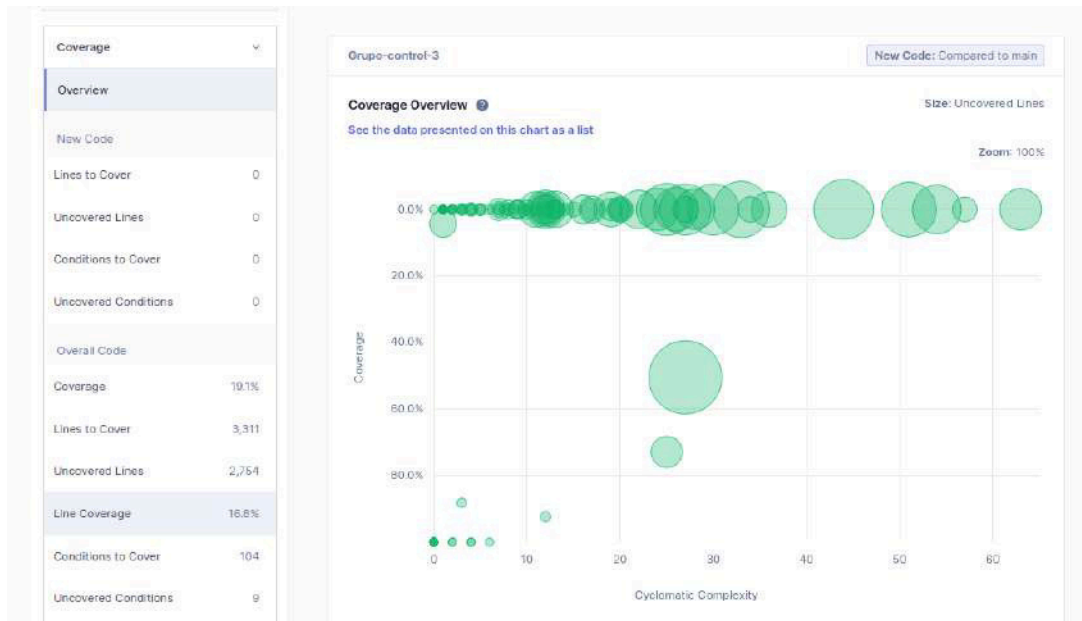


Figura 220 Diagrama Cobertura del grupo de control caso 3

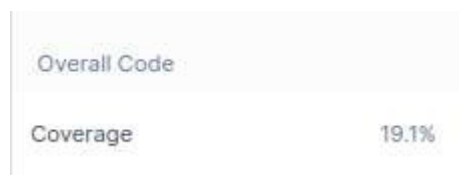


Figura 221 Coverage del grupo de control caso 3

- **Caso de estudio 4**

De acuerdo con el análisis en SonarQube, se registró un 21% de cumplimiento en la ejecución de pruebas unitarias sobre las funciones definidas en el proyecto, así como lo indica la Figura 222 y 223.

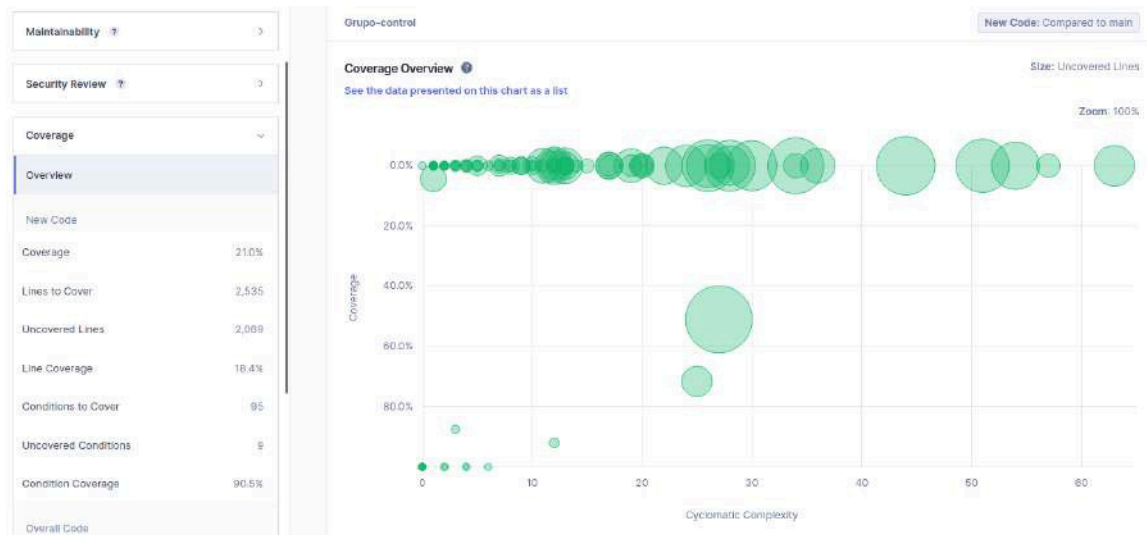


Figura 222 Diagrama Cobertura del grupo de control caso 4



Figura 223 Coverage del grupo de control caso 4

Capítulo V: Resultados y Discusión

Como se mencionó en el capítulo de metodología en la fase de validación del proyecto, el código estático fue analizado por las herramientas como SonarQube, dependency cruise o un Script de análisis automatizado, el código fue proveído por el Repositorio de Github Movies desarrollada en la implementación de la arquitectura con 2 grupos uno experimental de 3 capas y otro de control de 4 capas. El grupo experimental tuvo un promedio de 121 archivos promedio y 21403 líneas de código promedio y luego del filtro se obtuvo un total de 78 archivos promedio y 6957.25 líneas de código promedio en el caso del grupo de control se obtuvo 127.75 archivos promedio y 22697.25 líneas de código promedio y luego del filtro se obtuvo 81.75 archivos promedio y 7069.25 líneas de código promedio.

Luego se aplicó dependiendo el caso una abstracción del proyecto mediante nodos y así se obtuvo un archivo deps.json con los nodos del proyecto, un archivo de profundidad de herencia json, y un archivo xml de cobertura para el análisis en el sonarQube.

Posteriormente, se llevó a cabo el enfoque de pruebas por escenarios y la evaluación comparativa entre el grupo experimental y el grupo de control, consideró el uso de la arquitectura Clean Architecture basadas en tres capas y cuatro capas, respectivamente, el análisis tomó como referencia los resultados obtenidos previamente, destacando 5 criterios principales: modularidad, reusabilidad, analizabilidad, modificabilidad y testabilidad.

5.1. Resultados del Grupo experimental

En esta sección se recopiló los resultados obtenidos para el grupo experimental, organizados en función de las métricas definidas previamente. La Tabla 24 muestra los valores deseados, los rangos de referencias y los resultados alcanzados en cada caso.

Tabla 24 Resultados del grupo experimental

Métrica	Deseado	Rango	Casos de estudio			
			X1	X2	X3	X4
MT1M1	0	0 a 1	0.253	0.25	0.25	0.311
MT1M2	1	1 a 4	2.143	2.228	2.13	2.107
MT2M1	1	0 a 1	0.929	0.931	0.931	0.933
MT3M1	1	0 a 1	0.636	0.583	0.583	0.636
MT3M2	1	0 a 1	0.722	0.722	0.722	0.722

Métrica	Deseado	Rango	Casos de estudio			
			X1	X2	X3	X4
MT4M1	1	1 a 15	2.69	2.688	2.692	2.692
MT4M2	0	0 a 4	0	0	0	0
MT4M3	0	0 a 1	none	none	none	none
MT4M4	max	None	4	3.63	3.75	3.13
MT4M5	0	0 a 1	none	none	none	none
MT5M1	1	0 a 1	0.36	0.344	0.36	0.344

5.2. Resultados del Grupo de Control

En esta sección se recopiló los resultados obtenidos para el grupo de control, organizados en función de las métricas definidas previamente. La Tabla 25 muestra los valores deseados, los rangos de referencias y los resultados alcanzados en cada caso.

Tabla 25 Resultados del grupo de control

Métrica	Deseado	Rango	Casos de estudio			
			X1	X2	X3	X4
MT1M1	0	0 a 1	0.33	0.33	0.339	0.342
MT1M2	1	1 a 4	2.038	2.113	2.045	2.132
MT2M1	1	0 a 1	0.833	0.935	0.941	0.943
MT3M1	1	0 a 1	0.458	0.458	0.407	0.429
MT3M2	1	0 a 1	0.606	0.606	0.583	0.65
MT4M1	1	1 a 15	2.352	2.344	2.298	2.294
MT4M2	0	0 a 4	0	0	0	0
MT4M3	0	0 a 1	none	none	none	none
MT4M4	max	none	2.25	2.25	2.125	2.125
MT4M5	0	0 a 1	none	none	none	none
MT5M1	1	0 a 1	0.216	0.206	0.191	0.21

5.3. Normalización de los resultados

Una vez obtenido los resultados se realizó una normalización como se indica en la Tabla 27 y Tabla 28 de los datos con las siguientes ecuaciones para mantener rangos mínimos y máximos entre 0 a 1, esto para todos los resultados obtenidos posterior al análisis de los resultados usando las ecuaciones la Tabla 26.

Tabla 26 Normalización de las métricas

Métrica	Deseado	Rango	Ecuaciones
MT1M1	0	0 a 1	$Si\ 0 < x < 1; 1 - x; Si\ 0\ es\ lo\ deseado$
MT1M2	1	1 a 4	$Si\ 1 < x < 4; = (1 - (x - 1)/(4 - 1));$ $Si\ 1\ es\ lo\ deseado$
MT2M1	1	0 a 1	$Si\ 0 < x < 1; x; Si\ 1\ es\ lo\ deseado$
MT3M1	1	0 a 1	$Si\ 0 < x < 1; x; Si\ 1\ es\ lo\ deseado$
MT3M2	1	0 a 1	$Si\ 0 < x < 1; x; Si\ 1\ es\ lo\ deseado$
MT4M1	1	1 a 15	$Si\ 1 < x < 15; (1 - (x - 1)/(15 - 1));$ $Si\ 1\ es\ lo\ deseado$
MT4M2	0	0 a 4	$Si\ 0 < x < 4; 1(1 - (x/4)); Si\ 0\ es\ lo\ deseado$
MT4M3	0	0 a 1	$Si\ 0 < x < 1; 1 - x; Si\ 0\ es\ lo\ deseado$
MT4M4	max	None	$Si\ 1 < x < Caso\ max; ((X - 1)/(caso\ max - 1));$ $Si\ caso\ max\ es\ lo\ deseado$

Métrica	Deseado	Rango	Ecuaciones
MT4M5	0	0 a 1	$Si\ 0 < x < 1; 1 - x; Si\ 0\ es\ lo\ deseado$
MT5M1	1	0 a 1	$Si\ 0 < x < 1; x; Si\ 1\ es\ lo\ deseado$

Tabla 27 Resultado normalizado del grupo experimental

Métrica	Deseado	Rango	Normalización de los Casos de estudio				
			N1	N2	N3	N4	N promedio
MT1M1	1	0 a 1	0.747	0.75	0.75	0.689	0.734
MT1M2	1	0 a 1	0.619	0.59	$\frac{0.623}{3}$	0.631	0.616
MT2M1	1	0 a 1	0.929	0.931	0.931	0.933	0.931
MT3M1	1	0 a 1	0.636	0.583	0.583	0.636	0.61
MT3M2	1	0 a 1	0.722	0.722	0.722	0.722	0.722
MT4M1	1	0 a 1	0.8792	0.8794	$\frac{0.879}{1}$	0.8791	0.8793
MT4M2	1	0 a 1	1	1	1	1	1
MT4M3	1	0 a 1	none	none	none	none	none
MT4M4	1	0 a 1	1	0.876	0.916	0.71	0.875
MT4M5	1	0 a 1	none	none	none	none	none
MT5M1	1	0 a 1	0.36	0.344	0.36	0.344	0.352

Tabla 28 Resultado en normalizado del grupo de control

Métrica	Deseado	Rango	Normalización de los Casos de estudio				
			N1	N2	N3	N4	N promedio
MT1M1	0 a 1	0 a 1	0.667	0.67	0.661	0.658	0.664
MT1M2	0 a 1	0 a 1	0.654	0.629	0.651	0.622	0.639
MT2M1	0 a 1	0 a 1	0.833	0.935	0.941	0.943	0.913

Métrica	Deseado	Rango	Normalización de los Casos de estudio				
			N1	N2	N3	N4	N promedio
MT3M1	0 a 1	0 a 1	0.458	0.458	0.407	0.429	0.438
MT3M2	0 a 1	0 a 1	0.606	0.606	0.583	0.65	0.611
MT4M1	0 a 1	0 a 1	0.903	0.904	0.907	0.907	0.905
MT4M2	0 a 1	0 a 1	1	1	1	1	1
MT4M3	0 a 1	0 a 1	none	none	none	none	none
MT4M4	0 a 1	0 a 1	1	0.876	0.916	0.71	0.875
MT4M5	0 a 1	0 a 1	none	none	none	none	none
MT5M1	0 a 1	0 a 1	0.216	0.206	0.191	0.21	0.205

5.4. Resultados por sub-características

5.4.1 Modularidad

Los resultados normalizados se recopiló de las Tablas 27 y Tabla 28, así como se mostró la comparación en la Figura 224, la cual fue utilizada para medir tanto al grupo experimental como al grupo de control según, para hallar la modularidad se debe realizar la suma de las métricas MT1M1 y MTM2.

