# Módulo para la gestión de grupos de estudio e integración universitaria UIS en la plataforma COMA del grupo CALUMET

Andrés Felipe Olivar Gutíerrez & Danna Valentina Prada Briceño

Trabajo de Grado para Optar al Título de Profesional en Ingeniería de Sistemas

Director

Luis Ignacio González Ramírez

Ingeniero de Sistemas Magister en Informática

Universidad Industrial de Santander

Ingeniería de Sistemas

Bucaramanga

2026

**Agradecimientos**

Expresamos nuestro más sincero agradecimiento a todas las personas y entidades que, de una u otra forma, hicieron posible la realización de este proyecto de grado.

A nuestras familias, por ser nuestro soporte incondicional, brindándonos su amor, paciencia y aliento en los momentos de dificultad. En especial a nuestros padres, quienes siempre creyeron en las capacidades y nos apoyaron en cada paso de este camino académico.

A nuestro director de trabajo de grado, el ingeniero Luis Ignacio González, por su orientación, dedicación y compromiso. Su guía fue fundamental para llevar a cabo este proyecto con rigor y excelencia.

A la Universidad Industrial de Santande, por ofrecernos la oportunidad de formarnos profesionalmente, así como por los recursos y herramientas académicas que nos brindaron durante la trayectoria universitaria.

A nuestros compañeros y amigos, quienes con sus palabras de ánimo y su apoyo nos recordaron que los desafíos son más llevaderos cuando se comparten.

Finalmente, agradecemos a todos los estudiantes y docentes que participaron de manera directa o indirecta en este trabajo, aportando sus experiencias y conocimientos para enriquecer este proyecto.

A todos, muchas gracias.

**Tabla de Contenido**

Introducción . . . . . 15

1 Generalidades del proyecto 16

1.1 Planteamiento y justificación del problema . . . . . 16

1.1.1 Definiciones, acrónimos y abreviaciones . . . . . 18

1.1.2 Alcance del proyecto . . . . . 21

1.2 Objetivos . . . . . 22

1.2.1 Objetivo general . . . . . 22

1.2.2 Objetivos específicos . . . . . 22

2 Visión general del sistema 23

2.1 Usuarios del sistema . . . . . 23

2.2 Restricciones generales . . . . . 24

2.2.1 Tecnológicas . . . . . 24

2.2.2 Normativas . . . . . 25

2.2.3 Regulatorias . . . . . 26

2.3 Cronograma . . . . . 26

2.4 Presupuesto . . . . . 27

3 Marco de referencia 29

3.1 Fundamentos teóricos . . . . . 29

3.1.1 Integración universitaria y aprendizaje colectivo . . . . . 29

3.1.2 Tecnologías educativas y plataformas digitales . . . . . 29

3.1.3 Arquitecturas de software . . . . . 30

3.1.4 Microservicios y arquitectura hexagonal . . . . . 30

3.1.5 Marcos de trabajo ágiles en el desarrollo del software . . . . . 32

3.1.6 Ciclo de vida del software . . . . . 32
3.1.7 Buenas prácticas y clean code . . . . . 32
3.1.8 Pruebas de software y evaluaciones de calidad . . . . . 33
3.1.9 Máquinas virtuales . . . . . 33
3.2 Antecedentes del tema . . . . . 34
3.2.1 Sistemas de gestión de aprendizaje . . . . . 34
3.2.2 Tendencias actuales . . . . . 34

4 Planificación y análisis 35
4.1 Análisis de requerimientos . . . . . 36
4.1.1 Estándar ISO/IEC/IEEE 29148 — Ingeniería de Requerimientos . . . . . 36
4.1.2 Principios del estándar . . . . . 36
4.1.3 Requerimientos funcionales . . . . . 38
4.1.4 Requerimientos no funcionales . . . . . 72
4.2 Proceso de trabajo y marcos de trabajo ágil . . . . . 81
4.3 Diferencias entre épicas, historias de usuario y sprints . . . . . 82
4.3.1 Épicas . . . . . 82
4.3.2 Historias de usuario . . . . . 82
4.3.3 Sprints . . . . . 82
4.3.4 Relación entre épicas, historias y sprints . . . . . 83
4.3.5 Historias de usuario y configuración del trabajo en JIRA . . . . . 83
4.3.6 Primer sprint: Base del microservicio . . . . . 87
4.3.7 Segundo sprint: Primera versión del chat . . . . . 89
4.3.8 Tercer sprint: Primera versión del mural . . . . . 90
4.3.9 Cuarto sprint: Versión completa panel de recursos . . . . . 91
4.3.10 Quinto sprint: Funciones de calificación . . . . . 92
4.3.11 Trabajo posterior a los sprints . . . . . 94

5 Diseño 98

5.1 Diagramas de caso de uso . . . . . 98

5.1.1 Sistema general para usuario solicitante . . . . . 98

5.1.2 Sistema general para líder de grupo . . . . . 100

5.1.3 Funciones de un miembro de grupo en el mural . . . . . 102

5.1.4 Funciones del líder en el mural . . . . . 104

5.1.5 Funciones de un miembro en el chat grupal . . . . . 105

5.1.6 Funciones del líder en el chat grupal . . . . . 106

5.1.7 Funciones del Moderador en el chat grupal . . . . . 107

5.1.8 Funciones de los miembros en el panel de recursos . . . . . 108

5.1.9 Funciones de un usuario solicitante en el BDP . . . . . 109

5.2 Diagrama de clases . . . . . 109

5.2.1 Dominio de Usuario/Perfil y Grupo de estudio . . . . . 110

5.2.2 Dominio del Mural y el chat grupal . . . . . 110

5.2.3 Dominio del Botón de pánico y Notificación . . . . . 111

5.2.4 Dominio BDP, Calificaciones, reseñas y tops semanales . . . . . 112

5.3 Modelo entidad relación físico . . . . . 112

5.4 Mockups . . . . . 114

5.4.1 Vista de los grupos del usuario . . . . . 114

5.4.2 Vista del proceso de publicación . . . . . 114

5.4.3 Vista del mural del grupo . . . . . 115

5.4.4 Vista del panel de recursos . . . . . 115

5.4.5 Vista del panel de chats y miembros de un grupo de estudio . . . . . 116

5.5 Diagrama de arquitectura . . . . . 116

5.6 Diagrama de componentes . . . . . 117

6 Implementación 118

6.1 Estándares de ingeniería utilizados . . . . . 118

6.2 Selección de herramientas . . . . . 120
6.2.1 Manejo de base de datos: MySQL . . . . . 120
6.2.2 Entornos de programación: IntelliJ IDEA, NetBeans y Visual Studio Code . . . . . 121
6.2.3 Framework utilizado: Spring Boot . . . . . 121
6.2.4 Lenguaje backend: Java . . . . . 121
6.2.5 Lenguajes frontend: Herramienta interna de CALUMET Suamox-Elise . . . . . 122
6.2.6 Herramienta de despliegue: Nginx + Docker . . . . . 123
6.2.7 Gestión de dependencias: Maven . . . . . 124
6.2.8 Documentación de API: Swagger . . . . . 124
6.2.9 Protocolo de comunicación: WebSocket . . . . . 124
6.2.10 Control de versiones: Git . . . . . 125
6.2.11 Herramienta de despliegue: Docker . . . . . 125
6.3 Arquitectura de la solución . . . . . 126
6.3.1 Arquitectura hexagonal . . . . . 126
6.3.2 Domain-Driven Design (DDD-lite) . . . . . 126
6.3.3 Estructura del Backend . . . . . 127
6.3.4 Patrones de Diseño Implementados . . . . . 128
6.3.5 Patrón Observer (WebSocket) . . . . . 130
6.3.6 Spring-managed Singletons . . . . . 130
6.4 Metodología de desarrollo . . . . . 131
6.5 Gestión del código fuente . . . . . 131
6.5.1 Flujo de trabajo . . . . . 131
6.5.2 Evidencias del proceso de desarrollo . . . . . 134

7 Pruebas y validación 134
7.1 Pruebas de caja negra . . . . . 134
7.1.1 Gestión de Grupos de Estudio . . . . . 135
7.1.2 Unión e Inscripción a Grupos . . . . . 136

7.1.3 Publicaciones y Recursos . . . . . 138
7.1.4 Chat de Grupo . . . . . 140
7.1.5 Funciones Especiales . . . . . 141
7.2 Pruebas de caja Blanca . . . . . 144
7.2.1 Pruebas de Grupo de Estudio . . . . . 145
7.2.2 Publicaciones y recursos . . . . . 147
7.2.3 Chat de Grupo . . . . . 148
7.2.4 Funciones especiales . . . . . 149
7.2.5 Pruebas unitarias automatizadas . . . . . 150
7.2.6 Resultados finales del sistema . . . . . 150
7.3 Capacitación y documentación . . . . . 154

8 Despliegue 154
8.1 Entorno de desarrollo local . . . . . 154
8.2 Entorno de integración con contenedores . . . . . 155
8.3 Entorno de pruebas institucional . . . . . 156
8.4 Entorno de producción . . . . . 157

9 Conclusiones 158

10 Trabajo futuro 159
Referencias Bibliográficas . . . . . 160

## Lista de Tablas

1 Proyección del presupuesto . . . . . 28

2 Criterios de calidad de requisitos según ISO/IEC/IEEE 29148 . . . . . 37

3 Distribución de roles y participación en Scrum . . . . . 81

4 Comparativo entre épicas, historias de usuario y sprints . . . . . 83

5 Relación entre calificación Fibonacci y días máximos de desarrollo . . . . . 86

6 Distribución de tareas finalizadas en JIRA . . . . . 97

7 Paquetes del design system Elise . . . . . 123

8 Estadísticas del repositorio Git . . . . . 134

9 Validación de campos obligatorios en creación de grupo de estudio . . . . . 146

10 Casos de prueba para publicar en mural . . . . . 147

11 Casos de prueba para publicación de recursos en grupos de estudio . . . . . 148

12 Casos de prueba para envío de mensajes y audios en chat de grupo . . . . . 148

13 Casos de prueba para activación del botón de pánico . . . . . 149

14 Casos de prueba para búsqueda de compañeros con buscarBDP() . . . . . 149

## Lista de Figuras

1 Cronograma general del proyecto . . . . . 28

2 Arquitectura monolítica en comparación con la arquitectura de microservicios. . . . 31

3 Vista desde el panel JIRA de una historia de usuario sencilla . . . . . 84

4 Épica documentada del chat del grupo de estudio . . . . . 85

5 Vista del backlog en Jira . . . . . 86

6 Cronograma y visualización del primer sprint . . . . . 89

7 Cronograma y visualización del segundo sprint . . . . . 90

8 Cronograma y visualización del tercer sprint . . . . . 91

9 Cronograma y visualización del cuarto sprint . . . . . 91

10 Cronograma y visualización del quinto sprint . . . . . 92

11 Actividades por hacer en los sprints: resumen en jira . . . . . 93

12 Cronograma y relación Épicas-Sprints . . . . . 93

13 Etiquetas JIRA para el backlog . . . . . 94

14 Backlog del trabajo posterior a los sprints . . . . . 95

15 Reporte desde el BugTracker . . . . . 95

16 Reportes en la base de datos . . . . . 96

17 Reporte del BugTracker en JIRA . . . . . 96

18 Rama del reporte en github . . . . . 96

19 Estado final de las épicas en JIRA . . . . . 97

20 Sistema general para usuario solicitante . . . . . 98

21 Sistema general para lider de grupo . . . . . 100

22 Casos de uso de los miembros en el mural . . . . . 102

23 Funciones del líder en el mural . . . . . 104

24 Funciones de un miembro en el chat grupal . . . 105
25 Funciones del líder en el chat grupal . . . 106
26 Funciones del Moderador en el chat grupal . . . 107
27 Funciones del líder en el chat grupal . . . 108
28 Funciones de un usuario solicitante en el BDP . . . 109
29 Diagrama de clases para el dominio de Usuarios y Grupos de estudio . . . 110
30 Diagrama de clases para el dominio del Mural y Chat grupal . . . 111
31 Digrama de clases para el dominio del Botón de pánico y Notificación . . . 111
32 Digrama de clases para el dominio del BDP, Calificaciones, reseñas y tops semanales 112
33 Modelo ENTIDAD-RELACIÓN MySQL . . . 113
34 Vista de grupos del usuario . . . 114
35 Vista botón publicación Mural del grupo . . . 114
36 Panel del mural del grupo . . . 115
37 Vista de recursos dentro de un grupo . . . 115
38 Vista de chats y miembros dentro de un grupo . . . 116
39 Diagrama de arquitectura . . . 116
40 Diagrama de arquitectura de componentes . . . 117

41 Ejemplo de ramas creadas en GitHub . . . 132
42 Ejemplo de commit en GitHub . . . 133

43 Resultados de las pruebas usando mvn test . . . 150
44 Panel de grupos de estudio del usuario . . . 151
45 Vista final del chat grupal . . . 152
46 Vista final del mural de publicaciones . . . 152
47 Vista final del panel de recursos . . . 153
48 Vista del Buscador de parches . . . 153
49 Vista del Botón del pánico . . . 154

50 Contenedores de ambos servicios ejecutándose de forma independiente . . . . . 157

## Lista de apéndices

**Los apéndices están disponibles en el repositorio institucional**

**Apéndice A.** Manual de usuario para el uso de StudyParche

**Apéndice B.** Pruebas Unitarias

**Apéndice C.** Validación de calidad

## Resumen

**Título:** Módulo para la gestión de grupos de estudio e integración universitaria UIS en la plataforma COMA del grupo CALUMET*

**Autor:** Andrés Felipe Olivar Gutiérrez y Danna Valentina Prada Briceño**

**Palabras clave:** Grupo de estudio, SEA-ASAE, COMA, Ciclo básico.

**Descripción:** En la Universidad Industrial de Santander, un alto número de estudiantes presenta dificultades en asignaturas de ciclo básico, lo que con frecuencia conduce a la repetición de estas. Aunque existen programas de acompañamiento académico como el SEA, no se dispone de herramientas institucionales que fomenten la creación de entornos sociales informales que favorezcan el aprendizaje colaborativo. En un contexto donde la interacción digital es parte fundamental de la vida estudiantil, las plataformas universitarias actuales no ofrecen espacios que promuevan encuentros académicos casuales entre los estudiantes.

Este trabajo propone el diseño y desarrollo de un microservicio integrado a la plataforma COMA de la UIS, accesible tanto desde entornos web como móviles. Su objetivo es proporcionar a los estudiantes un espacio institucional para organizar encuentros informales, compartir recursos y fortalecer dinámicas sociales de aprendizaje dentro de la universidad.

El proyecto aborda el ciclo de vida completo del microservicio, desde el análisis de requerimientos y la definición de alcance, hasta la elaboración de diagramas, arquitecturas, planeación con metodologías ágiles e implementación. Se busca garantizar calidad y buenas prácticas en cada etapa, ofreciendo una solución tecnológica innovadora que responda a las necesidades académicas y sociales de los estudiantes.


---

* Trabajo de Grado.

** Universidad Industrial de Santander. Ingeniería de Sistemas. Director: Luis Ignacio González González. Magister en Informática.

## Abstract

**Title:** Module for the Management of Study Groups and University Integration at UIS in the COMA Platform of the CALUMET Group [*]

**Author:** Andrés Felipe Olivar Gutiérrez and Danna Valentina Prada Briceño[**]

**Keywords:** Study group, SEA-ASAE, COMA, Basic cycle.

**Description:** At the Universidad Industrial de Santander, a significant number of students face difficulties in basic cycle courses, often leading to course repetition. Although academic support programs such as the Open Education System (SEA) exist, there are no institutional tools that foster the creation of informal social environments that promote collaborative learning. In a context where digital interaction is a fundamental aspect of student life, current university platforms do not provide spaces that encourage casual academic encounters among students.

This work proposes the design and development of a microservice integrated into the COMA platform of the UIS, accessible from both web and mobile environments. Its main goal is to provide students with an institutional space to organize informal meetings, share resources, and strengthen social learning dynamics within the university.

The project covers the complete life cycle of the microservice, from requirements analysis and scope definition to the elaboration of diagrams, architectures, agile planning, and implementation. The aim is to ensure quality and good practices at each stage, delivering an innovative technological solution that addresses the academic and social needs of students.


---

[*] Graduation Thesis.

[**] Industrial University of Santander. Systems Engineering. Advisor: Luis Ignacio González González. Master in Computer Science.

## Introducción

En la actualidad, las dinámicas de la enseñanza evolucionan cada vez más a medida que la tecnología avanza; ya no se limitan a las aulas ni a los métodos tradicionales. Esto, sumado a que las interacciones entre estudiantes, las conversaciones informales y el intercambio espontáneo de conocimientos han demostrado ser componentes esenciales para fortalecer la comprensión y el desarrollo académico, resalta la importancia de promover espacios colaborativos. Sin embargo, a pesar de la creciente digitalización de los procesos educativos, muchos entornos universitarios carecen de plataformas virtuales que fomenten este tipo de aprendizaje. Las universidades, como espacios de formación integral, enfrentan el desafío de incorporar herramientas digitales que no solo faciliten la gestión académica, sino que también promuevan la interacción entre estudiantes y fortalezcan los procesos de aprendizaje colectivo.

En este contexto, En la Universidad Industrial de Santander (UIS), diversos estudiantes enfrentan dificultades a lo largo de su proceso formativo, especialmente en la consolidación de conocimientos y en la búsqueda de espacios colaborativos para el aprendizaje. Aunque la institución dispone de programas de apoyo académico como el Sistema de Excelencia Académica (SEA), estos se centran principalmente en la formalidad del proceso de enseñanza y no contemplan entornos digitales que fomenten la interacción espontánea entre los estudiantes.

Por su parte, plataformas institucionales como Moodle cumplen un papel fundamental en la gestión de contenidos académicos, pero su diseño está orientado al control docente de los recursos y no a la creación de espacios que incentiven el encuentro entre pares o el aprendizaje informal. Diversos estudios coinciden en que los entornos informales de aprendizaje favorecen la comprensión de los contenidos y el desarrollo de habilidades blandas, ya que estimulan la colaboración espontánea y la construcción colectiva del conocimiento. Sin embargo, a pesar del avance tecnológico, las plataformas institucionales actuales no suelen ofrecer funcionalidades que promuevan este tipo de interacciones.

En el caso particular de la UIS, la plataforma COMA actúa como un punto central de integración para los procesos académicos de todas las escuelas, lo que la convierte en un entorno ideal

para incorporar un componente orientado al aprendizaje colaborativo. A partir de esta necesidad, surge la propuesta de desarrollar un microservicio para la creación y gestión de grupos de estudio, diseñado para incentivar la cooperación entre estudiantes y fortalecer la cultura de apoyo mutuo dentro de la universidad.

El presente proyecto de trabajo de grado aborda el ciclo completo de desarrollo de software, desde el análisis de requerimientos y el diseño arquitectónico hasta la implementación y el despliegue del microservicio, siguiendo buenas prácticas de ingeniería y metodologías ágiles. El objetivo principal será ofrecer una solución tecnológica escalable y de calidad, capaz de integrarse al ecosistema institucional y de fomentar espacios informales de aprendizaje que contribuyan a mejorar el desempeño académico y fortalecer la cohesión estudiantil en la comunidad universitaria.

# 1. Generalidades del proyecto

En este capítulo se presentan las generalidades del proyecto, abordando el planteamiento del problema y su justificación, así como los objetivos generales y específicos que guiaron el desarrollo. También se estableció el alcance del trabajo, junto con un conjunto de definiciones y acrónimos relevantes, con el fin de ofrecer un marco claro que permita comprender la base sobre la cual se construyó la propuesta.

## 1.1 Planteamiento y justificación del problema

En la Universidad Industrial de Santander, es frecuente que los estudiantes enfrenten dificultades académicas en asignaturas específicas, lo que genera desmotivación y retraso en su plan de estudios. Actualmente, la institución cuenta con programas de apoyo como el SEA-ASAE, estrategia institucional orientada a brindar acompañamiento en asignaturas del ciclo básico de ingeniería y ciencias de la UIS (Universidad Industrial de Santander, n.d.); sin embargo, aunque constituye una alternativa valiosa, presenta limitaciones en flexibilidad (incluye penalizaciones económicas por inasistencia) y en alcance (se centra principalmente en asignaturas de ciclo básico). A lo anterior se suma la falta de versatilidad en la organización de los horarios y en el proceso de asignación

de tutores, ya que los estudiantes no tienen la posibilidad de elegir al acompañante ni adaptar las sesiones a su disponibilidad. Debido a estas restricciones, los estudiantes suelen recurrir a apoyos informales entre compañeros con mejor desempeño académico, conformando grupos de estudio que han demostrado favorecer la comprensión de contenidos, la motivación y la persistencia en el aprendizaje (Johnson & Johnson, 2009; Topping & Ehly, 1998). Esta dinámica, además de facilitar el aprendizaje entre compañeros, fomenta la integración universitaria al brindar un espacio donde los estudiantes pueden intercambiar sus ideas, conocer nuevos puntos de vista y construir conocimiento.

Sin embargo, no todos los estudiantes encuentran sencillo acercarse directamente a sus compañeros de clase para solicitar ayuda académica. Aquellos con mayor timidez o inseguridad suelen enfrentar barreras para integrarse en grupos de estudio, lo que los lleva a aislarse y enfrentar las dificultades por su cuenta. En este sentido, una herramienta tecnológica puede ofrecer un primer puente que facilite el contacto y la organización de los grupos, reduciendo la barrera social inicial y promoviendo la inclusión académica (Carranza Gangotena et al., 2022).

En este contexto, resulta pertinente el uso de la plataforma COMA (Comunidad Académica), un entorno institucional desarrollado y gestionado por el grupo de investigación CALUMET de la UIS. COMA centraliza los portales académicos de escuelas y facultades, ofreciendo un espacio para la gestión de procesos académicos y de integración estudiantil, lo que la convierte en el escenario ideal para la incorporación de nuevas herramientas orientadas a fortalecer el aprendizaje.

Ante este panorama, se justifica el desarrollo de un microservicio para su integración en la plataforma COMA, que permita a los estudiantes la creación y gestión de grupos de estudio con programación de reuniones y lugares de encuentro; la administración de recursos compartidos entre los integrantes; la integración de un chat grupal temporal con la misma duración del grupo; y un mural colaborativo donde los estudiantes podrán publicar ejercicios, preguntas y materiales de apoyo, lo que permitirá la interacción incluso en momentos en que el grupo no esté activo. Asimismo, se incluirán herramientas innovadoras como un “botón de pánico” en el chat para solicitudes inmediatas de ayuda, sistemas de calificación y retroalimentación tanto para estudiantes como para

líderes de un grupo, y un mecanismo de notificaciones que mantendrá a los usuarios informados de actividades y recordatorios. Con ello se fomentaría la conexión entre quienes requieren refuerzo académico y quienes están dispuestos a ofrecerlo, el apoyo estaría disponible en cualquier etapa del proceso académico, se promoverían las relaciones académicas y el fortalecimiento colectivo del conocimiento en la comunidad UIS (Slavin, 2011).

### *1.1.1 Definiciones, acrónimos y abreviaciones*

**Microservicio:** Estilo de arquitectura de software en el cual una aplicación se construye como un conjunto de servicios pequeños, independientes y desplegables de forma autónoma. Cada microservicio se encarga de una funcionalidad específica, comunicándose con los demás a través de APIs ligeras.

**COMA:** Acrónimo de Comunidad Académica. Plataforma institucional desarrollada por el grupo CALUMET.

**CALUMET:** Grupo de investigación de la Universidad Industrial de Santander dedicado al desarrollo de tecnologías educativas, plataformas institucionales y soluciones de software orientadas al aprendizaje y la gestión académica.

**Arquitectura de software:** Conjunto de decisiones estructurales que definen la organización, los componentes, las relaciones y los principios de diseño que guían la construcción de un sistema software.

**Arquitectura Hexagonal:** Estilo arquitectónico también conocido como *Ports and Adapters*, cuyo objetivo es separar la lógica de negocio del resto de elementos externos (interfaces, bases de datos, framework, API, etc.). Facilita la mantenibilidad, reemplazo de componentes y pruebas unitarias.

**Marcos de trabajo ágil:** Conjuntos de prácticas, roles y metodologías orientadas a gestionar proyectos de software mediante iteraciones cortas, retroalimentación continua y adaptación al

cambio. Ejemplos: Scrum, Kanban, XP.

**Patrones de diseño:** Paradigmas de código que brindan un proceso estructurado para resolver de manera reutilizable problemas recurrentes en el diseño y desarrollo de software.

**Jira:** Plataforma ágil utilizada para la planificación, documentación a nivel de usuario, creación de historias de usuario y administración del backlog dentro del proyecto.

**Scrum:** Marco de trabajo ágil basado en iteraciones cortas e incrementales enfocadas en la entrega continua de valor. Define roles específicos (Product Owner, Scrum Master y Equipo de Desarrollo), eventos estructurados (Sprint, Daily Scrum, Sprint Review y Sprint Retrospective) y artefactos formales (Product Backlog, Sprint Backlog e Incremento), permitiendo gestionar proyectos de manera adaptable, colaborativa y orientada a resultados.

**Sprint:** Unidad de trabajo dentro de un marco ágil, típicamente en Scrum, que consiste en ciclos de tiempo definidos donde se entregan avances incrementales del producto.

**Backlogs:** Listado priorizado de requerimientos, funcionalidades o tareas pendientes dentro de un proyecto ágil. Incluye tanto el *Product Backlog* como el *Sprint Backlog*.

**Spring Boot:** Framework de trabajo basado en Java que simplifica la creación y configuración de aplicaciones mediante un enfoque de inicialización rápida, ideal para el desarrollo de microservicios.

**Mural:** Espacio de trabajo perteneciente a un grupo de estudio, destinado a mantener un conjunto de publicaciones, ejercicios, preguntas, respuestas y discusiones relacionadas con el tema del grupo.

**BDP:** Acrónimo para Buscador de Parches. Funcionalidad propuesta para emparejar aleatoriamente estudiantes que buscan apoyo académico de manera inmediata.

**RF:** Acrónimo para Requerimiento Funcional. Representa acciones, comportamientos o capacidades específicas que debe cumplir el sistema.

**RNF:** Acrónimo para Requerimiento No Funcional. Describe atributos de calidad como rendimiento, seguridad, usabilidad, disponibilidad o mantenibilidad.

**API:** Acrónimo para Application Programming Interface. Es un conjunto de rutas y contratos de comunicación entre frontend y backend.

**REST:** Acrónimo para Representational State Transfer. Es un estilo arquitectónico de servicios HTTP usado en los endpoints del backend.

**DTO:** Acrónimo para Data Transfer Object. Son objetos en el codigo usados para transportar datos entre capas y en respuestas/peticiones HTTP.

**ORM:** Acrónimo para Object-Relational Mapping. Mapeo objeto-relacional entre clases Java y tablas SQL.

**SQL:** Acrónimo para Structured Query Languaje. Lenguaje de consulta usado sobre bases de datos relacionales.

**WebSocket:** Es un protocolo que funciona como canal bidireccional en tiempo real.

**STOMP:** Acrónimo para Simple/Streaming Text Oriented Messaging Protocol. Es un mensaje sobre WebSocket usado por cliente y servidor.