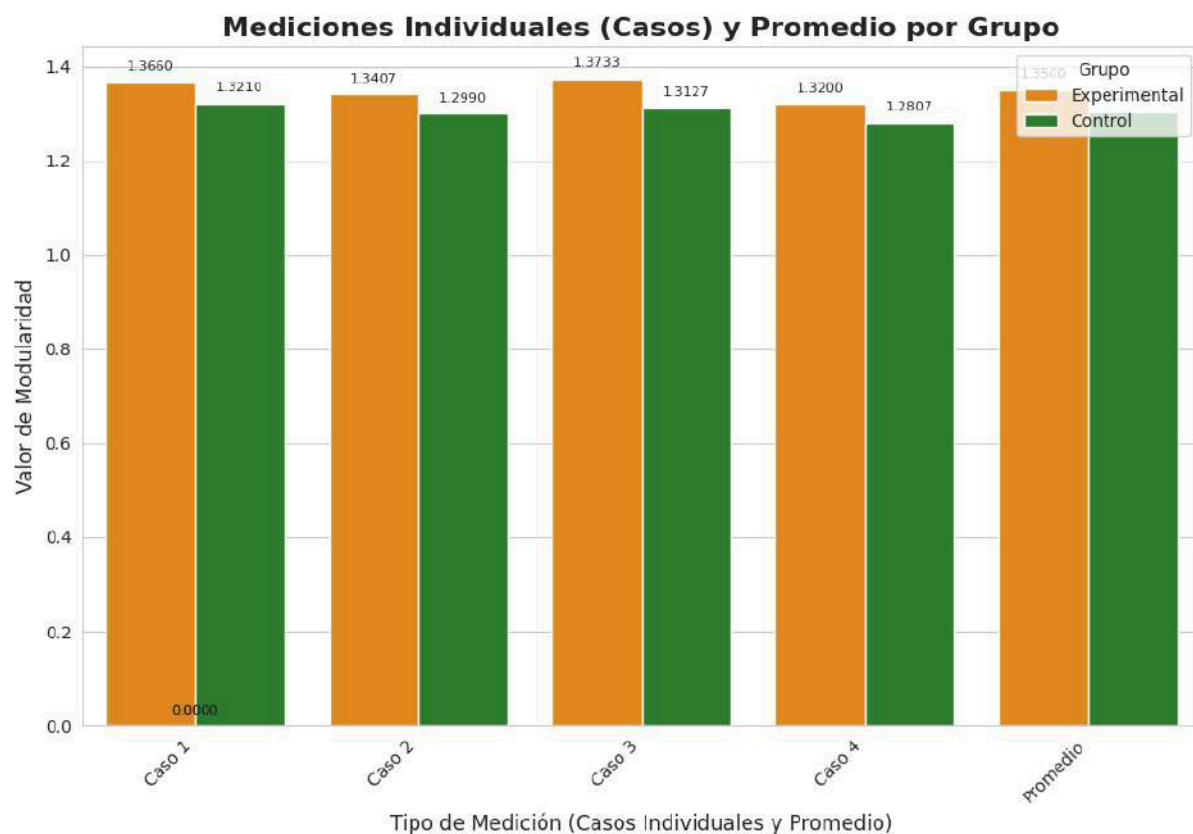


Figura 224 Diagrama comparativo de la modularidad

En conjunto, se alcanzó una mejora del 3.85% en promedio de la “modularidad” del proyecto, mejorando la capacidad del sistema por la correcta división e independencias de su modularidad. La Tabla 29 mostró la comparación de la modularidad a nivel de sistema.

Tabla 29 Resultados del criterio de modularidad.

Modularidad	Experimental	Control
Caso 1	1.366	1.321
Caso 2	1.340	1.299
Caso 3	1.373	1.312
Caso 4	1.32	1.28
Caso promedio	1.35	1.30

5.4.2. Reusabilidad

Los resultados normalizados se recopilaron de las Tablas 27 y Tabla 28, la cual fue utilizada para medir tanto al grupo experimental como al grupo de control. En la figura, para la métrica MT2M1 se representó como la reusabilidad del sistema.

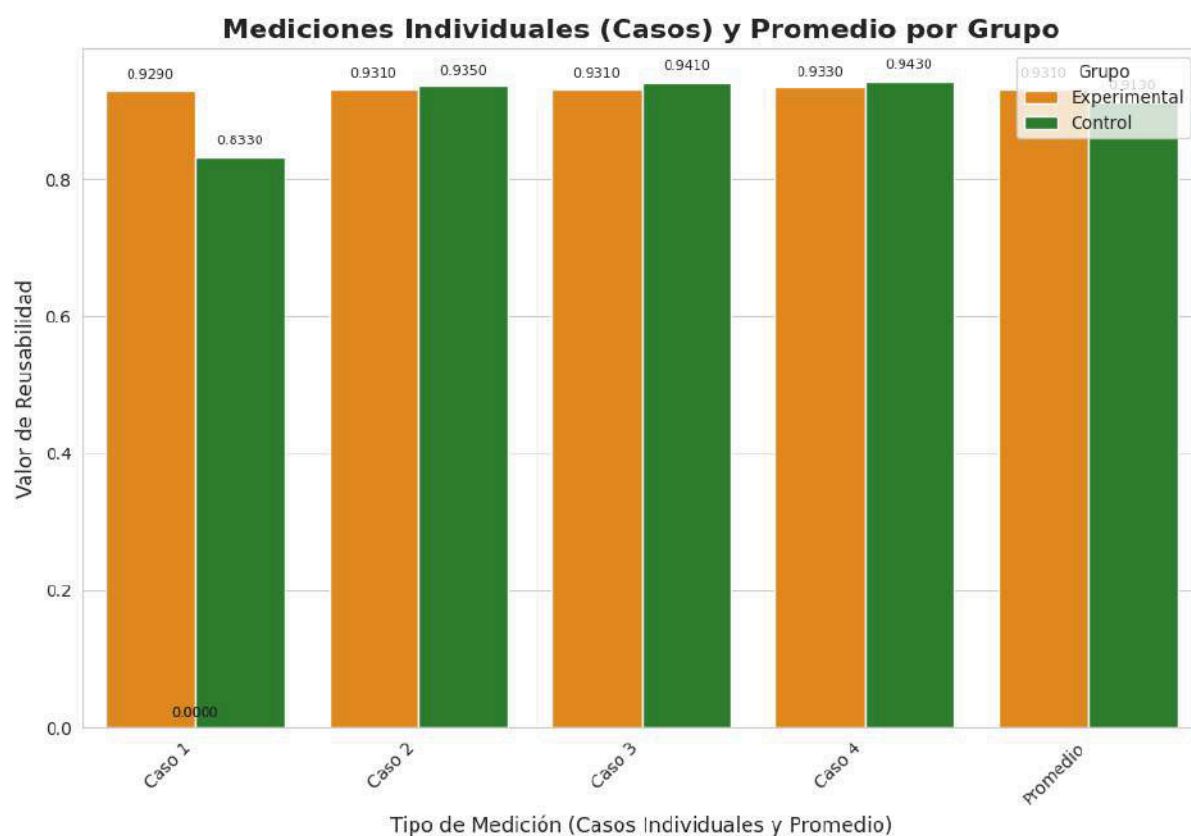


Figura 225 Diagrama comparativo para las métricas de reusabilidad

En síntesis, se ha mejorado un 1.97% en la “reusabilidad” promedio del sistema, mejorando la capacidad de ser reusado los componentes dentro del sistema según la Tabla 30.

Tabla 30 Resultados del criterio de reusabilidad

Reusabilidad	Experimental	Control
Caso 1	0,929	0,833
Caso 2	0,931	0,935
Caso 3	0,931	0,941
Caso 4	0,933	0,943
Caso promedio	0,931	0,913

5.4.3. Analizabilidad

Los resultados normalizados se recopilaron de las Tablas 27 y Tabla 28, la cual fue utilizada para medir tanto al grupo experimental como al grupo de control. En la Figura 226, se mostró la suma de las métricas MT3M1 y MT3M2, que representa la analizabilidad del proyecto.

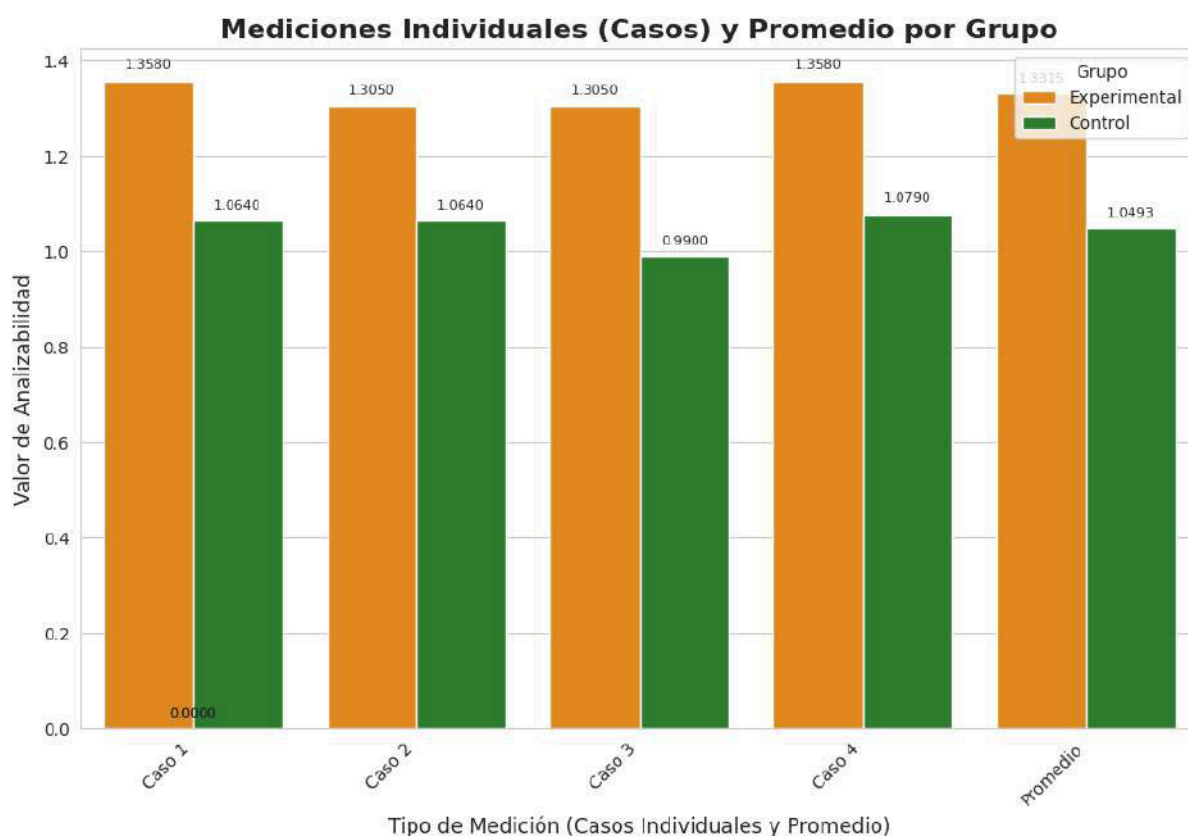


Figura 226 Diagrama comparativo para las métricas de Analizabilidad

En resumen, se ha mejorado la “analizabilidad” del proyecto, obteniendo un 28.08% promedio en el criterio de analizabilidad. La Tabla 31 mostró la comparación de la analizabilidad a nivel del grupo de control y experimental.