**SockJS:** Librería de compatibilidad/transporte para conexiones WebSocket.

**SSR:** Acrónimo para Server-Side Rendering. Renderizado inicial del HTML del frontend en el servidor.

**SPA:** Acrónimo para Single-Page Application. Aplicación de una sola página con fallback de rutas a index.html

**Docker:** Es una plataforma de software de código abierto que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones dentro de contenedores

**Bearer:** tipo de token de acceso utilizado en protocolos de autorización como HTTP.

### *1.1.2 Alcance del proyecto*

Con el fin de cumplir con lo estipulado en el cronograma del proyecto, se estimaron 16 semanas de trabajo, equivalentes a un semestre universitario. El alcance del desarrollo abarcó la implementación de las funcionalidades principales necesarias para la operación del sistema, incluyendo la creación de grupos de estudio, la gestión de reuniones, el manejo de recursos compartidos, el mural colaborativo y el chat temporal. También se desarrollaron módulos adicionales como el Botón del Pánico (BDP), el sistema de calificaciones, las notificaciones internas del servicio y la funcionalidad LFR.

Asimismo, se elaboró la documentación completa del ciclo de vida del proyecto, cubriendo el levantamiento de requerimientos, el diseño arquitectónico, la implementación, la validación mediante pruebas de caja negra y caja blanca y la preparación del entorno de despliegue usando docker. El microservicio fue desplegado en una máquina virtual que funciona como servidor único de servicios para COMA por temas de infraestructura, donde se verificó su funcionamiento básico. y su integración con la base de datos institucional utilizada por la plataforma COMA.

El proyecto no incluyó actividades relacionadas con la medición de adopción real por parte de usuarios finales, estrategias de divulgación o análisis del impacto en la interacción estudiantil. Tampoco se evaluó la compatibilidad nativa del sistema en dispositivos móviles Android o iOS, dado que el desarrollo se enfocó en una interfaz responsiva para entornos de escritorio utilizados por COMA. Del mismo modo, no se abordaron procesos de balanceo de carga ni pruebas de rendimiento a gran escala, sin embargo se realizaron las pruebas de rendimiento y funcionales en componentes muy especificos que lo precisaran, ya que estos aspectos exceden los objetivos definidos.

Finalmente, el microservicio fue diseñado para operar con las escuelas, programas y usuarios registrados en las bases de datos institucionales existentes en COMA para la sede Bucaramanga, sin extenderse a sedes regionales ni a otras plataformas académicas.

## 1.2 Objetivos

### *1.2.1 Objetivo general*

Desarrollar un microservicio en la plataforma COMA del grupo CALUMET, que permita a los estudiantes de la UIS crear, gestionar y participar en grupos de estudio interdisciplinarios, para facilitar encuentros informales de aprendizaje y servir como herramienta para la administración de recursos, la planificación de encuentros y el apoyo de la integración universitaria.

### *1.2.2 Objetivos específicos*

1. Documentar el ciclo de vida del proyecto, desde el análisis de requerimientos hasta su implementación en la plataforma COMA.

2. Diseñar la arquitectura del microservicio siguiendo principios de arquitectura hexagonal y buenas prácticas de diseño de software.

3. Implementar las funcionalidades que permitan la creación, gestión y participación en grupos de estudio, incluyendo recursos compartidos y herramientas de interacción.

4. Aplicar un marco de trabajo ágil como scrum mediante la plataforma de jira para la planificación, seguimiento y validación del desarrollo.

5. Validar la calidad y la funcionalidad del microservicio mediante la aplicación y documentación de pruebas unitarias de caja negra y caja blanca, complementadas con análisis de calidad y restricciones desarrollados a partir de una plantilla personalizada elaborada para este proyecto.

6. Desplegar el microservicio en un entorno independiente, asegurando su disponibilidad como servicio en la plataforma COMA.

# 2. Visión general del sistema

El sistema desarrollado consiste en una herramienta orientada a facilitar la creación, gestión y vinculación de estudiantes a grupos de estudio dentro del entorno académico universitario. El objetivo principal es apoyar a la comunidad estudiantil en la búsqueda de espacios colaborativos, remotos y presenciales, de aprendizaje; permitiendo que los usuarios puedan unirse, crear o administrar grupos de estudio interdisciplinarios.

Se basa en un microservicio desarrollado en Spring Boot, cuyas pruebas de despliegue se hicieron en un entorno independiente de desarrollo y su despliegue final se realiza en una máquina virtual institucional junto al servicio principal por temas de infraestructura. Este microservicio gestiona la comunicación entre la aplicación y la base de datos institucional, permitiendo la verificación de estudiantes inscritos en la sede de Bucaramanga que utilicen la plataforma COMA o que se encuentren dentro del sistema académico.

La plataforma incluye funcionalidades como la gestión de grupos de estudio; es decir, los usuarios podrán elegir entre unirse a un grupo existente o crear su propio grupo con todas las herramientas que este ofrece. Podrán gestionar el chat, las publicaciones del mural y el panel de recursos del grupo. Los usuarios que creen un grupo podrán establecer un horario para reuniones informales en la UIS, el cual será visible para todos los miembros del grupo. Además, el sistema cuenta con notificaciones y otras funcionalidades adicionales que permiten que un estudiante, incluso con poco tiempo, pueda encontrar el conocimiento que busca dentro de nuestro módulo.

## 2.1 Usuarios del sistema

Los usuarios principales del sistema se clasifican en las siguientes categorías:

- **Usuario solicitante:** Estudiante que desea integrarse a un grupo de estudio existente. Este usuario puede explorar los grupos disponibles, solicitar unirse, interactuar en el chat, ver y publicar en el mural y acceder a los recursos compartidos dentro del grupo.

- **Usuario líder de grupo:** Estudiante con iniciativa de crear, organizar y administrar un grupo de estudio. Tiene permisos para gestionar cupos, modificar descripciones, asignar roles internos y moderar la actividad del grupo.
- **Roles y estados internos adicionales de los miembros:**
  - **Moderador:** Usuario encargado de supervisar el comportamiento dentro del chat del grupo.
  - **Activo:** Usuario aceptado en un grupo, con acceso a los recursos y actividades internas.
  - **Desertor:** Miembro o líder que Abandonó el grupo.
  - **Vetado:** Miembro de un grupo que ha sido expulsado por el líder o un moderador.
  - **Rechazado:** Solicitud de ingreso rechazada por el líder.
  - **Interesado:** Solicitud de ingreso pendiente de aprobación.
  - **Silenciado:** No puede publicar ni enviar mensajes.

## 2.2 Restricciones generales

Las restricciones del sistema se agrupan en tres categorías principales: tecnológicas, normativas y regulatorias.

### *2.2.1 Tecnológicas*

- El microservicio debe ser desplegado en la máquina virtual institucional asignada para el proyecto.
- El sistema debe integrarse dentro de la base de datos institucional y ser parte de la plataforma COMA. Limitándose al uso de documentación oficial del grupo de estudio que valide las herramientas y la arquitectura.

- Se limita inicialmente la compatibilidad a dispositivos de escritorio y navegadores modernos; no se garantiza la adaptación para iOS, Android o dispositivos móviles de manera nativa, pero se establece la necesidad de ser responsivo con estos mismos.
- El microservicio debe operar bajo los recursos limitados asignados (CPU, RAM y almacenamiento), sin escalabilidad automática.
- El sistema debe comunicarse mediante APIs internas definidas previamente. En caso de no haber una API especifica para el microservicio se debe crear una nueva siguiendo las indicaciones del director.

### *2.2.2 Normativas*

- El proyecto debe alinearse con los lineamientos institucionales de calidad y gestión de procesos definidos por la Universidad Industrial de Santander, en el marco de su Sistema de Gestión de Calidad basado en la norma ISO 9001:2015 (Universidad Industrial de Santander, 2024).
- El desarrollo del sistema deberá garantizar el cumplimiento de buenas prácticas de ingeniería de software, incluyendo el manejo adecuado de excepciones, control de versiones y documentación técnica.
- El estudiante es responsable de la implementación de componentes que no correspondan a servicios institucionales públicos, así como del correcto funcionamiento del sistema desarrollado.
- El manejo de la información deberá cumplir con principios de confidencialidad, integridad y disponibilidad, especialmente en lo relacionado con datos sensibles de los usuarios.
- El acceso al sistema estará restringido a usuarios registrados como estudiantes activos de la sede Bucaramanga.

### *2.2.3 Regulatorias*

- El tratamiento de datos personales deberá cumplir con la Ley 1581 de 2012 (Congreso de la República de Colombia, 2012) y el Decreto 1377 de 2013 (Presidencia de la República de Colombia, 2013), garantizando los siguientes principios:
  - **Legalidad:** El tratamiento de datos debe realizarse conforme a la normativa vigente.
  - **Finalidad:** Los datos recolectados deberán utilizarse únicamente para los propósitos definidos por el sistema.
  - **Libertad:** El tratamiento de datos personales requiere la autorización previa, expresa e informada del usuario.
  - **Seguridad:** Se deben implementar medidas técnicas, humanas y administrativas que garanticen la protección de la información.
  - **Confidencialidad:** La información personal no podrá ser divulgada sin autorización, incluso después de finalizada la relación con el sistema.
- El sistema deberá solicitar autorización previa al usuario para el tratamiento de sus datos personales y permitir la consulta, actualización y eliminación de los mismos, conforme a la normativa vigente.
- El manejo de datos personales deberá limitarse estrictamente a la información necesaria para la operación del sistema, en cumplimiento del principio de minimización de datos.
- El sistema deberá implementar mecanismos de autenticación segura y control de acceso basado en roles, con el fin de prevenir el acceso no autorizado a la información.

## 2.3 Cronograma

A continuación, se muestra, tanto de forma textual como gráfica, la distribución temporal de las fases y actividades principales descritas en el plan inicial y que fueron tomadas en cuenta

para el desarrollo del proyecto.

Durante las semanas 1 y 2 se realizó la fase de planeación y análisis, en la cual se definió la visión general del sistema, el mundo del problema, el alcance y los requerimientos funcionales y no funcionales del sistema. En esta etapa también se estableció el plan general de trabajo y se documentaron los criterios técnicos que guiaron el desarrollo del proyecto. Finalmente, se realizó la traducción de requerimientos a historias de usuario en Jira.

En las semanas 3 y 4 se llevó a cabo la fase de diseño y modelado del sistema, enfocada en la elaboración de diagramas de caso de uso, el modelo entidad–relación (MER) y los mockups de interfaz.

Las semanas 5 a 10 estuvieron destinadas a la fase de desarrollo e implementación, que constituyó la etapa central del proyecto y, por ende, la más extensa. Durante este periodo se construyó el microservicio aplicando principios de arquitectura hexagonal, se seleccionaron las herramientas, se configuró el entorno de trabajo y se gestionó el control de versiones mediante Git, con el fin de completar todos los sprints en las semanas acordadas.

Posteriormente, durante las semanas 11 a 13, se desarrolló la fase de pruebas y ajustes, incluyendo pruebas de caja blanca, caja negra, funcionalidad y calidad.

Finalmente, en las semanas 14 a 16 se ejecutó la fase de despliegue y documentación final. En esta etapa se configuró el entorno de producción en docker para la máquina virtual y se realizó el despliegue del microservicio.

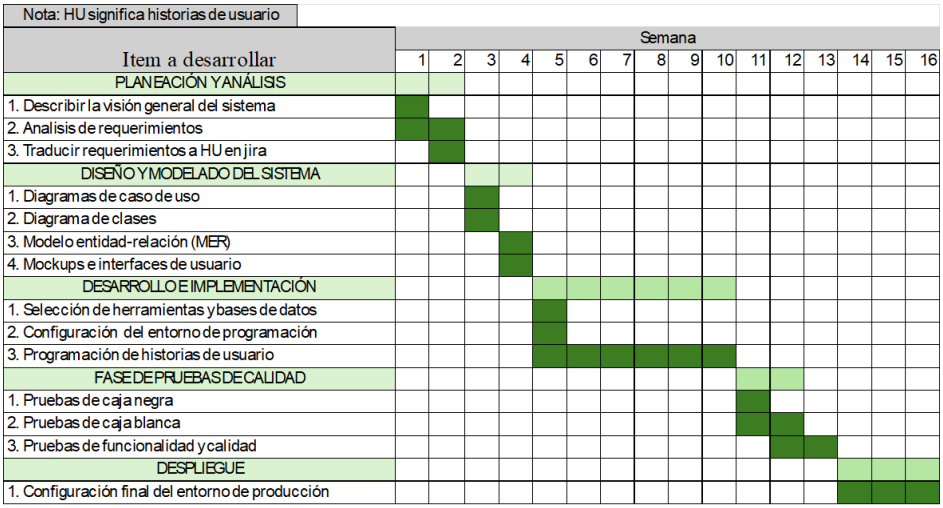
## 2.4 Presupuesto

De acuerdo con las directrices de la Unidad de Apoyo a la Gestión de Proyectos de la Vicerrectoría de Investigación y Extensión, el presupuesto del proyecto se determina considerando lo establecido en dicha unidad, tomando como base la modalidad de trabajo vigente y los recursos existentes

Los costos relacionados con los desarrolladores se obtuvieron de Computrabajo 2025, presenta proyecciones salariales fundamentadas en información de las ofertas para el puesto de Pro-

**Figura 1**

*Cronograma general del proyecto*




gramador/a web en Colombia a partir de fuentes obtenidas de las empresas, usuarios y empleados en los últimos doce meses (Computrabajo Colombia, s.f.)

**Tabla 1**

*Proyección del presupuesto*

| **Descripción** | **Responsable** | **Entidad** | **Unidad** | **C/U** | **Valor Unitario (COP)** | **Valor Total (COP)** |
|---|---|---|---|---|---|---|
| **Salarios** | Profesor planta: Luis Ignacio González | UIS | Hora | 64 | 305.000 | 19.520.000 |
| | Danna Valentina Prada Briceño | Autor | Hora | 320 | 14.000 | 4.480.000 |
| | Andrés Felipe Olivar Gutiérrez | Autor | Hora | 320 | 14.000 | 4.480.000 |
| **Gastos** | Internet | Autores | Mes | 4 | 100.000 | 400.000 |
| | Equipo de Cómputo | Autores | Unidad | 2 | 3.500.000 | 7.000.000 |
| | Transporte | Autores | Día | 80 | 10.600 | 848.000 |
| | Consumo eléctrico | Autores | Mes | 4 | 90.000 | 360.000 |
| **TOTAL** | | | | | | **37.088.000** |

Aunque no implicó una inversión económica, el presupuesto sirvió como una herramienta para visualizar la estructura y dar una aproximación de la demanda económica de un proyecto de

software en un entorno real.

# 3. Marco de referencia

## 3.1 Fundamentos teóricos

### *3.1.1 Integración universitaria y aprendizaje colectivo*

El aprendizaje grupal es una valiosa alternativa al aprendizaje tradicional, pues se entiende como un proceso en el que las habilidades y conocimientos se construyen de manera conjunta, y no como una única línea de información y método verídico entregado por el responsable. En este sentido, los procesos cognitivos y sociales del grupo permiten que cada miembro desarrolle competencias prácticas y comprenda de manera más significativa los contenidos (Carranza Gangotena et al., 2022; Pérez & Rodríguez, 2021).

Según Castellanos (2002), "la dinámica colaborativa transforma al grupo en una unidad que se sustenta a sí misma y promueve tanto el intercambio de ideas como la resolución conjunta de problemas".

### *3.1.2 Tecnologías educativas y plataformas digitales*

Las tecnologías educativas y las plataformas digitales constituyen un eje fundamental para expandir los procesos de enseñanza más allá del aula tradicional. Estas herramientas ofrecen espacios flexibles que permiten a los estudiantes experimentar la informalidad institucional, brindando escenarios para la interacción tanto síncrona como asíncrona, el acceso a recursos digitales y la colaboración entre pares ubicados en distintos lugares.

Según un estudio, las tecnologías digitales favorecen la personalización del aprendizaje, mejoran la participación estudiantil y contribuyen a reducir barreras de acceso (OECD, 2022). Asimismo, investigaciones recientes han demostrado que las plataformas digitales pueden generar un impacto positivo en el rendimiento académico y en el compromiso de los estudiantes (Frontiers in Psychology, 2022; Raj, 2024). En el contexto universitario, estas plataformas actúan como

verdaderas infraestructuras educativas, brindando nuevas formas de interacción, evaluación e intercambio de ideas (Zhao et al., 2023) .

### *3.1.3 Arquitecturas de software*

La arquitectura nos permite planificar a priori nuestro desarrollo y elegir el mejor conjunto de herramientas para llevar a cabo nuestros proyectos; es decir, constituye un elemento central en el desarrollo de sistemas. Una buena planeación comienza con la definición de una arquitectura sólida, ya que la escalabilidad, el rendimiento, la mantenibilidad y la calidad del software dependen de ella. Según Bass et al. (2012), "software architecture is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them".

Además, la arquitectura aborda cuestiones clave como el costo, el tiempo de desarrollo, el número de usuarios y el nivel de alcance del sistema. Como explica Kruchten (1995), "software architecture is a strategic design decision, where trade-offs must be made among performance, scalability, and cost, balancing both business and technical concerns".

Asimismo, la elección de una arquitectura específica debe responder a las necesidades y objetivos del sistema a desarrollar, garantizando que los requerimientos funcionales y no funcionales puedan satisfacerse de manera eficiente (Bass et al., 2012).

### *3.1.4 Microservicios y arquitectura hexagonal*

Los microservicios constituyen un estilo arquitectónico cuyo objetivo es dividir un sistema completo en varios servicios separados que pueden funcionar de manera independiente. A diferencia de los sistemas monolíticos, en los que si un servicio requiere más recursos es necesario escalar toda la infraestructura, los microservicios permiten escalar únicamente el componente necesario. En contraste con los sistemas monolíticos —como es el caso de la plataforma COMA, cuya estructura única dificulta la integración de nuevas funcionalidades sin afectar el sistema completo—, los microservicios permiten añadir o modificar componentes sin interrumpir el funcionamiento global.

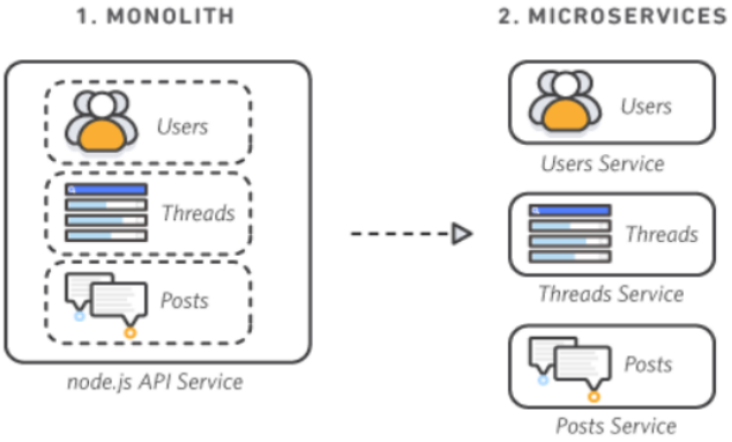
De esta manera, las plataformas educativas pueden soportar mejor el crecimiento de usuarios, la diversidad de necesidades y la incorporación continua de nuevas herramientas (Huet, 2022).

Para implementar microservicios de manera efectiva, es necesario considerar patrones arquitectónicos que refuercen el desacoplamiento y la mantenibilidad. La arquitectura hexagonal, también conocida como Ports and Adapters, fue propuesta por Cockburn (2005), quien la define como un enfoque que facilita la independencia entre el núcleo de la aplicación y sus mecanismos externos, permitiendo reemplazar o extender tecnologías sin comprometer la lógica central del sistema.

En conclusión, la combinación de microservicios con una arquitectura hexagonal aporta robustez y flexibilidad, garantizando que cada componente del sistema pueda evolucionar sin comprometer el todo.

**Figura 2**

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




*Tomado de Monolith to Microservices on AWS, por Amazon Web Services, 2020, https://aws.amazon.com/architecture/*

Además, la arquitectura hexagonal resulta especialmente adecuada en entornos basados en microservicios debido a que promueve una clara separación de responsabilidades entre la lógica de negocio y los mecanismos de comunicación externos. Gracias al uso de puertos y adaptadores,

cada microservicio puede interactuar con bases de datos, servicios externos, sistemas de mensajería o interfaces de usuario sin generar dependencias directas con estas tecnologías.

### *3.1.5 Marcos de trabajo ágiles en el desarrollo del software*

Los marcos de trabajo ágiles representan un cambio de paradigma frente a los modelos tradicionales de desarrollo en vez de hacer desarrollos individuales sin planificación, estos se basan en la entrega continua de valor y la colaboración entre los miembros del equipo. Su aplicación en proyectos de software permite responder mejor a las necesidades cambiantes de los usuarios y reducir riesgos asociados a la incertidumbre del entorno. Como señala Schwaber y Sutherland (2020), creadores de Scrum, “lo ágil permite enfrentar problemas complejos adaptativos, mientras se entrega de manera productiva y creativa productos de máximo valor”.

### *3.1.6 Ciclo de vida del software*

El ciclo de vida del software constituye un marco de referencia que organiza las etapas necesarias para el desarrollo de un sistema, desde la definición de requisitos hasta su mantenimiento. Su importancia radica en ofrecer un orden lógico de actividades que permite asegurar la calidad del producto y optimizar recursos. Según Sommerville (2011), “el ciclo de vida del software define el conjunto de fases y actividades involucradas en la producción de software, proporcionando una estructura para planear, administrar y controlar los proyectos de desarrollo”.

Hacer la documentación de cada una de estas fases permite dejar un rastro claro de las decisiones tomadas, estructurar todos los componentes antes y después del desarrollo, y mantener una buena hoja de ruta para los desarrolladores que facilite la continuidad del trabajo.

### *3.1.7 Buenas prácticas y clean code*

La calidad de un sistema no depende únicamente de sus funcionalidades, sino también de la forma en que el código es escrito y mantenido. Las buenas prácticas de programación permiten garantizar legibilidad, mantenibilidad y escalabilidad en los proyectos de software. Como señalan

Martin (2009) y Newman (2015), “clean code reads like well-written prose; it should be elegant and efficient, making it easy for other developers to understand and extend”.

En este sentido, la aplicación de principios como SOLID, la eliminación de duplicidad, la correcta organización de módulos y el uso de convenciones consistentes contribuyen a reducir la deuda técnica y facilitan la evolución del sistema. Para proyectos educativos de impacto institucional, como el presente, estas prácticas son fundamentales, pues aseguran la sostenibilidad y la posibilidad de integración con otros sistemas en el futuro.

### *3.1.8 Pruebas de software y evaluaciones de calidad*

Las pruebas de software constituyen una etapa fundamental en el ciclo de vida del desarrollo, ya que permiten verificar que el sistema cumpla con los requerimientos definidos y funcione bajo condiciones esperadas. Según Pressman y Maxim (2014), “the purpose of software testing is to reveal defects, not to prove that software works”.

Asimismo, la evaluación de calidad asegura que el producto final no solo sea funcional, sino que también cumpla criterios de confiabilidad, usabilidad y rendimiento, factores que determinan su sostenibilidad en contextos reales de uso (Springer, 2020).

### *3.1.9 Máquinas virtuales*

Las máquinas virtuales (VM) constituyen una herramienta esencial en el desarrollo, despliegue y mantenimiento de sistemas distribuidos, ya que permiten emular un entorno computacional completo sobre un sistema físico, sin alterar su configuración original. Su funcionamiento se basa en un hipervisor, encargado de gestionar los recursos de hardware y asignarlos de manera controlada a cada entorno virtual.

Gracias a esta tecnología, es posible aislar los procesos, reproducir entornos de prueba y garantizar la estabilidad del sistema, lo cual resulta fundamental durante las fases de integración y despliegue de aplicaciones. En el contexto de proyectos académicos y de investigación, las máquinas virtuales ofrecen una ventaja significativa: permiten crear entornos seguros, reproducibles y

configurables, evitando interferir con la infraestructura institucional o los servicios en producción.

Además, facilitan la configuración de servicios como bases de datos, servidores web o microservicios de backend en un entorno controlado, asegurando compatibilidad entre componentes y reduciendo riesgos de fallos por diferencias de entorno.

## 3.2 Antecedentes del tema

### *3.2.1 Sistemas de gestión de aprendizaje*

Los sistemas de gestión de aprendizaje (LMS) representan una de las primeras aplicaciones de arquitecturas de microservicios en plataformas educativas. Esta aproximación constituye una forma inicial de integración de microservicios en entornos de e-learning, con el fin de mejorar la interoperabilidad con plataformas como Moodle, pero manteniendo un enfoque principalmente en el control de recursos y la gestión académica desde la perspectiva del docente (Milovanović, 2021).

Sin embargo, aunque estos trabajos demuestran la viabilidad técnica de los microservicios en la educación digital, su énfasis sigue centrado en la figura del educador como actor principal que controla los recursos y procesos, sin incentivar de manera significativa la socialización ni la integración entre los estudiantes. En contraste, la propuesta de este proyecto busca descentralizar dicho control, fomentando un modelo colaborativo: un espacio donde cualquier estudiante con iniciativa pueda crear grupos de estudio y coordinar encuentros, sin la existencia de una autoridad central que dicte los procesos de enseñanza. Se trata, más bien, de un ambiente que se adapta a los propios miembros del grupo, a su medida, a su ritmo y a sus intereses. De esta forma, se mantienen las ventajas de los microservicios, pero la innovación de la presente propuesta radica en su énfasis en la igualdad, la autonomía estudiantil y el aprendizaje informal.

### *3.2.2 Tendencias actuales*

Las tendencias actuales evidencian un creciente interés por el e-learning, especialmente por los entornos colaborativos e informales donde los estudiantes pueden compartir experiencias, contrastar perspectivas y construir conocimiento de manera conjunta. De acuerdo con el estudio

Learning environments preferred by university students: a shift toward informal and flexible learning environments (Sommerville, 2011), existe una inclinación marcada hacia espacios virtuales que promuevan la flexibilidad, la autonomía y la interacción social como parte esencial del proceso de aprendizaje.

En este contexto, han surgido plataformas como StudyStream, Study Together y comunidades virtuales en Discord o Notion, que conectan a estudiantes de distintas partes del mundo para estudiar en simultáneo, compartir hábitos de estudio o colaborar en temas específicos. Estas plataformas, aunque no pertenecen al ámbito institucional, reflejan una tendencia global hacia la creación de espacios digitales de aprendizaje social, donde la motivación y la colaboración surgen de manera orgánica y desde la iniciativa u interés de los propios usuarios.

Asimismo, aplicaciones reconocidas como Duolingo han demostrado el impacto positivo de los entornos virtuales flexibles e interactivos en la retención y motivación del usuario. A través de estrategias como las rachas de estudio, los desafíos diarios o la gamificación del progreso, estas plataformas logran mantener el interés y la constancia en el aprendizaje. Sin embargo, a diferencia de estos modelos estructurados y guiados por algoritmos predefinidos, la propuesta de este proyecto busca ofrecer un espacio más libre y personalizado, donde los propios estudiantes definan sus dinámicas de aprendizaje. En este entorno, cualquier usuario que necesite apoyo en un tema podrá encontrar grupos, pares o comunidades dispuestas a colaborar, de modo que la variedad de conocimientos disponibles dependerá directamente de la diversidad y participación de la comunidad universitaria.