Tabla 31 Resultados del criterio de analizabilidad

Analizabilidad	Experimental	Control
Caso 1	1.358	1.064
Caso 2	1.305	1.064
Caso 3	1.305	0.99

Caso 4	1.358	1.079
Caso promedio	1.332	1.04

5.4.4. Modificabilidad

Los resultados normalizados se recopilaron de las Tablas 27 y Tabla 28, la cual fue utilizada para medir tanto al grupo experimental como al grupo de control. En la Figura 227, se mostró la suma de las métricas MT4M1, MT4M2 y MT4M4 que representa la modificabilidad del proyecto.

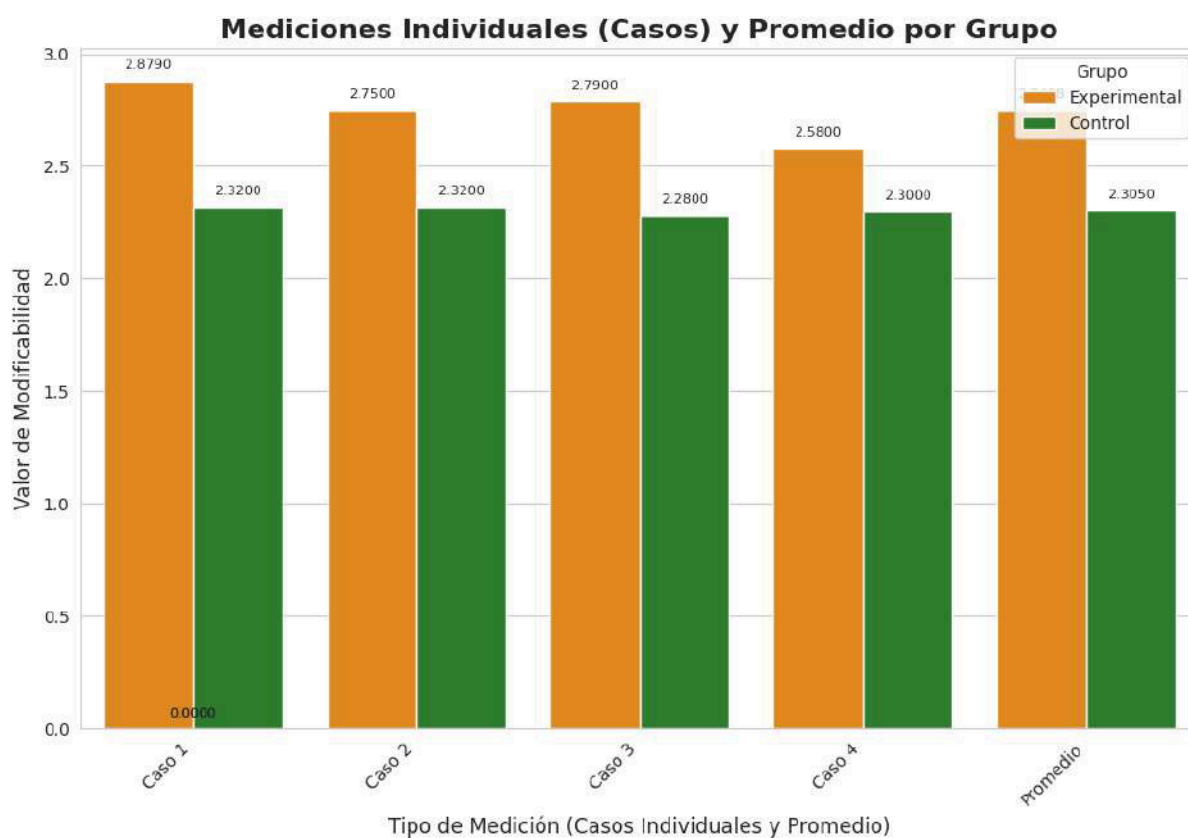


Figura 227 Diagrama comparativo para las métricas de Modificabilidad

De acuerdo a los resultados se ha mejorado un 19.78% promedio en la “modificabilidad” del proyecto, obteniendo la misma capacidad de ser modificado dentro del sistema. La Tabla 32 mostró la modificabilidad a nivel de sistema.

Tabla 32 Resultados del criterio de Modificabilidad

Modificabilidad	Experimental	Control
Caso 1	2.879	2.32
Caso 2	2.75	2.32
Caso 3	2.79	2.28
Caso 4	2.58	2.28
Caso promedio	2.755	2.3

5.4.5. Testabilidad

Los resultados normalizados se recopilaron de las Tablas 27 y Tabla 28, la cual fue utilizada para medir tanto al grupo experimental como al grupo de control según, se consideró las métricas de reusabilidad. En la Figura 228, mostró la métrica MT5M1 que representa la testabilidad del sistema.

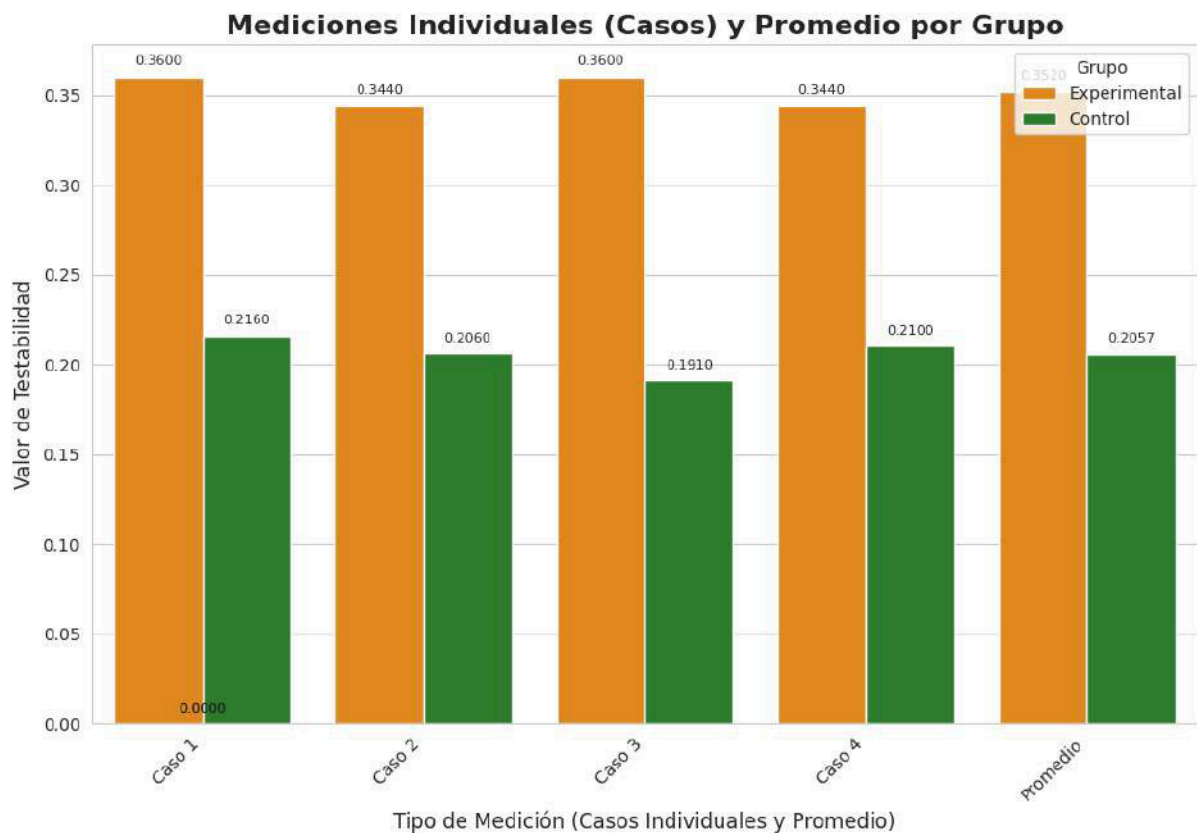


Figura 228 Diagrama comparativo para las métricas de Testabilidad

Los resultados nos indican una mejora del 71.71% promedio en la “testabilidad” del proyecto, mejorando la capacidad de ser testeado el sistema. La Tabla 33 mostró la testabilidad a nivel de sistema.

Tabla 33 Resultados del criterio de Testabilidad

Testabilidad	Experimental	Control
Caso 1	0.36	0.216
Caso 2	0.344	0.206
Caso 3	0.36	0.191
Caso 4	0.344	0.21
Caso promedio	0.352	0.205

5.5. Resultados del Sistema

En términos generales, los resultados mostraron una mejora global del 16.71% promedio en la calidad del proyecto, destacando incrementos en modularidad (3.85%), reusabilidad (1.97%), analizabilidad (28.08%), modificabilidad (19.78%) y testabilidad (71.71%). La Figura 229 representó las métricas usadas en el sistema.

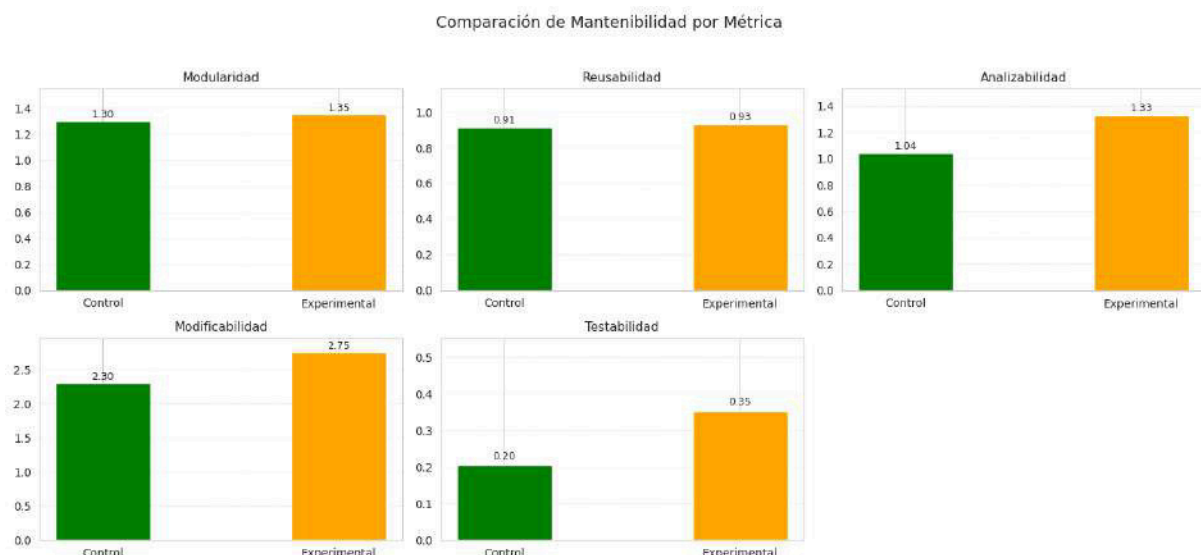


Figura 229 Diagrama comparativo de las métricas de mantenibilidad del sistema

En la Figura 230, se presentó el diagrama de líneas que ilustró la variación de las métricas de calidad entre el grupo experimental y el de control, el cual, se observó un incremento en la

analizabilidad, modificabilidad, testabilidad, mientras que la modularidad y reusabilidad permanece un incremento en menor medida. La tendencia general reflejó una mejora del 16.71% en la calidad del proyecto.