## 4. Planificación y análisis

Cumpliendo con el plan en el cual se remarca el ciclo de vida del software, se partió por la documentación de los requerimientos funcionales y no funcionales estipulada en la primera fase del ciclo de vida del software: planificación y análisis.

En esta fase se documentaron todos los requerimientos funcionales y no funcionales que componen el microservicio.

Los requerimientos planteados en este documento se presentan de forma técnica y siguiendo el estándar ISO/IEC/IEEE 29148, debido a que constituyen una guía para los desarrolladores participantes y representan un vistazo más profundo a la funcionalidad que se debe implementar. Sin embargo, la planificación meramente técnica puede llegar a ser tediosa para los desarrolladores, y es por ello que estos requerimientos posteriormente tendrán su lugar como historias de usuario, las cuales son representaciones más pequeñas que componen un solo requerimiento y proporcionan a los desarrolladores un enfoque más claro y focalizado del trabajo.

De acuerdo con Sommerville (2011), la descomposición de requerimientos en unidades más pequeñas y comprensibles facilita la trazabilidad, la verificación y la gestión del proceso de desarrollo, especialmente cuando se trabaja con marcos de trabajo ágiles.

## 4.1 Análisis de requerimientos

### *4.1.1 Estándar ISO/IEC/IEEE 29148 — Ingeniería de Requerimientos*

Nos basamos en el estándar «**ISO/IEC/IEEE 29148:2018 Systems and software engineering – Life cycle processes – Requirements engineering**» **(2018)**, el cual establece las prácticas formales para la documentación, gestión y validación de requisitos de software y sistemas. Este estándar sustituye al IEEE 830 y define criterios claros para garantizar que los requisitos sean comprensibles, verificables y trazables a lo largo de todo el ciclo de vida del sistema (Wiegers & Beatty, 2013).

### *4.1.2 Principios del estándar*

El estándar establece que cada requisito debe cumplir con un conjunto de características esenciales. A continuación se presenta una tabla representatitva de los Criterios de calidad para requerimientos (ISO/IEC/IEEE 29148).

**Tabla 2**
*Criterios de calidad de requisitos según ISO/IEC/IEEE 29148*

| **Criterio** | **Descripción** |
|---|---|
| Correcto | El requisito corresponde a una necesidad real y no contiene errores. |
| No ambiguo | Puede interpretarse de una sola manera, sin confusiones. |
| Completo | Incluye toda la información necesaria: condiciones, restricciones y contexto. |
| Consistente | No entra en conflicto con otros requisitos del sistema. |
| Verificable | Puede comprobarse mediante pruebas, inspecciones o demostraciones. |
| Viable | Es técnica y económicamente posible de implementar. |
| Trazable | Se puede vincular con su origen (casos de uso, necesidades del usuario) y con su implementación (código, pruebas). |
| Necesario | Aporta valor al sistema y no es redundante. |
| Único | Describe una sola funcionalidad o regla, evitando mezclar múltiples aspectos en un mismo requisito. |

En conclusión, la estructura mencionada en el plan para la documentación de los requerimientos se mantiene; sin embargo, consideramos pertinente añadir un elemento más: los actores que participan en cada requerimiento. Esta estructura solo aplica a los requerimientos funcionales, debido a que los requerimientos no funcionales requieren descripciones más variables dependiendo de su naturaleza.

La estructura final para los requerimientos funcionales es la siguiente:

1. **Descripción:** Explica de manera técnica y completa la funcionalidad que el sistema debe ofrecer.

2. **Entradas:** Datos o información que el requerimiento recibe para su ejecución.

3. **Salidas:** Información generada como resultado del funcionamiento.

4. **Criterios de aceptación:** Condiciones que deben cumplirse para considerar el requerimiento correctamente implementado.

5. **Actores:** Usuarios o componentes que interactúan con la funcionalidad descrita en el requerimiento.

"Los requerimientos funcionales y no funcionales proporcionan una descripción completa de lo que debe hacer un sistema y las restricciones bajo las cuales debe operar." (Sommerville, 2011)

### *4.1.3 Requerimientos funcionales*

**MOGE_RF001: Verificar inicio de sesión del estudiante**

- **Descripción:** El sistema debe verificar que el token con los datos encriptados del usuario sea valido y hagan referencia a un usuario activo en la base de datos.
- **Entrada:** El usuario selecciona el botón de acceso al módulo de grupos de estudio desde el panel principal.
- **Salida:**
  - Panel informativo del error de autenticación si el usuario no tiene una sesión activa.
  - Redirección al módulo de grupos de estudio si el usuario ya tiene una sesión activa.
- **Criterios de aceptación:**
  - El sistema debe detectar correctamente si el usuario tiene o no una sesión activa en COMA.
  - Si el usuario no tiene sesión activa, debe mostrar el panel informativo del error de autenticación.

    - Si el usuario ya tiene sesión activa, debe acceder al módulo de grupos de estudio sin pasos adicionales.
    - El token es guardado en session storage y usado con bearer para las funciones dentro del servicio.

- **Actores:** Estudiante autenticado o no autenticado en COMA.

**MOGE_RF002: Crear grupo de estudio**

- **Descripción:** El sistema debe permitir la creación de un grupo de estudio; verificando que la información ingresada en el formulario cumpla con las normas predefinidas, tales como campos obligatorios y la longitud máxima permitida, además debe seleccionar la escuela y programa académico del usuario que crea el grupo automáticamente, garantizando la correcta ejecución de la operación y la creación de sus componentes asociados (Mural, panel de recursos y chat grupal). En caso contrario, el sistema debe impedir la creación del grupo e informar el motivo al usuario.
- **Entrada:** Campos obligatorios requeridos para la creación del grupo de estudio.
    - Nombre del grupo,
    - Descripción del grupo,
    - Modalidad (público o privado),
    - Asignatura,
- **Salida:**
    - Creación exitosa del grupo de estudio y su entorno asociado.
    - Recarga directa del panel con el nuevo grupo añadido.
    - Confirmación del resultado de la operación.
- **Criterios de Aceptación:**

  - El sistema debe validar que los datos ingresados cumplan con las reglas definidas, tales como campos obligatorios y longitud permitida.
  - El sistema debe notificar el resultado de la operación, ya sea exitoso o fallido.
  - Deben inicializarse correctamente el mural, el panel de recursos y el chat asociados.
  - El sistema debe mostrar mensajes claros ante éxito o fallo en la operación.

- **Actores:** Estudiante aspirante a líder de grupo.

**MOGE_RF003: Programar horario del grupo de estudio**

- **Descripción:** El sistema debe permitir que el líder de un grupo de estudio registre los horarios de reunión del grupo. Los horarios podrán definirse durante la creación del grupo o en la posterior edición de la información del mismo.

  El sistema debe ofrecer una interfaz de selección de días de la semana y franjas horarias personalizables. El líder podrá seleccionar uno o varios horarios dentro del grupo.

  Los horarios registrados deberán mostrarse en el panel de información del grupo y tambien dentro del mural del grupo y serán visibles para todos los miembros.

- **Entrada:** Selección realizada por el líder del grupo en el formulario de programación de horarios, incluyendo:

  - Día de la semana.
  - Hora de inicio de la reunión hora fin de la reunión.

- **Salida:**

  - Registro exitoso de los horarios del grupo.
  - Visualización de los horarios en el panel de información del grupo.

- **Criterios de aceptación:**

  - Solo el líder del grupo puede crear o modificar los horarios del grupo.
  - No se debe permitir registrar horarios en formato inválido o incompleto.
  - Los horarios configurados deben mostrarse correctamente en el panel de información del grupo.

- **Actores:** Líder del grupo de estudio.

**MOGE_RF004: Editar atributos del grupo de estudio**

- **Descripción:** El sistema debe permitir al líder del grupo de estudio actualizar los valores registrados anteriormente en el grupo de estudio (nombre, descripción, materia asociada e imagen de portada), mediante el ícono de configuración ubicado en el panel principal del grupo que abrirá el formulario de edición del grupo.
- **Entrada:** Datos editados por el líder del grupo, tales como nombre del grupo, descripción, imagen del grupo y su materia asociada.
- **Salida:** Confirmación de actualización exitosa y visualización del grupo de estudio con los nuevos atributos modificados con la persistencia en base de datos.
- **Criterios de Aceptación:**
  - Se debe validar que el usuario que realiza la edición sea el líder del grupo.
  - El sistema debe mostrar los datos actuales para permitir su modificación.
  - Si ocurre un error, se debe mostrar un mensaje indicando que no fue posible guardar los cambios.
  - La edición de los horarios se gestiona en el requerimiento MOGE_RF008.
- **Actores:** Líder del grupo de estudio

**MOGE_RF005: Solicitar unirse a grupo de estudio**

- **Descripción:** El sistema debe permitir a los usuarios enviar una solicitud de inscripción a los líderes de un grupo de estudio privado con el fin de unirse a dicho grupo. Esto se realiza desde el panel de búsqueda de grupos de estudio, donde, al seleccionar un grupo de interés, se mostrará la opción .Enviar solicitud de inscripción"si el grupo es privado o Ünirse"si el grupo es público.

  En el caso de los grupos privados, el estudiante no podrá participar ni interactuar hasta que su solicitud sea aceptada, mientras tanto su estado interno como miembro se mantiene en "Interesado". En los grupos públicos, el usuario deberá presionar "Unirse" para integrarse al grupo y poder interactuar con sus funcionalidades.

- **Entrada:**

  - Selección del grupo de estudio por parte del usuario.
  - Acción del usuario: "Enviar solicitud de inscripción" o "Unirse".

- **Salida:**

  - Mensaje de confirmación de solicitud enviada o unión exitosa.
  - Notificación enviada al líder del grupo para solicitudes privadas.
  - Cambio de estado del usuario: "pendiente", "Activo" o "rechazado".

- **Criterios de Aceptación:**

  - El sistema debe validar que el usuario no pertenezca previamente al grupo.
  - Si el grupo es privado, la solicitud debe quedar en estado pendiente hasta la aprobación del líder y se crea un nuevo miembro con estado "Interesado".
  - Si el grupo es público, el usuario debe unirse automáticamente sin aprobación previa.
  - El sistema debe impedir enviar múltiples solicitudes simultáneas al mismo grupo.

- El sistema debe notificar al usuario cuando su solicitud sea aceptada o rechazada.

- **Actores:**
  - Usuario solicitante
  - Líder del grupo de estudio

**MOGE_RF006: Gestionar solicitudes del grupo de estudio**

- **Descripción:** El sistema debe permitir al líder del grupo de estudio visualizar, aceptar o rechazar las solicitudes de inscripción enviadas por los estudiantes interesados en unirse al grupo.

  Cada acción de aceptación o rechazo deberá generar una notificación automática dirigida al estudiante solicitante, informándole la decisión del líder.

- **Entrada:**
  - Lista de solicitudes pendientes asociadas al grupo.
  - Selección de una solicitud específica.
  - Acción del líder: aceptar o rechazar.

- **Salida:**
  - Cambio del estado de la solicitud (aceptada o rechazada).
  - Notificación enviada al estudiante solicitante.
  - En caso de aceptación, cambio del estado del miembro de ”Interesado.ª .ªctivo”.

- **Criterios de Aceptación:**
  - Solo el líder del grupo puede gestionar solicitudes.
  - No se deben mostrar solicitudes de usuarios que ya pertenezcan al grupo.
  - Cada solicitud debe quedar marcada con su estado (pendiente, aceptada o rechazada).

  - El sistema debe registrar fecha y hora de la decisión tomada.
  - El solicitante debe recibir una notificación del resultado.

- **Actores:**
  - Líder del grupo de estudio
  - Estudiante solicitante

**MOGE_RF007: Editar horarios del grupo de estudio**

- **Descripción:**

  El sistema debe permitir que el líder del grupo de estudio modifique los horarios registrados del grupo. El líder podrá cambiar el día de la semana y la franja horaria seleccionada.

  El sistema debe validar que el nuevo horario cumpla con las restricciones definidas y actualizar la información mostrada en el panel del grupo.

- **Entrada:** Datos editados por el líder:
  - nuevo día de reunión,
  - nueva hora de inicio,
  - selección del horario a modificar.

- **Salida:**
  - confirmación de edición exitosa del horario,
  - actualización del panel de información del grupo,

- **Criterios de aceptación:**
  - Solo el líder del grupo puede editar los horarios.
  - Los cambios deben reflejarse inmediatamente en el panel del grupo.

- **Actores:** Líder del grupo de estudio.

**MOGE_RF008: Gestionar notificaciones**

- **Descripción:** El sistema debe permitir a los usuarios mantenerse informados mediante un sistema de notificaciones asociado a las acciones relevantes del servicio. Las notificaciones se clasifican en dos tipos principales:
  - **Notificaciones generales:** se visualizan en el panel "Mis notificaciones". Corresponden a eventos como:
    - Respuestas a solicitudes de ingreso a grupos de estudio.
    - Comentarios, reacciones o publicaciones en murales.
    - Solicitudes de encuentros personalizados.
  - **Notificaciones de eventos:** se muestran en las secciones específicas del sistema donde ocurre el evento. Incluyen, entre otros:
    - activación del Botón del Pánico,
    - nuevas publicaciones en el mural,
    - Mensajes rápidos en el buscador de parches.
- **Entrada:**
  - Evento generado por algún módulo del sistema.
  - Identificación del usuario destinatario de la notificación.
- **Salida:**
  - Registro de la notificación en el sistema.
  - Asociación de la notificación al usuario correspondiente.
  - Actualización del estado de la notificación (no leída, leída).
- **Criterios de Aceptación:**

- El sistema debe generar automáticamente notificaciones a partir de eventos definidos.
- Las notificaciones deben asociarse correctamente al usuario destinatario.
- El usuario debe poder visualizar sus notificaciones.
- El usuario debe poder marcar notificaciones como leídas.
- El sistema debe diferenciar entre notificaciones generales y contextuales.
- Las notificaciones deben almacenarse con su fecha y hora de generación.