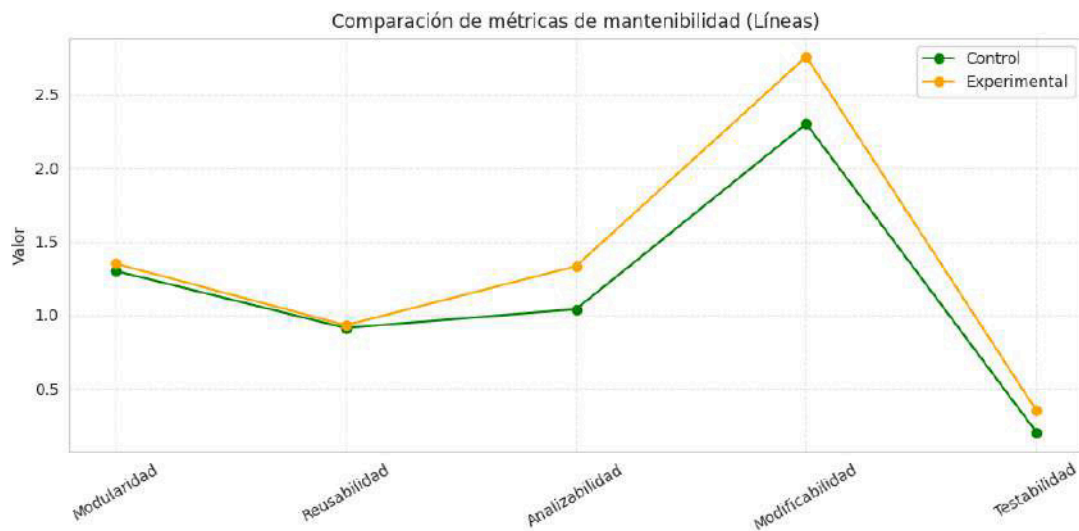


Figura 230 Diagrama de línea de las métricas de mantenibilidad del sistema

La interpretación global de la Figura 240 mostró un polígono del grupo experimental más amplio en la mayoría de ejes, lo que confirmó una mejora general de la mantenibilidad del 16.71%.

Comparación de Métricas de Calidad (Control vs. Experimental)

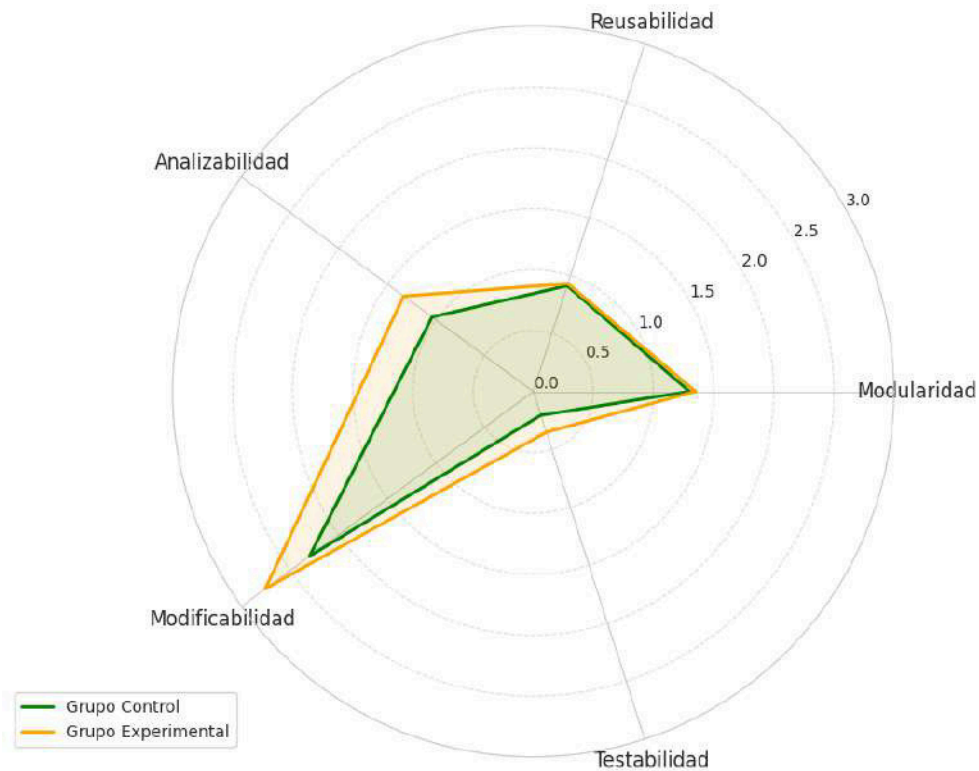


Figura 240 Diagrama de radar de las métricas de mantenibilidad del sistema

La Tabla 34 presentó los valores promedios obtenidos para cada criterio de mantenibilidad en ambos grupos, evidenciando las diferencias específicas que sustentan la mejora global alcanzada.

Tabla 34 Descripción de los resultados de los criterios de mantenibilidad

Criterios de mantenibilidad	Fórmula	Valor de Grupo	
		Experimental promedio	Control promedio
Modularidad	MT1M1 + MT1M2	1.32	1.28
Reusabilidad	MT2M1	0.99	0.88
Analizabilidad	MT3M1+MT3M2	1.358	1.286
Modificabilidad	MT4M1 + MT4M2 + MT4M4	2.87	2.57
Testabilidad	MT5M1	0.344	0.21
Total		6.882	6.226

La Tabla 35 resumen los resultados comparativos de los criterios de mantenibilidad entre los grupos de control y experimental donde mostró en conjunto una tendencia positiva hacia la mejora de la calidad del proyecto. La Figura 241 mostró una comparativa de barras grupo experimental versus grupo de control lo que confirmó una mejora en cada sub característica .

Tabla 35 Resultados en rendimiento promedio

Criterio	Control	Experimental	Diferencia (%)
Modularidad	1.30	1.35	+3.85
Reusabilidad	0.913	0.931	+1.97
Analizabilidad	1.04	1.332	+28.08
Modificabilidad	2.3	2.755	+19.78
Testabilidad	0.205	0.352	+71.71
Proyecto	5.758	6.72	+16.71

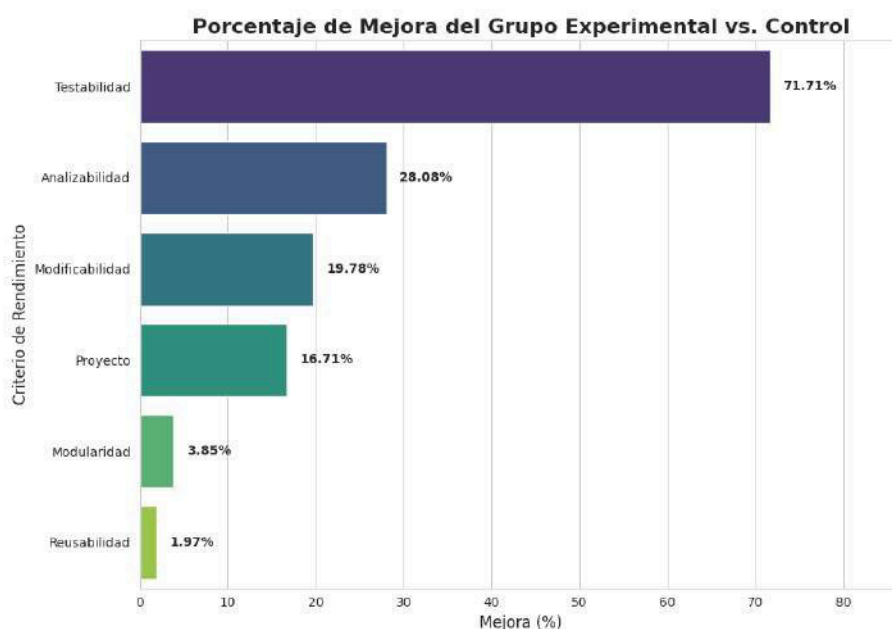


Figura 241 Diagrama del porcentaje de mejora según criterio de mantenibilidad y sistema

5.6. Discusión

Con los resultados obtenidos, en la Tabla 34 y 35, se realizó un análisis frente a otros trabajos que también implementaron arquitectura en sus proyectos, nuestro hallazgo logró obtener un aumento de +16.71% en la mantenibilidad general con una Clean Architecture simplificada de 3 capas, esta contribución superó las limitaciones de estudios previos, lo cual garantizó una mantenibilidad del código mayor frente a las arquitecturas evaluadas por los otros autores en la literatura. Específicamente el estudio de (Arias-Orezano et al., 2021), solo demostró una mejora del +7% en analizabilidad y otra de 0.7% en testabilidad al usar una arquitectura tradicional de 4 capas, frente a nuestro trabajo con una mejora de +71.71% de testabilidad y +28.08% de analizabilidad ambos bajo la norma ISO/IEC 25010. Además, la mejora combinada de la modificabilidad +19.78% y reusabilidad del +1.97% cuantificó la flexibilidad que (Aguilar y Figueira 2020) en su implementación de Clean Architecture tradicional solo logró un análisis de migración cualitativa de la interfaz, donde solo pudo observar el sistema y no una medición cuantificable. Adicionalmente, esta alta estabilidad mitiga directamente la menor cobertura de prueba que (Milić et al. 2022) identificó como riesgo en sus arquitecturas modulares. El enfoque riguroso en sus resultados internos a nivel de código se diferencia de los estudios que priorizaron productividad como el trabajo de (Naimi et al. 2025) donde generó el proyecto entre 7 horas en promedio pero generó un riesgo potencial a nivel de mantenibilidad, los buenos resultados de los criterios de modificabilidad y reusabilidad tienen relación directa con los objetivos de (Rojas et al. 2021; Sadek et al. 2022), donde la alta modificabilidad garantiza que cualquier servicio individual en sus componentes SOAP pueda ser actualizado, manteniendo la evolución con menor esfuerzo y riesgo, lo mencionado disminuyó el costo de desarrollo que buscaba (Rojas et al. 2021) para las PYMES. Por lo tanto el modelo de 3 capas es fundamental en la calidad de código y garantiza la sostenibilidad de diferentes situaciones y problemáticas, incluso las más complejas.

Capítulo VI: Conclusiones

6.1. Conclusiones

6.1.1. Conclusión general

En esta presente investigación, se desarrolló un sistema de películas web, donde se utilizó una arquitectura basada en Clean Architecture tanto de 3 capas para el grupo experimental y 4 capas para el grupo de control, concluyendo que el sistema tiene una repercusión positiva en la mantenibilidad según los criterios de la ISO/IEC 25010 y la ISO/IEC 25022. La metodología de investigación estuvo compuesta por 4 fases: (i) Evaluación arquitectónica (ii) Diseño de la arquitectura de 3 capas (iii) Implementación de sistema y (iv) Validación de la propuesta.

6.1.2. Conclusiones específicas

- **OE1 : Analizar y revisar diferentes tipos de arquitectura para implementar un sistema.**

Se analizó selección de 32 trabajos de investigación, se identificó que 8 de nuestros artículos abordaron arquitectura de software, 8 frameworks para la construcción de software, 8 principios y patrones de diseño y 8 relacionados con las métricas de evaluación del sistema de esto se justificó la adopción de la Clean Architecture como el diseño debido a su validación de resultado empírico.

- **OE2 : Diseñar una Arquitectura para un sistema que disminuya la complejidad inicial en proyectos basados en Clean Architecture**

Para este fin se decidió usar un diseño de Clean Architecture basado en 3 capas, regida por los principios SOLID y patrones de diseño Bridge, Strategy y Builder; estos 2 últimos en la variabilidad tecnológica propuesta en los puertos el cual obtuvo buenos resultados en la implementación de un proyecto bajo el diseño de 3 capas propuesto, debido a la generación de 4 casos de estudio se propuso generar diferente funcionalidades, por esta razón se optó por una metodología Ágil lo cual aporta flexibilidad en el diseño.

- **OE3 : Desarrollar un sistema que sirva como apoyo para corroborar la reducción de la complejidad inicial basado en Clean Architecture .**

Se diseñó e implementó un sistema web para el apoyo de la validación de la arquitectura, que cuenta con 6 módulos: (i) módulo de gestión de usuario, (ii) módulo de películas, (iii) módulo de favoritos, (iv) módulo de calificaciones, (v) módulo de lista de seguimiento y (vi) módulo de accesibilidad usando la metodología ágil SCRUM, debido a la división de los requisitos y tiempos de desarrollo que constó en 2 sprint y 6 historias de usuario .