- **Actores:**
  - Estudiante usuario del sistema
  - Líder de grupo de estudio

**MOGE_RF009: Salir de un grupo de estudio**

- **Descripción:** El sistema debe permitir que cualquier miembro de un grupo de estudio abandone voluntariamente dicho grupo. Al hacerlo, su estado dentro del grupo cambiará a "DESERTOR", y perderá acceso a las funcionalidades asociadas al mismo.

  Cuando un miembro abandone el grupo:
  - Su estado como miembro deberá actualizarse a "Desertor".
  - El grupo dejará de ser visible para el usuario en su lista de grupos.

- **Entrada:**
  - Identificador del usuario autenticado.
  - Identificador del grupo de estudio.
  - Acción explícita del usuario: "Salir del grupo".

- **Salida:**
  - Confirmación de salida exitosa del grupo.

- **Criterios de Aceptación:**
  - El sistema debe permitir que un miembro abandone el grupo en cualquier momento.
  - Una vez confirmada la salida:
    - el usuario no debe tener acceso al contenido interno del grupo,
    - el usuario no debe recibir notificaciones relacionadas con el grupo,
    - el sistema debe persistir el nuevo estado del miembro.
  - En caso de que el usuario sea el líder del grupo, el sistema deberá definir un mecanismo para la gestión del liderazgo antes de permitir la salida.
- **Actores:**
  - Miembro del grupo de estudio.
  - Líder del grupo de estudio.

**MOGE_RF010: Expulsar miembro del grupo de estudio**

- **Descripción:** El sistema debe permitir al líder de un grupo de estudio expulsar a cualquier miembro del grupo.

  La expulsión podrá realizarse desde:
  - La lista de miembros del grupo en el mural.
  - Un mensaje del usuario dentro del chat del grupo vía clic derecho en el avatar.

  Al expulsar a un miembro:
  - Su estado como miembro pasa a ”Vetado”.

  En el caso de los grupos de estudio públicos, el miembro expulsado no podrá volver a unirse al grupo ni visualizar su contenido.
- **Entrada:**

- Identificador del líder autenticado.
- Identificador del grupo de estudio.
- Identificador del miembro a expulsar.
- Confirmación explícita de la acción de expulsión.

- **Salida:**
  - Confirmación de expulsión exitosa.
  - Restricción de acceso al grupo para el usuario expulsado.
- **Criterios de Aceptación:**
  - El usuario que ejecuta la acción debe:
    - Ser líder del grupo.
  - Después de la expulsión:
    - El usuario no debe poder acceder al grupo.
    - El usuario no debe recibir nuevas notificaciones del grupo.
    - El estado como miembro pasa a ”Vetado”
- **Actores:**
  - Líder del grupo de estudio.
  - Miembro expulsado.

**MOGE_RF011: Buscar grupo de estudio**

- **Descripción:** El sistema debe permitir a los usuarios buscar grupos de estudio existentes. Inicialmente se mostrarán todos los grupos públicos disponibles y, posteriormente, el usuario podrá aplicar filtros por asignatura, privacidad o nombre. El sistema debe ordenar los resultados priorizando grupos mejor calificados, luego grupos recientes y, finalmente, grupos antiguos.

- **Entrada:**
  - Opción de búsqueda del sistema.
  - Filtros seleccionados (asignatura, modalidad, búsqueda por nombre).
- **Salida:**
  - Listado de grupos de estudio coincidentes.
  - Información resumida del grupo (nombre, líder, calificación, escuela, programa académico, materia, horarios (si aplica) y modalidad).
  - Mensaje informativo cuando no existan resultados.
- **Criterios de Aceptación:**
  - El sistema debe permitir la búsqueda aun si no se aplican filtros.
  - Los filtros deben poder combinarse entre sí.
  - El sistema debe mostrar como mínimo:
    - nombre del grupo, líder, escuela o programa, asignatura, horarios (si aplica) y modalidad (público/privado).
  - Si no existen resultados, se muestra el mensaje: *"No se encontraron grupos con los criterios seleccionados."*
- **Actores:**
  - Usuario solicitante.

**MOGE_RF012: Crear automáticamente los perfiles de usuarios**

- **Descripción:** El sistema debe crear automáticamente perfiles para todos los usuarios que pertenecen a la plataforma en el momento en que ingresan al módulo de grupos de estudio. Cada perfil incluirá una descripción adicional editable por el estudiante y sus respectivas

calificaciones como miembro y como líder de grupo. El sistema permitirá cambiar la foto del perfil, pero no será posible modificar el nombre completo ni el programa académico, ya que estos datos provienen del sistema académico institucional.

- **Entrada:**
  - Ingreso del usuario al módulo de grupos de estudio por primera vez.
  - Datos provenientes de la base de datos COMA: nombre completo, identificación y programa académico.
- **Salida:**
  - Perfil de usuario creado en la base de datos.
- **Criterios de Aceptación:**
  - Al ingresar por primera vez al módulo, el sistema crea automáticamente el perfil del usuario.
  - El usuario puede editar únicamente la fotografía y la descripción personal.
  - El usuario no puede modificar su nombre completo ni su programa académico.
  - Si el perfil ya existe, el sistema no lo crea nuevamente.
  - El perfil queda almacenado correctamente en la base de datos.
- **Actores:**
  - Usuario

**MOGE_RF013: Gestionar perfil de usuario**

- **Descripción:** El sistema debe permitir al usuario consultar y modificar la información editable de su perfil dentro del módulo de grupos de estudio. El usuario podrá actualizar su fotografía de perfil y modificar su descripción personal. No podrá modificar su nombre completo ni su programa académico, ya que estos datos provienen del sistema académico institucional.

- **Entrada:**
  - Solicitud del usuario para editar su perfil.
  - Solicitud del usuario para editar fotografía, descripción personal o fondo de perfil.
- **Salida:**
  - Perfil actualizado.
  - Mensajes de confirmación o error.
- **Criterios de Aceptación:**
  - El usuario puede visualizar su información básica de perfil.
  - El usuario puede editar únicamente su descripción personal, fotografía de perfil y fondo de perfil.
  - El sistema no permite modificar el nombre completo ni el programa académico.
  - Los cambios realizados se almacenan correctamente en la base de datos.
  - El sistema muestra mensajes de confirmación o error.
- **Actores:**
  - Usuario.

**MOGE_RF014: Asignar roles en el grupo de estudio**

- **Descripción:** El sistema debe permitir que el líder del grupo de estudio asigne y retire roles especiales a los miembros del grupo, tales como moderador del chat grupal, con el fin de apoyar la gestión del grupo y el control de los espacios de interacción. El panel de asignación de roles se da desde el chat del grupo de estudio, y se desplegará mediante un click derecho en el avatar de algún miembro.
- **Entrada:**

    - Selección de un miembro del grupo por parte del líder.
    - Selección del rol a asignar o retirar.

- **Salida:**

    - Confirmación de asignación o retiro del rol.
    - Actualización del listado de roles del grupo.

- **Criterios de Aceptación:**

    - Solo el líder del grupo puede asignar o retirar roles.
    - El sistema debe permitir asignar como mínimo el rol de moderador del chat grupal.
    - El sistema debe validar que el usuario pertenece al grupo antes de asignarle un rol.
    - Los cambios realizados se reflejan de forma inmediata en el grupo.
    - El sistema muestra mensajes claros de confirmación o error.

- **Actores:**

    - Líder del grupo
    - Miembro del grupo

**MOGE_RF015: Gestionar chat grupal**

- **Descripción:** El sistema debe permitir que los miembros del grupo de estudio envíen y reciban mensajes y archivos en tiempo real a través del chat grupal. Cada mensaje debe mostrar el nombre y el avatar del remitente. El chat contará con un fondo configurable y un panel de información del grupo visible al hacer clic en el icono de información del chat.

- **Entrada:**

    - Mensajes de texto ingresados por los usuarios.
    - Archivos adjuntos enviados por los usuarios.

    - Solicitud de visualización del chat.

- **Salida:**

    - Mensajes y archivos entregados en tiempo real.
    - Historial de conversación almacenado.

- **Criterios de Aceptación:**

    - Solo los miembros del grupo pueden participar en el chat.
    - Los mensajes se envían y reciben en tiempo real.
    - Cada mensaje muestra el nombre y avatar del remitente.
    - El sistema permite enviar archivos adjuntos según las políticas de tamaño permitido.
    - El usuario puede visualizar el historial de conversación.
    - El chat muestra la información del grupo al seleccionar su nombre.
    - Los mensajes se persisten en base de datos.
    - Los archivos se persisten en el servidor de imágenes.

- **Actores:**

    - Miembros del grupo de estudio

**MOGE_RF016: Moderar chat grupal**

- **Descripción:** El sistema debe permitir que los usuarios con rol de moderador o líder del grupo gestionen el chat grupal, pudiendo editar el nombre del chat, la descripción del grupo y permitir el cambio del estado de miembros en el chat como silenciados o vetados.

- **Entrada:**

    - Solicitud para editar nombre, descripción o fondo del chat.

- Solicitud para silenciar o vetar un miembro del grupo.

- **Salida:**
  - Nombre, descripción o fondo del chat actualizados.
  - Registro del evento de moderación.

- **Criterios de Aceptación:**
  - Solo el líder o moderadores pueden moderar el chat.
  - El sistema permite editar el nombre y descripción del chat grupal.
  - Los cambios se reflejan de manera inmediata para todos los miembros.
  - Se evita que los usuarios sin rol autorizado realicen acciones de moderación.

- **Actores:**
  - Líder del grupo
  - Moderador
  - Miembros del grupo de estudio

**MOGE_RF017: Gestionar calificaciones de usuarios y grupos**

- **Descripción:** El sistema debe permitir gestionar calificaciones entre usuarios y grupos de estudio. Los usuarios podrán calificar:
  - A un grupo de estudio,
  - Al líder de un grupo de estudio,
  - A otros miembros del grupo de estudio.

  Las calificaciones no se realizan de forma libre, sino seleccionando etiquetas predefinidas tales como: *"estudiante ejemplar"*, *"líder vago"*, *"buen grupo"*, entre otras. Estas califica-

ciones serán almacenadas por el sistema y, a partir de un promedio ponderado de las mismas, se asignará un título visible en el perfil del usuario o en la información del grupo de estudio.

Solo podrán calificarse entre sí:

- Miembros del mismo grupo de estudio.

- **Entrada:**
  - Selección del usuario o grupo a calificar.
  - Selección de etiqueta de calificación predefinida.
  - Confirmación de envío de calificación.
- **Salida:**
  - Registro de la calificación en el sistema.
  - Recalculo del promedio de calificaciones del usuario o grupo.
  - Actualización del título visible en el perfil o información del grupo.
- **Criterios de Aceptación:**
  - Solo los usuarios que compartan grupo en común pueden emitir calificaciones.
  - Las calificaciones solo pueden seleccionarse desde una lista predefinida de etiquetas.
  - Un usuario no puede calificar dos veces al mismo usuario o grupo por el mismo evento.
  - El sistema debe recalcular automáticamente el promedio de calificaciones.
  - El sistema debe actualizar el título visible del usuario o grupo cuando cambie su promedio.
- **Actores:**
  - Miembros de grupos de estudio
  - Líderes de grupos de estudio

**MOGE_RF018: Asignar reseñas a miembros y grupos**

- **Descripción:** El sistema debe permitir a los miembros de un grupo de estudio emitir reseñas dirigidas a otros miembros del mismo grupo o al grupo en general.

  Cada reseña deberá contener un contenido textual con una longitud máxima de 500 caracteres y deberá estar asociada a un único destinatario (usuario o grupo).

- **Entrada:**

  - Identificador del usuario emisor.
  - Identificador del grupo de estudio.
  - Identificador del usuario destinatario (opcional).
  - Indicador del tipo de reseña (usuario o grupo).
  - Contenido de la reseña.

- **Salida:**

  - Registro de la reseña asociada al destinatario correspondiente.

- **Criterios de Aceptación:**

  - Solo los miembros que pertenezcan al mismo grupo de estudio pueden emitir reseñas entre sí.
  - El sistema debe validar que el destinatario pertenezca al grupo.
  - Cada usuario solo puede emitir una reseña hacia un mismo destinatario dentro de un grupo.
  - El contenido de la reseña no debe exceder los 500 caracteres.
  - La reseña debe quedar correctamente asociada al emisor, al destinatario y al grupo de estudio.

- **Actores:**
  - Miembro del grupo de estudio.

**MOGE_RF019: Gestionar mural del grupo de estudio**

- **Descripción:** El sistema debe crear automáticamente un mural asociado a cada grupo de estudio en el momento de su creación. El mural permitirá a los miembros del grupo publicar material relevante para el tema del grupo, así como consultar las publicaciones existentes mediante su interfaz. El mural funcionará como un almacén estructurado de publicaciones, recursos, información grupal, herramientas del grupo las cuales estarán en un panel de navegación dentro del mural.
- **Entrada:**
  - Solicitud para actualizar fondo o nombre del mural
- **Salida:**
  - Nuevo nombre actualizado o fondo del mural.
- **Criterios de Aceptación:**
  - Cada grupo de estudio debe tener un mural propio e independiente.
  - Solo los miembros del grupo pueden crear publicaciones en su mural.
  - El líder o moderadores del grupo pueden eliminar publicaciones que incumplan las normas.
  - las publicaciones se muestran según: destacadas, mas recientes, mas reaccionadas.
- **Actores:**
  - Miembros del grupo de estudio
  - Líder del grupo de estudio

- Moderadores del grupo de estudio

**MOGE_RF020: Gestionar publicaciones del mural**

- **Descripción:** El sistema debe permitir a los miembros de cada grupo de estudio crear, visualizar, comentar y reaccionar a publicaciones dentro del mural del grupo. Cada publicación debe contener un título y un cuerpo de texto, y podrá incluir elementos como imágenes, enlaces, color de fuente, fondo de la publicación y estilos de fuente. El sistema debe mostrar las publicaciones en un listado en forma de tarjetas donde se observe el autor, el título, la fecha de publicación y un resumen del contenido. Al seleccionar una tarjeta, el sistema debe mostrar la publicación completa, junto con sus comentarios y reacciones.