- **OE4 : Validar la mantenibilidad del diseño arquitectónico considerando el caso de estudio**

Para validar el modelo se categorizó una evaluación por 2 grupos comparativos: el grupo experimental y el grupo de control cada uno con 4 escenarios de estudio. En el primero se aplicó la arquitectura propuesta de tres capas, y en el segundo, se empleó una arquitectura de 4 capas, ambas basadas en la Clean Architecture, En el repositorio se tomó los datos estático tanto por grupo experimental que tuvo un promedio de 121 archivos promedio y 21403 líneas de código promedio, luego del filtro se obtuvo un total de 78 archivos promedio y 6957.25 líneas de código promedio en el caso del grupo de control se obtuvo 127.75 archivos promedio y 22697.25 líneas de código promedio y luego del filtro se obtuvo 81.75 archivos promedio y 7069.25 líneas de código promedio, una vez obtenidos los resultado posterior a su transformación en archivos se aplicó una normalización para obtener valores numéricos entre 0 y 1. Los resultados obtenidos demostraron que la arquitectura basada en 3 capas alcanzó resultados para los criterios de mantenibilidad: modularidad, reusabilidad, analizabilidad, modificabilidad y testabilidad con una mejora de 3.85%, 1.97%, 28.08%, 19.78% y 71.71% respectivamente y una mejora general del 16.71%, asegurando una mejora en la mantenibilidad del Software.

6.2. Trabajos futuros

Como trabajo futuro, se recomienda implementar el sistema propuesto utilizando otro framework, como Angular, dado su respaldo empresarial, asimismo, se sugiere la posibilidad de llevar el sistema a un entorno laboral real, con mayor volumen de datos y un periodo de mantenibilidad superior a un año, a fin de observar el comportamiento de las métricas que no pudieron ser evaluadas en este estudio.

Referencias bibliográficas

- Aguilar, P., & Figueira, L. B. (2020). Clean Architecture is not only about business logic. I Workshop de Tecnologia da Fatec Ribeirão Preto, Vol. 1, n.º 1, jul/2020. Faculdade de Tecnologia do Estado de São Paulo – FATEC Ribeirão Preto. Disponible en http://fatecrp.edu.br/WorkTec/edicoes/2020-1/trabalhos/I-Worktec-Pablo_Aguilar.pdf
- Aktar, M. S., Liang, P., Waseem, M., Tahir, A., Ahmad, A., Zhang, B., & Li, Z. (2025). Architecture decisions in quantum software systems: An empirical study on Stack Exchange and GitHub. Information and Software Technology, 177. <https://doi.org/10.1016/j.infsof.2024.107587>
- Arango, E.C., & Loaiza, O.L. (2021). SCRUM Framework Extended with Clean Architecture Practices for Software Maintainability. Software Engineering and Algorithms.
- Avilés Matute, S., Avila-Pesantez, D., & Avila, M. (2020). Desarrollo de sistema Web basado en los frameworks de Laravel y VueJs, para la gestión por procesos: Un estudio de caso. Revista Peruana de Computación y Sistemas, 3(2), 3–10. <https://doi.org/10.15381/rpcs.v3i2.19256>
- Baul Canlas, R., Cruz Piad, K., & Carpio Lagman, A. (2021). An ISO/IEC 25010 Based Software Quality Assessment of a Faculty Research Productivity Monitoring and Prediction System. ACM International Conference Proceeding Series, 238–242. <https://doi.org/10.1145/3512576.3512619>
- Cai, C. H., Sun, J., & Dobbie, G. (2025). The automation of design model repair. Science of Computer Programming, 245. <https://doi.org/10.1016/j.scico.2025.103313>
- Cecil, R. (2017). Clean Architecture A Craftman's Guide to Software Structure and Desing. <https://drive.google.com/drive/u/1/folders/10nHHKadBteKYjs94vHjue3wQ4yELY6tf>

- Crespo-Martínez, E.; Astudillo-Rodríguez, C.; C., Chica-Contreras, G., & Vásquez-Aguilera, A. (2023). Technology Acceptance Model of ERP software in Small Business: A Systematic Literature review. <https://doi.org/10.29019/enfoqueute.884>
- Eyada, M. M., Saber, W., el Genidy, M. M., & Amer, F. (2020). Performance Evaluation of IoT Data Management Using MongoDB Versus MySQL Databases in Different Cloud Environments. *IEEE Access*, 8, 110656–110668. <https://doi.org/10.1109/ACCESS.2020.3002164>
- Francisco Arias-Orezano, J., David Reyna-Barreto, B., & Mamani-Apaza, G. (2021). Repercusión de arquitectura limpia y la norma ISO/IEC 25010 en la mantenibilidad de aplicativos Android. <https://doi.org/10.22430/22565337.2104>
- Garcia, J., Mirakhorli, M., Xiao, L., Zhao, Y., Mujhid, I., Pham, K., Okutan, A., Malek, S., Kazman, R., Cai, Y., & Medvidovic, N. (2021). Constructing a Shared Infrastructure for Software Architecture Analysis and Maintenance. *Proceedings - IEEE 18th International Conference on Software Architecture, ICSA 2021*, 150–161. <https://doi.org/10.1109/ICSA51549.2021.00022>
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2003). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. ISBN: 0-201-63361-2.
- Gupta, S., Lago, P., & Donker, R., (2021). A Framework of Software Architecture Principles for Sustainability-driven Design and Measurement. *Proceedings - 2021 IEEE 18th International Conference on Software Architecture Companion, ICSA-C 2021*, 31–37. <https://doi.org/10.1109/ICSA-C52384.2021.00012>
- Houichime, T., & el Amrani, Y. (2025). Context Is All You Need: A Hybrid Attention-Based Method for Detecting Code Design Patterns. *IEEE Access*, 13, 9689–9707. <https://doi.org/10.1109/ACCESS.2025.3525849>

International Organization for Standardization. (2011). *ISO/IEC 25010:2011: Systems and software engineering — Systems and software product quality requirements and evaluation (SQuaRE) — System and software quality models*. ISO.

International Organization for Standardization. (2016). *ISO/IEC 25022:2016: Systems and software engineering — Systems and software quality requirements and evaluation (SQuaRE) — Measurement of quality in use*. ISO.

Jhunjhunwala, P., Blech, J. O., Zoitl, A., Atmojo, U. D., & Vyatkin, V. (2021). A Design Pattern for Monitoring Adapter Connections in IEC 61499. Proceedings of the IEEE International Conference on Industrial Technology, 2021-March, 967–972. <https://doi.org/10.1109/ICIT46573.2021.9453685>

Jolak, R., Karlsson, S., & Dobslaw, F. (2025). An empirical investigation of the impact of architectural smells on software maintainability. Journal of Systems and Software, 225. <https://doi.org/10.1016/j.jss.2025.112382>

Kerdoudi, M. L., Ziadi, T., Tibermacine, C., & Sadou, S. (2022). A novel approach for Software Architecture Product Line Engineering. Journal of Systems and Software, 186. <https://doi.org/10.1016/j.jss.2021.111191>

Lei, Z., Zhou, H., Hu, W., & Liu, G.-P. (2022). Toward an international platform: A web-based multi-language system for remote and virtual laboratories using React framework. Heliyon, 8(10), e10780. <https://doi.org/10.1016/j.heliyon.2022.e10780>

Liu, A., Lefever, J., Han, Y., Cai, Y., & Academy, G. (2024). Prevalence and Severity of Design Anti-Patterns in Open Source Programs-A Large-Scale Study. <https://doi.org/https://doi.org/10.1016/j.infsof.2024.107429>

M. Falco, E. Scott y G. Robiolo, “Overview of an Automated Framework to Measure and Track the Quality Level of a Product,” 2020 IEEE Congreso Bienal de Argentina sobre Ciencia y Tecnología, IEEE, 2020.

- Milić, M., & Makajić-Nikolić, D. (2022). Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures. *Symmetry*, 14(9). <https://doi.org/10.3390/sym14091824>
- Martínez Z., J. C., Henao, C., Henao, F., & Zapata, E. (2021). Utilización de arquitecturas limpias para trabajo con buenas prácticas en la construcción de aplicaciones Java. *Revista Innovación Digital y Desarrollo Sostenible – IDS*, 1(2), 133–140. <https://doi.org/10.47185/27113760.v1n2.37>
- Márquez, G., Astudillo, H., & Kazman, R. (2023). Architectural tactics in software architecture: A systematic mapping study. *Journal of Systems and Software*, 197. <https://doi.org/10.1016/j.jss.2022.111558>
- Mena, D., & Santorum, M. (2021). Maintainability and Portability Evaluation of the React Native Framework Applying the ISO/IEC 25010. *Advances in Intelligent Systems and Computing*, 1273 AISC, 429–439. https://doi.org/10.1007/978-3-030-59194-6_35
- Naimi, L., Benaddi, L., Ouaddi, C., Souha, A., Abdelmalek, H., & Jakimi, A. (2025). Model-Driven Architecture for Accelerating the Development of Tourism Applications. *IEEE Access*, 13, 131696–131715. <https://doi.org/10.1109/ACCESS.2025.3592552>
- Ollila, R., Mäkitalo, N., & Mikkonen, T. (2022). Modern Web Frameworks: A Comparison of Rendering Performance. *Journal of Web Engineering*, 21(3), 789–814. <https://doi.org/10.13052/jwe15>
- Paliuanan, L. P., Talamayan, C., Karl Agpalza, J. A., Agoto, J. S., Javier, B. S., & Talamayan, C. T. (2022). Development and Evaluation of the Institutionally-Farmed Research On-line Repository and Management System (InFORMs) towards Knowledge-Sharing and Utilization. *Article in International Journal of Computer Network and Information Security*, 22(5), 722. <https://doi.org/10.22937/IJCSNS.2022.22.5.99>

- Patiño Vásquez, D. A., & Llerena Izquierdo, J. F. (2024). *A Literature Review Focused on Current Trends and Practices in Web Application Development*. Papel presentado en el X Congreso Internacional de Ciencia, Tecnología e Innovación para la Sociedad (CITIS 2024). <https://doi.org/10.17163/abyaups.92.695>
- Piyawardhana, M. D. V. P., Bawantha, H. D. M., Madhuwantha, K. T. B., & de Silva, A. L. C. (2023). An empirical study of the impact of software design patterns on code quality. <https://doi.org/10.36227/techrxiv.23540073.v1>
- Quiña-Mera, A., Flores Landeta, D., Rea-Peñañiel, X. M., & Guevara-Vega, C. (2022). Quality Evaluation of a Spring Cloud Microservices Architecture Implementation. *Smart Innovation, Systems and Technologies*, 252, 745–754. https://doi.org/10.1007/978-981-16-4126-8_66
- Rahmi Dewi, M., Ngaliah, N., & Rochimah, S. (2020). Maintainability Measurement and evaluation of myITS Mobile Application Using ISO 25010 Quality Standard. *Proceedings - 2020 International Seminar on Application for Technology of Information and Communication: IT Challenges for Sustainability, Scalability, and Security in the Age of Digital Disruption, ISemantic 2020*, 530–536. <https://doi.org/10.1109/iSemantic50169.2020.9234283>
- Rathinam, S. (2022). Analysis and Comparison of Different Frontend Frameworks. 1–10. https://www.suryaanshrathinam.com/static/media/SuryaanshRathinam_ComparisonOfFrontendFrameworks.a6d0c12b1b711c4cc8aa.pdf
- Robayo Linares, J. E., Ortiz Suarez, S., Calderón Moreno, R. (2024). Análisis del rendimiento de aplicaciones web desarrolladas sobre Blazor de .NET Core, una revisión de su estado del arte. *Revista Ingeniería, Matemáticas y Ciencias de la Información*, 11(22), 1. Dialnet
- Rojas, H., Arias, K. A., & Renteria, R. (2021). Service-oriented architecture design for small and medium enterprises with infrastructure and cost optimization. *Procedia Computer Science*, 179, 488–497. <https://doi.org/10.1016/j.procs.2021.01.032>

- Rashid, R. S., & Ismael, A. A. (2025). *The Role of Web Programming in Modern IT Solutions: Trends and Challenges*. *Journal of Information Systems Engineering and Management*, 10(27s). <https://doi.org/10.52783/jisem.v10i27s.4380>
- Saundariya, K., Abirami, M., Senthil, K. R., Prabakaran, D., Srimathi, B., & Nagarajan, G. (2021). Webapp service for booking handyman using MongoDB, express JS, React JS, node JS. 2021 3rd International Conference on Signal Processing and Communication, ICPSC 2021, 180–183. <https://doi.org/10.1109/ICSPC51351.2021.9451783>
- Sadek, J., Craig, D., & Trenell, M. (2022). Design and implementation of medical searching systems based on microservices and serverless architectures. *Procedia Computer Science*, 196, 615–622. <https://doi.org/10.1016/j.procs.2021.12.056>
- Saundariya, K., Abirami, M., Senthil, K. R., Prabakaran, D., Srimathi, B., & Nagarajan, G. (2021). Webapp service for booking handyman using MongoDB, express JS, React JS, node JS. 2021 3rd International Conference on Signal Processing and Communication, ICPSC 2021, 180–183. <https://doi.org/10.1109/ICSPC51351.2021.9451783>
- Swacha, J., & Kulpa, A. (2023). Evolution of popularity and multiaspectual comparison of widely used web development frameworks. *Electronics*, 12(17), 3563. <https://doi.org/10.3390/electronics12173563>
- Tesoriero, R., Rueda, A., Gallud, J. A., Lozano, M. D., & Fernando, A. (2022). Transformation Architecture for Multi-Layered WebApp Source Code Generation. *IEEE Access*, 10, 5223–5237. <https://doi.org/10.1109/ACCESS.2022.3141702>
- Vale, G., Correia, F. F., Guerra, E., Guerra, E. M., De, T., Rosa, O., Fritzsche, J., & Bogner, J. (2022). Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs Thatiane de Oliveira Rosa Instituto Federal do Tocantins

Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs. <https://doi.org/10.48550/arXiv.2201.03598>

Yanakiev, I., Lazar, B. M., & Capiluppi, A. (2025). Applying SOLID principles for the refactoring of legacy code: An experience report. *Journal of Systems and Software*, 220. <https://doi.org/10.1016/j.jss.2024.112254>

Zurita, E., Wiley, H., Carhuallanqui, S., Libia, M. ;, Calvanapón, A., & Alicia, F. (2022). Control interno en la rentabilidad de una empresa de servicios generales-Perú. *Vision de Futuro*, 27. <https://doi.org/10.36995/j.visiondefuturo.2023.27.01.005.es>

Anexos

Anexo A

Documento de Requerimientos y Prototipos

Objetivo	238
Alcance	238
Definiciones	238
Dueño del documento	238
Descripción	238
Propósito	238
Roles y responsabilidades del equipo Scrum	238
Actores involucrados	239
Requerimientos Funcionales	239
Requerimientos No Funcionales	240
Caso de Uso y Prototipos	240
UC001 - Módulo de gestión de usuario	241
UC002 - Módulo de películas	242
UC003 - Módulo de favoritos	244
UC004 - Módulo de calificaciones	246
UC005 - Módulo de lista de seguimiento	247
UC006 - Módulo de accesibilidad	250

Objetivo

El objetivo de este documento es ayudar a disminuir la complejidad inicial del mantenimiento del software usando 3 capas basadas en Clean Architecture.

Alcance

El documento está dirigido a desarrolladores y personal técnico que participan en el ciclo de vida del software, cubre los lineamientos y consideraciones para implementar y mantener el software bajo la arquitectura de 3 capas y se añade las buenas prácticas de integración.

Definiciones

Clean Architecture : Arquitectura que está diseñada en capas concéntricas con el fin de lograr la alta mantenibilidad, independencia, facilidad de prueba y separación de responsabilidad en sus 4 capas: capa de entidades, capa de casos de uso, capa de interfaces o adaptadores, capa de presentación.

Dueño del documento

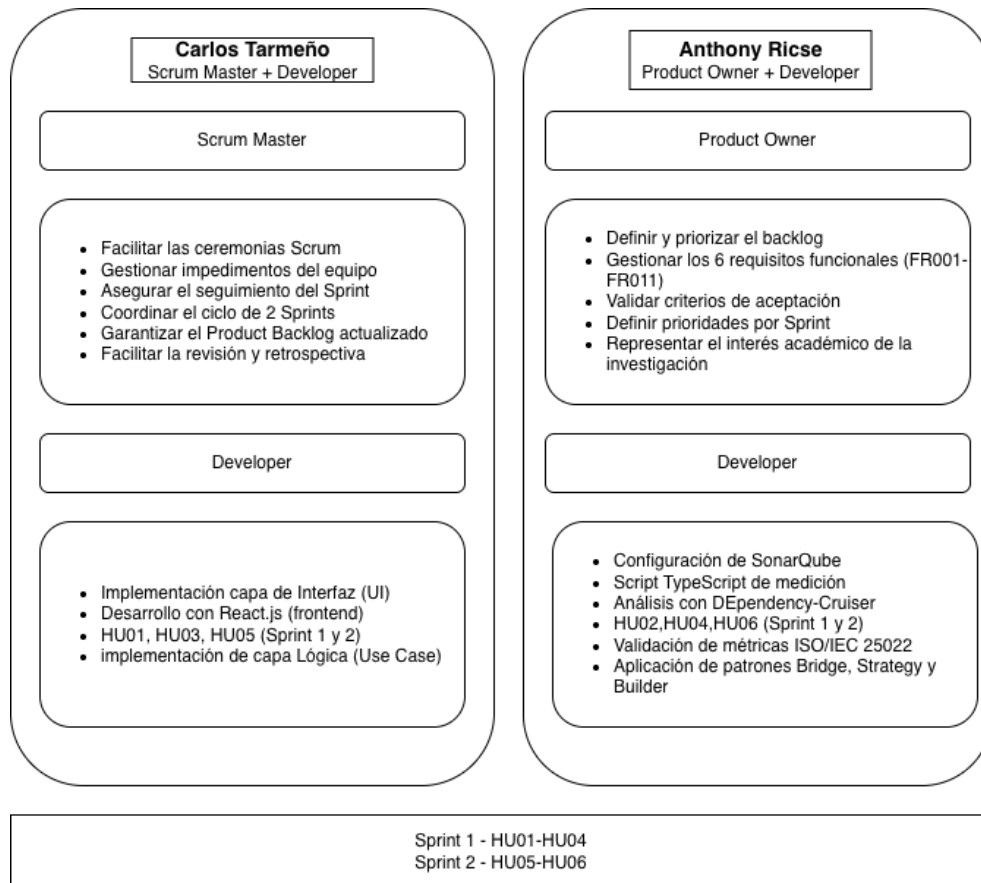
Anthony Ricse Perez y Carlos Tarmeño autores de la Tesis “Arquitectura para mitigar la complejidad inicial del mantenimiento del software”.

Descripción

- Propósito

Reducir los problemas de mantenibilidad que suelen presentarse en los proyectos que implementan Clean Architecture, estableciendo lineamientos simplificados basados en una arquitectura de 3 capas que permitan un desarrollo más ordenado, escalable y fácil de entender desde las primeras etapas del proyecto.

- Roles y responsabilidades del equipo Scrum



- Actores involucrados
 - Usuario Normal, quien accede al sistema para revisar el contenido de las películas.
 - Usuario Visitante, quien tiene un acceso temporal para tener una lista de seguimiento de sus películas
 - Usuario Miembro, quien además de observar y tener su lista de seguimiento, adicionalmente podrá tener sus películas favoritas y calificarlas.

Requerimientos Funcionales

Los requerimientos funcionales son los siguientes:

- Funcionalidades del módulo gestión de usuario:
 - FR001 - Ingresar y cerrar sesión de la aplicación
- Funcionalidades del módulo Películas:
 - FR002 - Visualizar películas
 - FR003 - Visualizar carrusel de películas

- Funcionalidades del módulo de favoritos:
 - FR004 - Agregar películas favoritas
 - FR005 - Eliminar películas favoritas
- Funcionalidades del módulo de calificaciones:
 - FR006 - Agregar una calificación a una película
 - FR007 - Eliminar una calificación a una película
 - FR008 - Actualizar una calificación a una película
- Funcionalidades del módulo de lista de seguimiento:
 - FR009 - Agregar películas a mi lista de seguimiento
 - FR010 - Eliminar películas de mi lista de seguimiento
- Funcionalidades del módulo de accesibilidad
 - FR011 - Visualizar el modo oscuro

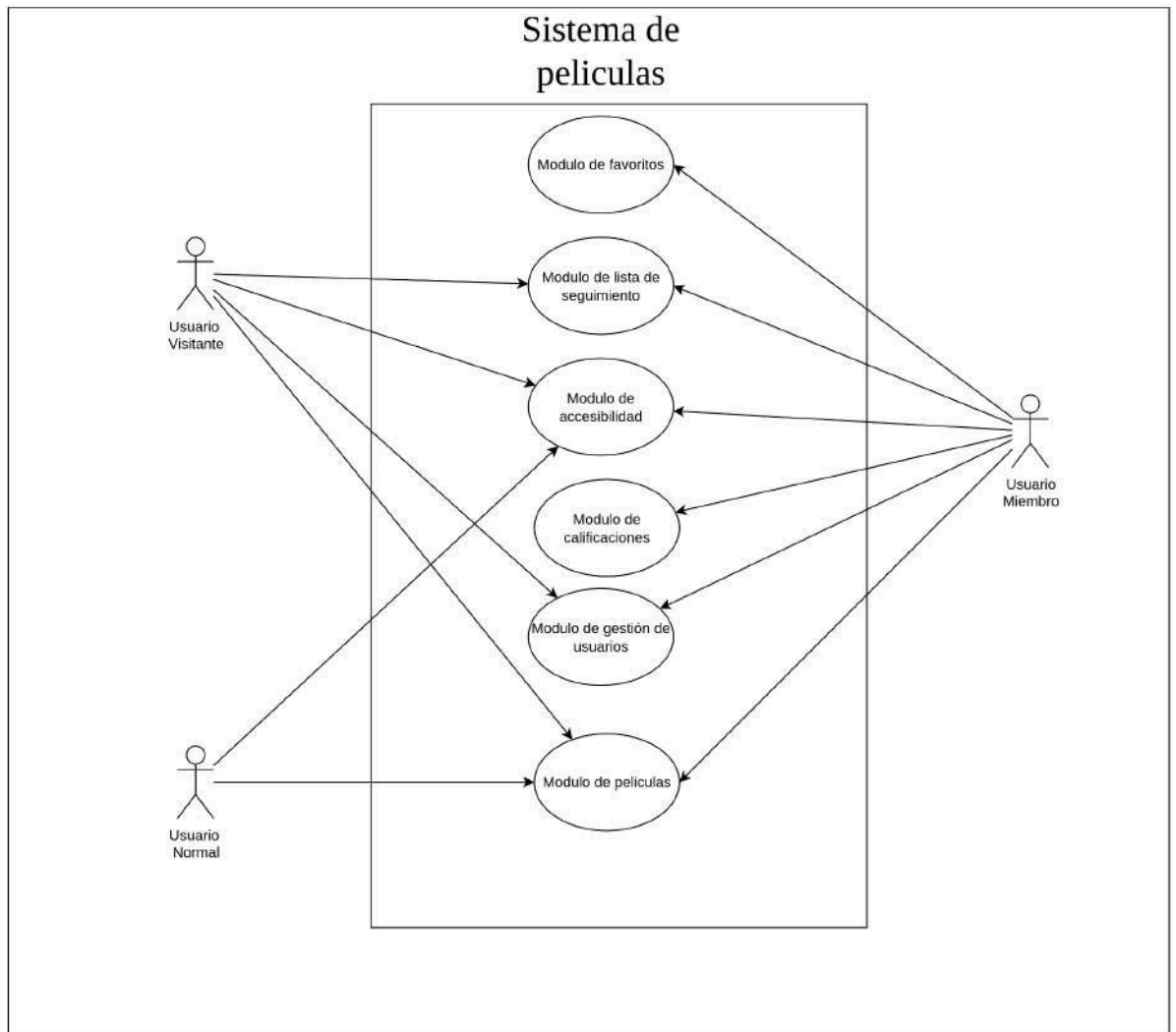
Requerimientos No Funcionales

Los requerimientos no funcionales son los siguientes

- Requerimiento de usabilidad : El sistema web permite al usuario interactuar de una manera fácil con sus películas.
- Requerimiento de portabilidad : Se debe tener acceso a internet y un dispositivo que permite acceder a la página.
- Mantenibilidad: Este sistema es fácil de mantener, además que el código cuenta con una arquitectura que le permite desacoplar los módulos, los servicios y la lógica de negocio brindando, además de una facilidad de actualización.
- Rendimiento: Los tiempos de respuesta deben ser rápidos para el uso del usuario.

Caso de Uso y Prototipos

En la siguiente figura se muestra el diagrama de casos de uso del sistema de películas con sus módulos.