  Estas publicaciones podrán ser editables y eliminables según el autor lo decida además también tendrán la opción de guardar un borrador.

- **Entrada:**

  - Título de la publicación.
  - Contenido de la publicación (hasta 10.000 caracteres).
  - Archivos de imagen opcionales.
  - Enlaces opcionales.
  - Color de fuente.
  - Estilos de fuente.

- **Salida:**

  - Confirmación de creación, edición o eliminación de publicaciones.
  - Lista actualizada de publicaciones del mural.
  - Visualización detallada de una publicación seleccionada.
  - Visualización en tiempo real de la publicación creada.

  - Visualización de los borradores del miembro.

- **Criterios de Aceptación:**

  - Solo los miembros del grupo pueden crear publicaciones en su mural.
  - Cada publicación debe almacenarse asociada al grupo y al autor.
  - El título es obligatorio y el contenido no debe superar los 10.000 caracteres.
  - Los miembros del grupo pueden reaccionar y comentar las publicaciones.
  - El líder o moderadores del grupo pueden eliminar publicaciones.
  - Eliminar las publicaciones las deja en un SOFT DELETE, cambiando unicamente la visibilidad en el mural para esa publicación.

- **Actores:**

  - Miembros del grupo de estudio.
  - Líder del grupo de estudio.

**MOGE_RF021: Gestionar comentarios de las publicaciones**

- **Descripción:** El sistema debe permitir que los miembros de un grupo de estudio realicen comentarios sobre las publicaciones del mural del grupo.

  Cada comentario deberá estar asociado a un autor y a una publicación específica. Asimismo, el sistema deberá permitir la creación de respuestas a comentarios, formando estructuras jerárquicas de discusión.

  Los comentarios podrán ser editados o eliminados por su autor.

- **Entrada:**

  - Contenido del comentario.
  - Identificador de la publicación.

  - Identificador del usuario.
  - (Opcional) Identificador del comentario padre.

- **Salida:**
  - Confirmación de creación, edición o eliminación del comentario.
  - Lista actualizada de comentarios asociados a la publicación.

- **Criterios de Aceptación:**
  - Solo los miembros del grupo pueden comentar publicaciones del grupo.
  - Cada comentario debe estar asociado a una única publicación y a su autor.
  - El sistema debe permitir responder a comentarios existentes.
  - Los usuarios deben poder editar o eliminar sus propios comentarios.
  - El sistema debe mantener la relación jerárquica entre comentarios y sus respuestas.

- **Actores:**
  - Usuario miembro del grupo de estudio.

**MOGE_RF022: Reaccionar a publicaciones y comentarios**

- **Descripción:** El sistema debe permitir que un miembro del grupo reaccione a una publicación o a un comentario.

  La reacción del usuario deberá registrarse en el sistema y reflejarse en el conteo total de reacciones asociado al elemento. Asimismo, el usuario podrá retirar su reacción, actualizando el conteo correspondiente.

- **Entrada:**
  - Identificador de la publicación o comentario.
  - Identificador del usuario autenticado.

- Acción de agregar o retirar reacción.

- **Salida:**
  - Registro o eliminación de la reacción del usuario.
  - Actualización del conteo de reacciones asociado al elemento.
- **Criterios de Aceptación:**
  - Un usuario solo puede registrar una reacción por publicación o comentario.
  - El sistema debe permitir retirar la reacción previamente registrada.
  - El conteo de reacciones debe reflejar el número total de reacciones activas.
  - El sistema debe mantener la consistencia de los datos ante múltiples interacciones concurrentes.
- **Actores:**
  - Miembro del grupo de estudio.

**MOGE_RF023: Gestionar recursos compartidos**

- **Descripción:** El sistema debe proporcionar un módulo de recursos asociado a cada grupo de estudio, permitiendo a sus miembros registrar, consultar, actualizar y eliminar recursos relacionados con el tema del grupo.

  Cada recurso deberá incluir información como título, descripción, autor y contenido asociado (archivo o enlace).

- **Entrada:**
  - Identificador del grupo de estudio.
  - Datos del recurso (título, descripción, archivo opcional, enlaces).
  - Identificador del usuario que registra o edita el recurso.

- **Salida:**
  - Confirmación de creación, actualización o eliminación del recurso.
  - Listado actualizado de recursos asociados al grupo de estudio.
  - Mensajes de error en caso de permisos insuficientes o datos incompletos.
- **Criterios de Aceptación:**
  - Solo los miembros del grupo pueden acceder a los recursos del grupo.
  - Los miembros pueden registrar nuevos recursos.
  - El autor de un recurso puede editarlo o eliminarlo.
  - El líder y los moderadores pueden gestionar cualquier recurso del grupo.
  - El sistema debe permitir el acceso y descarga de los archivos asociados a los recursos.
  - Cada recurso debe estar correctamente asociado a su autor y al grupo correspondiente.
- **Actores:**
  - Miembros del grupo de estudio.

**MOGE_RF024: Gestionar mecanismo de alerta del grupo de estudio**

- **Descripción:** El sistema debe permitir que un miembro del grupo de estudio active un mecanismo de alerta para solicitar ayuda inmediata.

  Al activarse, el sistema deberá notificar a todos los miembros del grupo y generar una señal de alerta. Asimismo, el sistema deberá restringir la frecuencia de uso mediante un tiempo de reutilización definido.
- **Entrada:**
  - Identificador del usuario que activa la alerta.
  - Identificador del grupo de estudio.

- Acción de activación de la alerta.

- **Salida:**
  - Notificación enviada a los miembros del grupo.
  - Registro de la activación de la alerta.
  - Mensaje de error en caso de intento de uso dentro del tiempo de restricción.

- **Criterios de Aceptación:**
  - El sistema debe permitir activar la alerta a cualquier miembro del grupo.
  - El sistema debe notificar a todos los miembros del grupo de estudio.
  - El sistema debe registrar la fecha y hora de activación de la alerta.
  - El sistema debe restringir la activación del mecanismo a una vez cada 2 horas por grupo.

- **Actores:**
  - Miembro del grupo de estudio.

**MOGE_RF025: Solicitar encuentro personalizado**

- **Descripción:** El sistema debe permitir que un usuario envíe una solicitud de encuentro personalizado a otro usuario de la plataforma.

  La solicitud podrá incluir un mensaje opcional en el que se describa el motivo del encuentro. El usuario destinatario podrá visualizar la solicitud, leer el mensaje y decidir si la acepta o la rechaza.

- **Entrada:**
  - Identificador del usuario destinatario.
  - Mensaje de solicitud (opcional).

- **Salida:**
  - Registro de la solicitud en el sistema.
  - Notificación enviada al usuario destinatario.
  - Notificación de aceptación o rechazo al usuario solicitante.
- **Criterios de Aceptación:**
  - El sistema no debe permitir que un usuario envíe una solicitud a sí mismo.
  - El sistema debe evitar múltiples solicitudes pendientes entre los mismos usuarios.
  - El usuario destinatario debe poder aceptar o rechazar la solicitud.
  - El sistema debe notificar al solicitante el resultado de su solicitud.
- **Actores:**
  - Usuario solicitante.
  - Usuario destinatario.

**MOGE_RF026: Destacar publicación en el mural**

- **Descripción:** El sistema debe permitir al líder del grupo marcar una publicación del mural como destacada. Una publicación destacada cambia su interfaz para hacerse más visible respecto a las demás y debe mostrarse siempre en la primera posición del mural.
- **Entrada:**
  - Identificador del grupo de estudio.
  - Identificador de la publicación.
  - Acción de destacar o quitar destacado.
- **Salida:**

- Publicación marcada visualmente como destacada.
- Reordenamiento del mural mostrando la publicación destacada en primer lugar.

- **Criterios de Aceptación:**
  - Solo el líder del grupo puede destacar o quitar el destacado de una publicación.
  - La publicación destacada debe permanecer visible en primer lugar del mural.
  - El cambio de estado debe reflejarse en la interfaz en tiempo real.
  - Persistencia del nuevo estado de la publicación en base de datos.

- **Actores:**
  - Líder del grupo de estudio.

**MOGE_RF027: Crear automáticamente los componentes asociados al grupo de estudio**

- **Descripción:** El sistema debe generar automáticamente los componentes asociados a un grupo de estudio al momento de su creación.

  Al registrar un nuevo grupo, el sistema deberá crear y asociar los siguientes componentes: mural del grupo, chat grupal y módulo de recursos compartidos, dejándolos disponibles para su uso.

- **Entrada:**
  - Datos de creación del grupo de estudio.
  - Identificador del usuario líder.

- **Salida:**
  - Grupo de estudio registrado en el sistema.
  - Componentes asociados al grupo (mural, chat y recursos).

- **Criterios de Aceptación:**
  - El sistema debe crear automáticamente todos los componentes asociados sin intervención adicional del usuario.
  - Todos los componentes deben quedar correctamente vinculados al identificador del grupo de estudio.
  - El sistema debe garantizar la consistencia en la creación de los componentes asociados.
  - En caso de fallo en la creación de algún componente, el sistema debe evitar estados inconsistentes en el grupo de estudio.
- **Actores:**
  - Líder del grupo de estudio.

**MOGE_RF028: Buscar encuentro rápido aleatorio (BDP)**

- **Descripción:** El sistema debe permitir a los usuarios solicitar un encuentro rápido mediante un mecanismo de emparejamiento automático basado en una materia de interés.

  El sistema deberá agrupar a los usuarios que seleccionen el mismo tema en grupos de hasta tres participantes. Una vez conformado el grupo, el sistema generará un encuentro asociado, incluyendo una sugerencia de lugar y tiempo para la reunión.
- **Entrada:**
  - Identificador del usuario solicitante.
  - Tema seleccionado para el encuentro.
- **Salida:**
  - Grupo temporal de encuentro creado.
  - Notificación enviada a los usuarios emparejados.

  - Información del encuentro (lugar y tiempo sugerido).

- **Criterios de Aceptación:**

  - El emparejamiento debe realizarse únicamente entre usuarios con el mismo tema seleccionado.
  - El sistema debe conformar grupos de hasta tres participantes.
  - Si no hay suficientes usuarios disponibles, el sistema debe informar al usuario y permitirle esperar o cambiar el tema.
  - Una vez conformado el grupo, el sistema debe generar la información del encuentro.
  - Los usuarios emparejados deben recibir una notificación del encuentro generado.

- **Actores:**

  - Usuario solicitante.
  - Usuarios emparejados.

**MOGE_RF029: Gestionar chat de encuentro rápido (BDP)**

- **Descripción:** El sistema debe permitir la comunicación entre los usuarios emparejados dentro de un encuentro rápido mediante un canal de mensajería asociado al grupo temporal.

- **Entrada:**

  - Identificador del grupo de encuentro.
  - Identificador del usuario emisor.
  - Contenido del mensaje.

- **Salida:**

  - Mensaje enviado a los participantes del encuentro.
  - Registro del mensaje en el sistema.

- **Criterios de Aceptación:**
  - Solo los usuarios emparejados pueden enviar y recibir mensajes.
  - Los mensajes deben asociarse correctamente al grupo de encuentro.
  - El sistema debe mantener el historial de mensajes durante la vigencia del encuentro.
- **Actores:**
  - Usuarios emparejados en el encuentro.

**MOGE_RF030: Gestionar estado de emparejamiento (BDP)**

- **Descripción:** El sistema debe gestionar el estado de las solicitudes de encuentro rápido, permitiendo controlar el ciclo de vida del proceso de emparejamiento.
- **Entrada:**
  - Solicitud de emparejamiento por parte del usuario.
  - Eventos del sistema (EN_COLA, MATCH_MAKING, COMPLETADO, CANCELADO).
- **Salida:**
  - Actualización del estado de la solicitud.
  - Notificación al usuario sobre cambios en el estado.
- **Criterios de Aceptación:**
  - El sistema debe manejar estados como: EN_COLA, MATCH_MAKING, COMPLETADO, CANCELADO, EXPIRADO.
  - El sistema debe evitar que un usuario tenga múltiples solicitudes activas simultáneamente.
  - El sistema debe permitir la cancelación de la solicitud antes del emparejamiento.

- **Actores:**
  - Usuario solicitante.

**MOGE_RF031: Crear nota adhesiva**

- **Descripción:** El sistema debe permitir a los miembros de un grupo de estudio crear notas adhesivas asociadas al grupo, como mecanismo de comunicación asíncrona.

  Cada nota deberá contener un título, un contenido con una longitud máxima de 600 caracteres y un atributo de color. Asimismo, la nota podrá dirigirse a todos los miembros del grupo, a un miembro específico o a un subconjunto de miembros.

- **Entrada:**
  - Identificador del usuario emisor.
  - Identificador del grupo de estudio.
  - Identificador(es) de los usuarios destinatarios (opcional).
  - Título de la nota.
  - Color de la nota.
  - Contenido de la nota.

- **Salida:**
  - Registro de la nota adhesiva en el sistema.
  - Notificación enviada a los usuarios destinatarios.

- **Criterios de Aceptación:**
  - Solo los miembros del grupo pueden crear notas adhesivas.
  - El sistema debe validar que los destinatarios pertenezcan al grupo.
  - El contenido de la nota no debe exceder los 600 caracteres.