UC001 - Modulo de gestion de usuario

En este caso de uso se valida el ingreso de un usuario al sistema.

Actor(es)

- Usuario normal, Usuario visitante y Usuario miembro

Pre-condiciones

- Tener acceso a internet

Post-condiciones

- El sistema permite el ingreso del usuario

Flujo básico

- Ingresa a la página web
- Ingresa al sistema

- Sale del sistema

Flujo de Excepción

- Fallo en el servicio

Criterios De aceptación

- El sistema valida la existencia del usuario

Requerimiento asociados

ID Requerimiento	Nombre del Requerimiento
FR001	Ingresar y cerrar sesión de la aplicación

Prototipo

Page 1

https://www.sistemadepelicula.io

Sistema de películas

Iniciar sesión

Usuario

iniciar sesion

continuar como invitado

UC002 - Módulo de películas

En este caso de uso se realiza la gestión de películas, en el cual se mostrará la cartelera de películas existentes.

Actor(es)

- Usuario normal, Usuario visitante y Usuario miembro

Pre-condiciones

- Se tiene que ingresar correctamente con el servicio de películas activo

Post-condiciones

- El sistema permite la gestión de películas

Flujo básico

- El usuario ingresa al sistema
- El usuario logra visualizar sus películas
- El usuario revisa el detalle de una película

Flujo de Excepción

- Fallos con el servicio de película

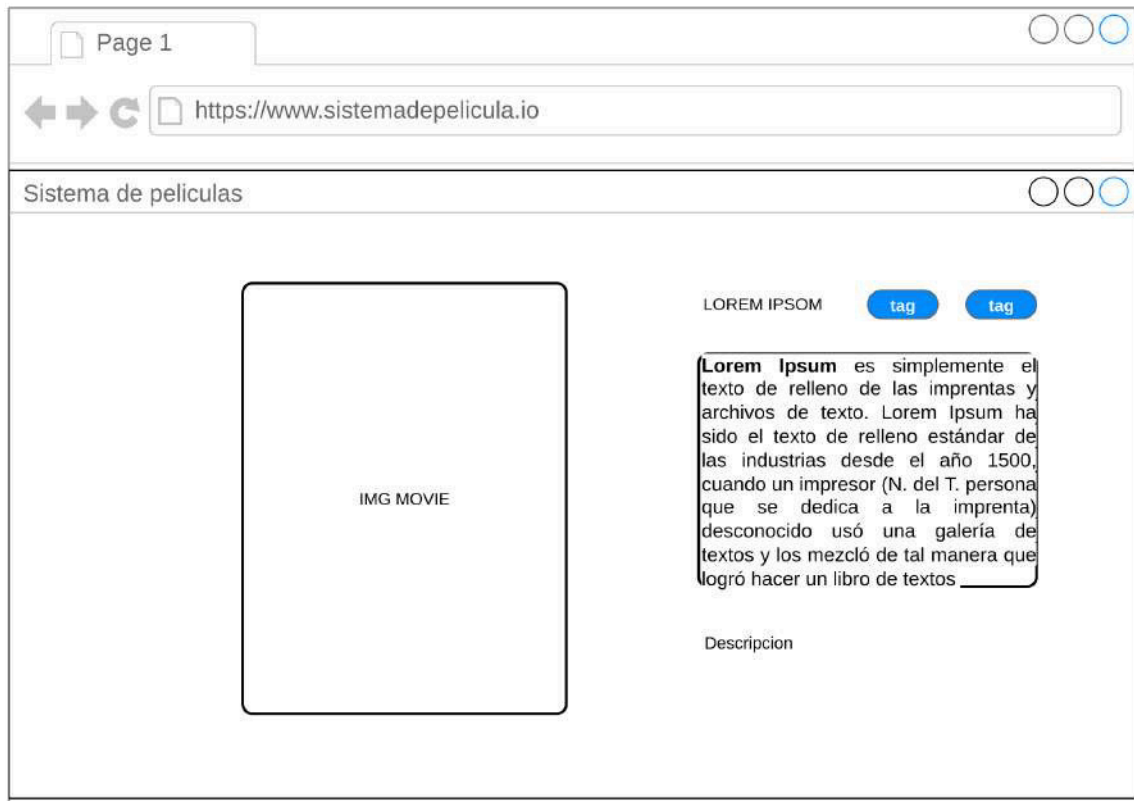
Criterios De aceptación

- El sistema logra mostrar el detalle de la película del usuario

Requerimiento asociados

ID Requerimiento	Nombre del Requerimiento
FR002	Visualizar película
FR003	Visualizar carrusel de películas

Prototipo



UC003 - Modulo de Favoritos

En este caso de uso se asigna la gestión de las película favoritas dentro del sistema

Actor(es)

- Usuario Miembro

Pre-condiciones

- El usuario debe estar logueado

Post-condiciones

- El sistema visualiza sus películas favoritas

Flujo básico

- El usuario ingresa al sistema
- El usuario se loguea en el sistema
- El usuario añade a favoritos su película

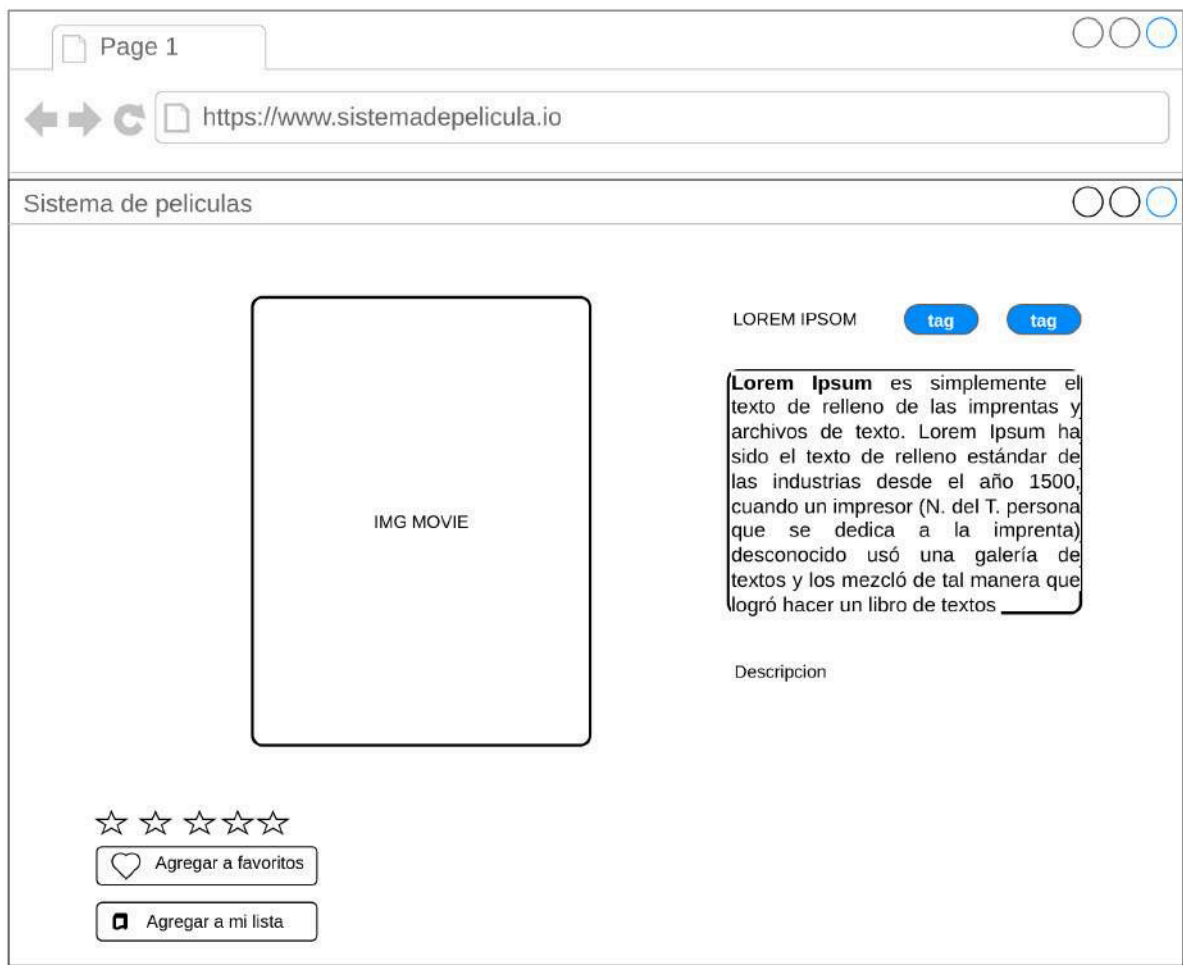
Criterios De aceptación

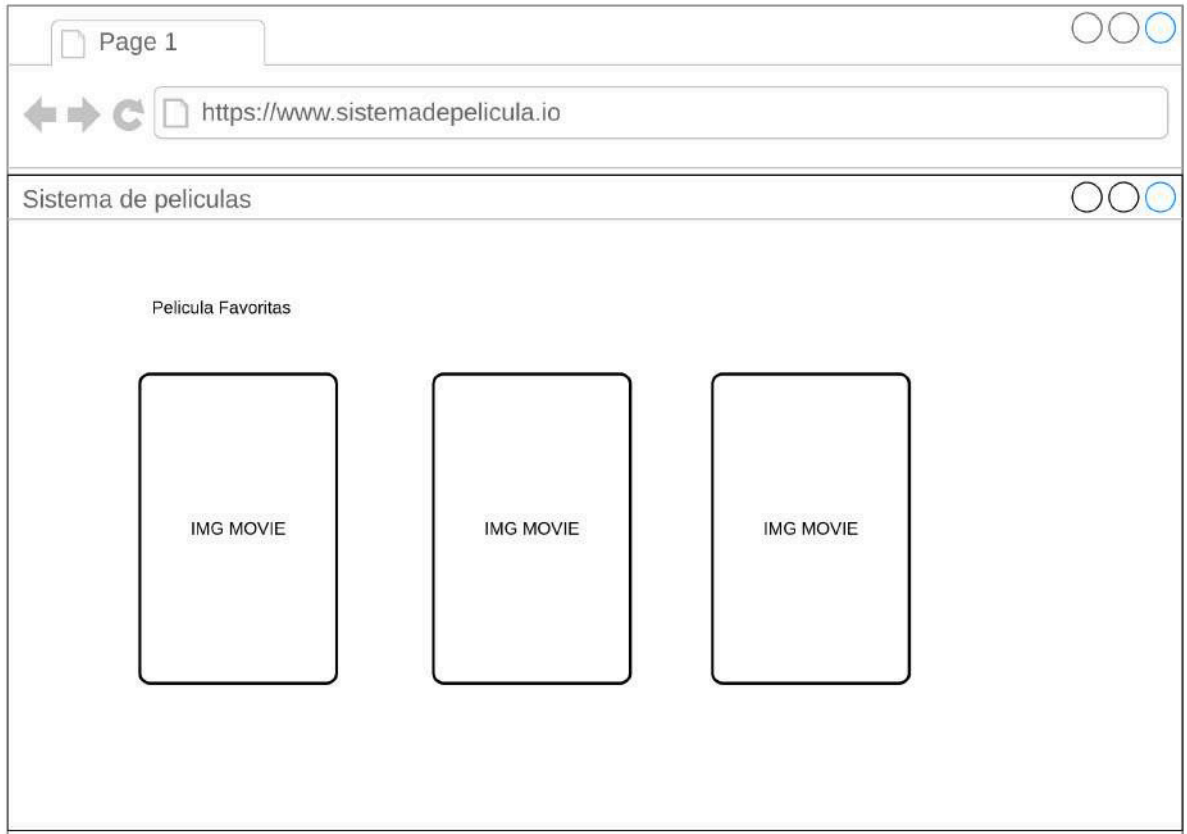
- El usuario visualiza sus películas favoritas

Requerimiento asociados

ID Requerimiento	Nombre del Requerimiento
FR004	Agregar películas favoritas
FR005	Eliminar películas favoritas

Prototipo





UC004 - Módulo de Calificaciones

En este caso de uso se asigna la calificación de las película dentro del sistema

Actor(es)

- Usuario Miembro

Pre-condiciones

- El usuario debe estar logueado

Post-condiciones

- El sistema visualizar la calificación de su película

Flujo básico

- El usuario ingresa al sistema
- El usuario se loguea en el sistema
- El usuario añade a calificaciones a sus películas

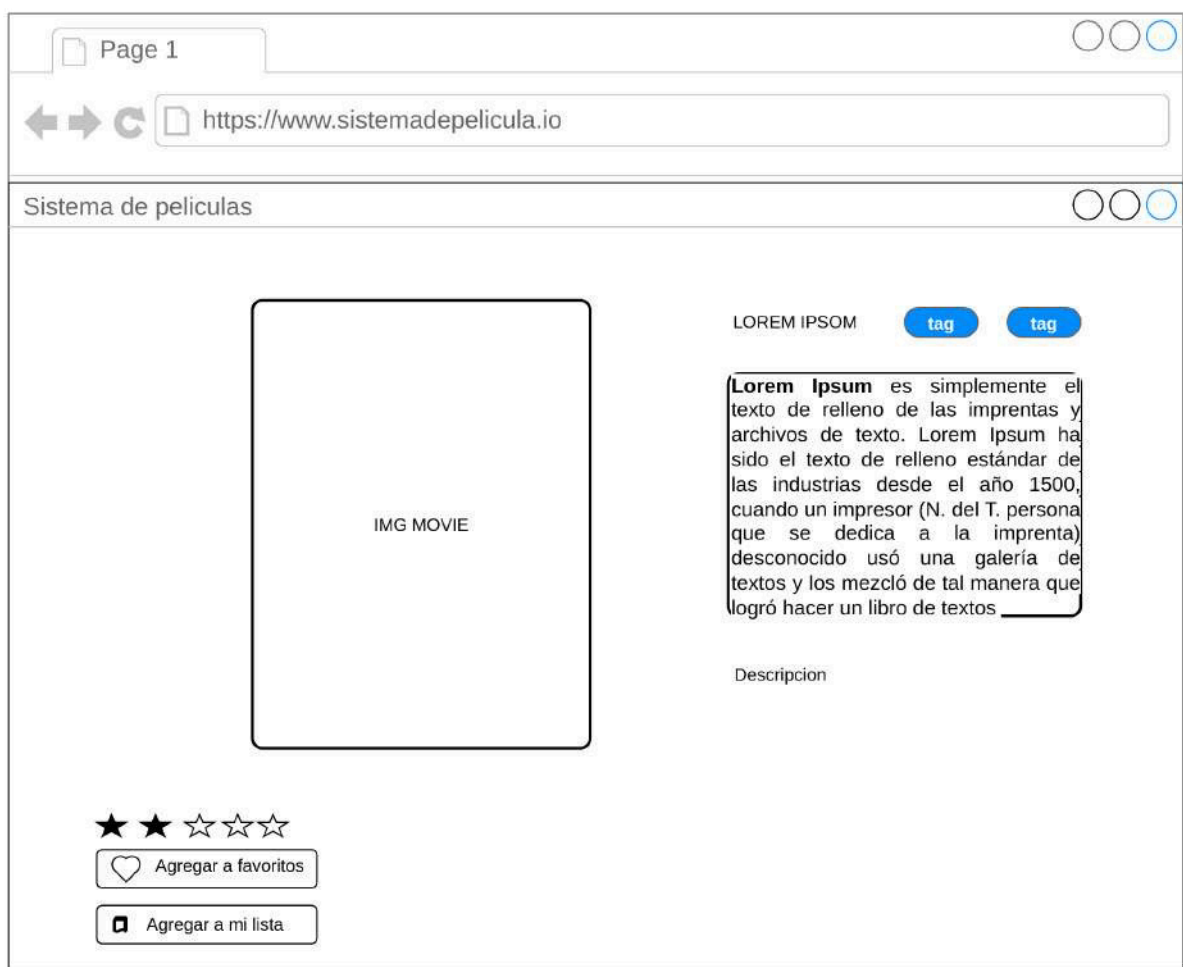
Criterios de aceptación

- El usuario visualiza sus películas calificadas

Requerimiento asociados

ID Requerimiento	Nombre del Requerimiento
FR006	Agregar una calificación a una película
FR007	Eliminar una calificación a una película
FR008	Actualizar una calificación a una película

Prototipo



UC005 - Módulo de lista de seguimiento

En este caso de uso se asigna la gestión de la lista de seguimiento de las películas dentro del sistema

Actor(es)

- Usuario visitante y Usuario Miembro

Pre-condiciones

- El usuario debe estar logueado

Post-condiciones

- El sistema visualiza su lista de películas seguidas

Flujo básico

- El usuario ingresa al sistema
- El usuario se loguea en el sistema
- El usuario añade una película a su lista de seguimiento

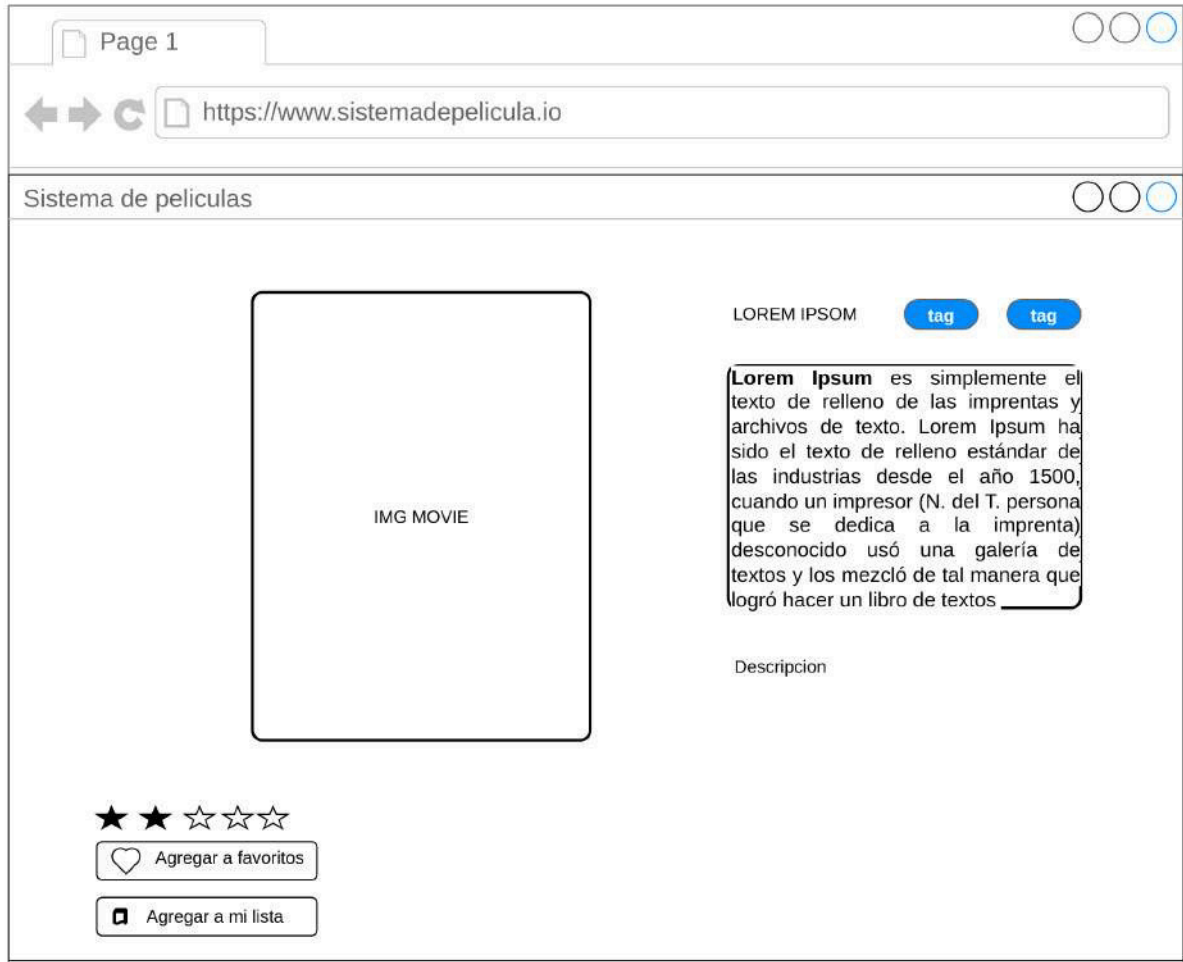
Criterios de aceptación

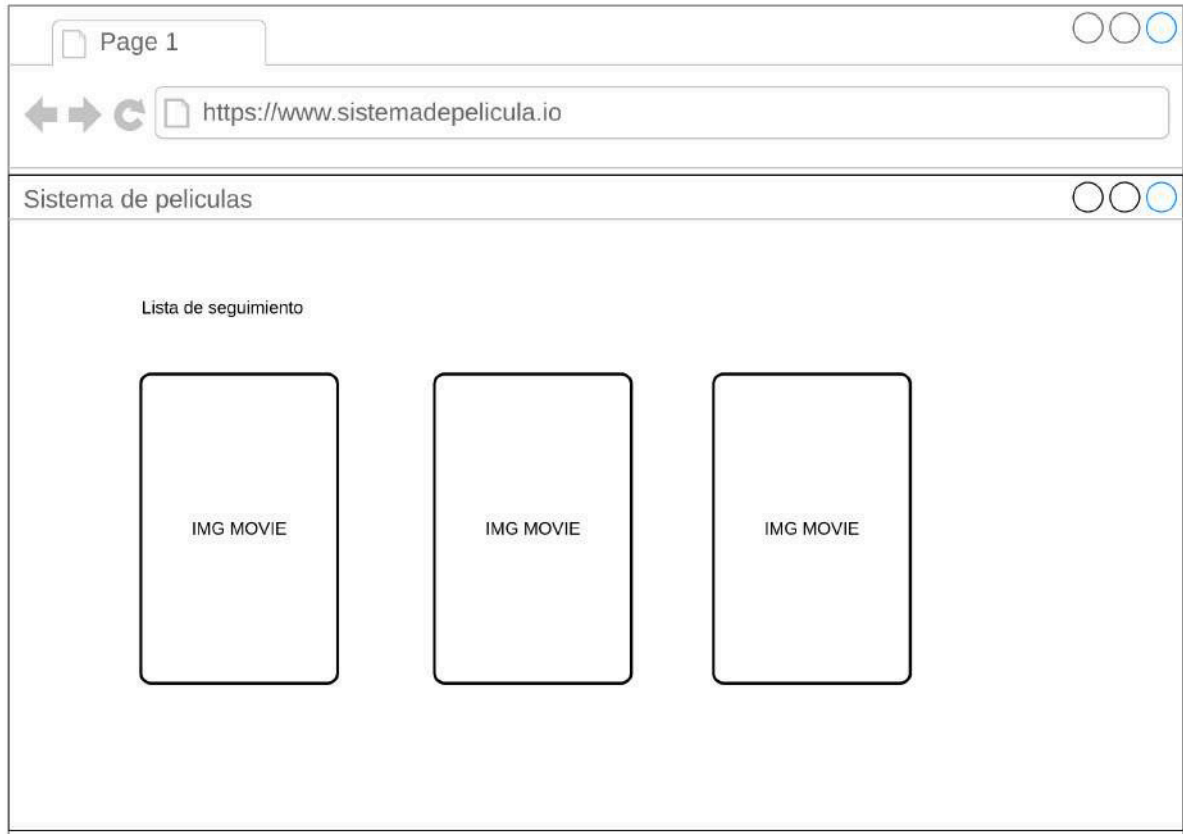
- El usuario su nueva película en la lista de seguimiento

Requerimiento asociados

ID Requerimiento	Nombre del Requerimiento
FR009	Agregar películas a mi lista de seguimiento
FR010	Eliminar películas de mi lista de seguimiento

Prototipo





UC006 - Módulo de accesibilidad

En este caso de uso se asigna la accesibilidad para los usuarios dentro del sistema.

Actor(es)

- Usuario normal, Usuario visitante y Usuario Miembro

Pre-condiciones

- El usuario debe acceder a internet

Post-condiciones

- El sistema visualiza su accesibilidad.

Flujo básico

- El usuario ingresa al sistema
- El usuario selecciona una accesibilidad
- El usuario visualiza el cambio dentro del sistema

Criterios de aceptación

- El usuario se visualiza el modo oscuro del sistema

Requerimiento asociados

ID Requerimiento	Nombre del Requerimiento
FR011	Visualizar el modo oscuro

Prototipo