  - El sistema debe permitir definir notas dirigidas a todos, a uno o a varios miembros.
  - La nota debe quedar correctamente asociada a su autor y al grupo de estudio.

- **Actores:**
  - Miembro del grupo de estudio.

**MOGE_RF032: Calcular top de publicadores**

- **Descripción:** El sistema debe calcular periódicamente el ranking de los usuarios con mayor número de publicaciones destacadas dentro de un intervalo de tiempo definido (semanal), con el fin de generar un listado de ”top publicadores”.

  Este proceso deberá ejecutarse automáticamente mediante una tarea programada del sistema.

- **Entrada:**
  - Registros de publicaciones destacadas.
  - Identificador de los usuarios autores de las publicaciones.
  - Intervalo de tiempo de evaluación (semana actual).

- **Salida:**
  - Listado ordenado de usuarios con mayor número de publicaciones destacadas.
  - Registro actualizado en la tabla de top de publicadores.

- **Criterios de Aceptación:**
  - El cálculo debe ejecutarse automáticamente una vez por semana.
  - El sistema debe considerar únicamente las publicaciones destacadas dentro del intervalo de tiempo definido.
  - El ranking debe ordenarse de forma descendente según la cantidad de publicaciones destacadas.

  - En caso de empate, el sistema debe aplicar un criterio adicional de ordenamiento (por ejemplo, fecha de última publicación destacada).
  - El sistema debe garantizar la consistencia de los datos almacenados en el ranking.

- **Actores:**
  - Sistema (tarea programada).

**MOGE_RF033: Calcular top de publicaciones con mayor número de reacciones**

- **Descripción:** El sistema debe calcular periódicamente el ranking de las publicaciones con mayor número de reacciones de forma semanal, con el fin de identificar las publicaciones más relevantes dentro de todos los grupo de estudio.

  Este proceso deberá ejecutarse automáticamente mediante una tarea programada del sistema.

- **Entrada:**
  - Registros de reacciones asociadas a las publicaciones.
  - Identificador de las publicaciones.
  - Identificador de los autores de las publicaciones.
  - Intervalo de tiempo de evaluación (semana actual).

- **Salida:**
  - Listado ordenado de publicaciones con mayor número de reacciones.
  - Registro actualizado en la tabla de top de publicaciones.

- **Criterios de Aceptación:**
  - El cálculo debe ejecutarse automáticamente una vez por semana.
  - El sistema debe considerar únicamente las reacciones generadas dentro del intervalo de tiempo definido.

  - El ranking debe ordenarse de forma descendente según la cantidad de reacciones.
  - En caso de empate, el sistema debe aplicar un criterio adicional de ordenamiento (por ejemplo, fecha de creación de la publicación).
  - El sistema debe garantizar la consistencia de los datos almacenados en el ranking.

- **Actores:**
  - Sistema (tarea programada).

**MOGE_RF034: Renovar sesión de usuario**

- **Descripción:** El sistema debe permitir la renovación de la sesión del usuario mediante la obtención de un nuevo token de autenticación desde el servicio principal.
- **Entrada:**
  - Token actual del usuario.
- **Salida:**
  - Nuevo token de autenticación válido.
- **Criterios de aceptación:**
  - El sistema debe validar la identidad del usuario antes de emitir un nuevo token.
  - El nuevo token debe reemplazar al anterior.

### *4.1.4 Requerimientos no funcionales*

**Rendimiento general**

**MOGE_NF01: Tiempo de redirección del sistema**

- **Descripción:** El sistema debe garantizar tiempos de redirección eficientes entre módulos internos, con el fin de proporcionar una experiencia de usuario fluida.

- **Criterios de aceptación:**
  - El tiempo total de redirección no debe superar los **2 segundos** en condiciones normales de operación.
  - El tiempo de respuesta debe mantenerse estable en al menos el **95 % de las solicitudes**.
  - El sistema debe soportar hasta **100 usuarios concurrentes** sin degradación significativa del rendimiento.

**MOGE_NF02: Desempeño de comunicaciones en tiempo real (WebSocket)**

- **Descripción:** El sistema debe garantizar el desempeño adecuado de las comunicaciones en tiempo real mediante el uso de WebSocket, asegurando baja latencia en el envío y recepción de mensajes.
- **Criterios de aceptación:**
  - El sistema debe soportar al menos **100 conexiones simultáneas activas**.
  - El tiempo de entrega de mensajes no debe superar los **300 ms** en condiciones normales.
  - El sistema debe mantener la estabilidad de las conexiones sin pérdidas de sesión en al menos el **95 % de los casos**.

**MOGE_NF03: Optimización de carga de recursos multimedia**

- **Descripción:** El sistema debe optimizar la carga y almacenamiento de recursos multimedia (imágenes, archivos y otros contenidos), con el fin de reducir tiempos de respuesta y consumo de ancho de banda.
- **Criterios de aceptación:**
  - Los archivos multimedia deben almacenarse de forma estructurada por categorías.
  - El tiempo de carga de recursos multimedia no debe superar los **2 segundos** en condiciones normales.

- El sistema debe soportar carga concurrente de archivos sin afectar significativamente el rendimiento general.

**MOGE_NF04: Optimización de consultas a base de datos**

- **Descripción:** El sistema debe optimizar las consultas a la base de datos para garantizar eficiencia en el acceso a la información y evitar sobrecarga en el servidor.
- **Criterios de aceptación:**
  - El sistema debe evitar el problema de consultas tipo N+1 mediante el uso de estrategias como *JOIN FETCH* en JPQL.
  - Las consultas deben aprovechar índices definidos en la base de datos.
  - El tiempo de respuesta de las consultas críticas no debe superar los **500 ms**.
  - Se debe garantizar la integridad referencial conforme al esquema de la base de datos institucional.

**MOGE_NF05: Tiempo de respuesta de la API**

- **Descripción:** El sistema debe garantizar tiempos de respuesta adecuados en los endpoints de la API, asegurando eficiencia en las operaciones de consulta y modificación de datos.
- **Criterios de aceptación:**
  - Las operaciones de lectura (GET) deben responder en menos de **500 ms**.
  - Las operaciones de escritura (POST, PUT, PATCH) deben responder en menos de **1 segundo**.
  - El sistema debe mantener estos tiempos bajo una carga de al menos **100 usuarios concurrentes**.

**Seguridad**

**MOGE_NF06: Autenticación mediante token firmado (Bearer)**

- **Descripción:** El sistema debe implementar un mecanismo de autenticación basado en tokens de acceso transmitidos mediante el esquema Bearer en las solicitudes HTTP.

  El token será generado por el servicio principal y validado por el microservicio mediante un mecanismo de firma criptográfica basado en HMAC, garantizando la integridad y autenticidad de la información contenida.

- **Criterios de aceptación:**

  - Todas las solicitudes a la API deben incluir un token en el encabezado `Authorization` bajo el esquema Bearer.
  - El sistema debe validar la firma del token utilizando un algoritmo HMAC compartido.
  - El token debe tener un tiempo de expiración máximo de **1 hora**.
  - El sistema debe rechazar solicitudes con tokens inválidos, alterados o expirados.
  - El sistema debe permitir la renovación del token mediante validación del usuario en el servicio principal.

**MOGE_NF07: Control de acceso por origen (CORS)**

- **Descripción:** El sistema debe restringir el acceso a sus recursos mediante la configuración de políticas de CORS (Cross-Origin Resource Sharing), permitiendo únicamente solicitudes provenientes de orígenes autorizados.

- **Criterios de aceptación:**

  - El sistema debe permitir solicitudes únicamente desde el dominio del servicio principal.
  - Las solicitudes provenientes de orígenes no autorizados deben ser rechazadas automáticamente.

- El sistema debe configurar adecuadamente los encabezados CORS en las respuestas HTTP.

**MOGE_NF08: Validación y sanitización de entradas**

- **Descripción:** El sistema debe validar y sanitizar todas las entradas de usuario con el fin de prevenir vulnerabilidades como inyección de código (SQL Injection, XSS) y ejecución de contenido malicioso.
- **Criterios de aceptación:**
  - Todas las entradas de usuario deben ser validadas en el servidor.
  - El sistema debe implementar mecanismos de sanitización para contenido HTML y texto ingresado por el usuario.
  - El sistema debe prevenir la ejecución de scripts o código malicioso en los datos almacenados o mostrados.
  - El sistema debe validar archivos cargados por el usuario (tipo, tamaño y contenido).

**MOGE_NF09: Validación de token en backend**

- **Descripción:** El sistema debe validar todos los tokens de autenticación en el backend del microservicio mediante un mecanismo de verificación de firma basado en HMAC, garantizando que la autenticación no dependa del cliente.
- **Criterios de aceptación:**
  - Toda solicitud protegida debe ser interceptada por un filtro de autenticación en el backend.
  - El sistema debe validar la integridad del token mediante HMAC.
  - El sistema debe rechazar solicitudes con tokens inválidos o manipulados.
  - El sistema debe rechazar solicitudes sin token.

**MOGE_NF10: Gestión de expiración y renovación de tokens**

- **Descripción:** El sistema debe controlar la vigencia de los tokens de autenticación, estableciendo un tiempo de expiración definido y un mecanismo de renovación mediante validación del servicio principal.
- **Criterios de aceptación:**
  - El token debe expirar después de un máximo de 1 hora.
  - El sistema debe rechazar tokens expirados.
  - El sistema debe permitir la renovación del token mediante interacción con el servicio principal.

**MOGE_NF11: Protección de endpoints mediante autenticación**

- **Descripción:** El sistema debe restringir el acceso a los endpoints del microservicio únicamente a usuarios autenticados.
- **Criterios de aceptación:**
  - Todos los endpoints sensibles deben requerir autenticación.
  - El sistema debe devolver códigos HTTP 401 o 403 según corresponda.
  - No se debe permitir acceso a recursos sin validación previa del token.

**Mantenibilidad**

**MOGE_NF12: Estándares y convenciones de desarrollo**

- **Descripción:** El sistema debe desarrollarse siguiendo estándares y convenciones definidas por el grupo de investigación, con el fin de garantizar la consistencia, legibilidad y mantenibilidad del código.

- **Criterios de aceptación:**
  - El proyecto debe mantener una estructura de arquitectura consistente (hexagonal + DDD lite organizada por features).
  - El código debe seguir convenciones de nomenclatura definidas (clases, métodos, paquetes).
  - Se debe utilizar *Lombok* para reducir código repetitivo en entidades y DTOs.
  - Los módulos del sistema deben estar claramente separados por responsabilidades.
  - No deben existir clases con múltiples responsabilidades (principio de responsabilidad única).

**MOGE_NF13: Documentación técnica del sistema**

- **Descripción:** El sistema debe contar con documentación técnica suficiente que permita su comprensión, mantenimiento y evolución por parte de nuevos desarrolladores.
- **Criterios de aceptación:**
  - Cada módulo del sistema debe contar con una descripción de su propósito y funcionamiento.
  - Las APIs deben estar documentadas mediante herramientas como OpenAPI/Swagger.
  - El proyecto debe incluir un archivo README con instrucciones de despliegue y ejecución.
  - Las decisiones arquitectónicas relevantes deben estar documentadas.

**MOGE_NF14: Mantenibilidad de la base de datos**

- **Descripción:** La estructura de la base de datos debe diseñarse siguiendo convenciones institucionales que faciliten su comprensión, evolución y mantenimiento a lo largo del tiempo, garantizando consistencia con la base de datos existente en la plataforma COMA.

- **Criterios de aceptación:**
  - Las tablas deben seguir convenciones de nomenclatura consistentes con la base de datos institucional, clasificándose de la siguiente manera:
    - **TP_NombreTabla:** Tablas principales o entidades base del sistema.
    - **TB_NombreTabla:** Tablas básicas o de parametrización.
    - **TR_NombreTabla:** Tablas relacionales o intermedias.
  - Los nombres de tablas y columnas deben ser descriptivos, manteniendo consistencia en el uso de mayúsculas, minúsculas y convenciones establecidas.
  - Las relaciones entre entidades deben definirse mediante restricciones de integridad referencial (claves foráneas).
  - El modelo de datos debe permitir la trazabilidad de las relaciones entre entidades del sistema.

**Capacidad**

**MOGE_NF15: Límites de almacenamiento de archivos**

- **Descripción:** El sistema debe establecer restricciones en el tamaño de los archivos cargados, con el fin de garantizar un uso eficiente del almacenamiento y evitar la degradación del rendimiento.
- **Criterios de aceptación:**
  - El tamaño máximo permitido para archivos de imagen debe ser de **2 MB**.
  - El tamaño máximo permitido para documentos (PDF, Word, entre otros) debe ser de **5 MB**.
  - El sistema debe rechazar archivos que excedan estos límites mostrando un mensaje claro al usuario.

- El sistema debe validar el tamaño del archivo tanto en frontend como en backend.

**Usabilidad y ux**

**MOGE_NF16: Interacción del panel lateral (sidebar)**

- **Descripción:** El sistema debe garantizar una interacción fluida y consistente del panel lateral en diferentes resoluciones de pantalla.
- **Criterios de aceptación:**
  - El tiempo de respuesta para mostrar u ocultar el sidebar debe ser inferior a **500 ms**.
  - El sidebar debe adaptarse correctamente a resoluciones de escritorio y dispositivos móviles sin pérdida de funcionalidad.
  - No deben presentarse desbordamientos o superposición de elementos en la interfaz.

**MOGE_NF17: Retroalimentación visual mediante notificaciones (TOAST)**

- **Descripción:** El sistema debe proporcionar retroalimentación inmediata al usuario mediante notificaciones visuales (toast) ante acciones relevantes.
- **Criterios de aceptación:**
  - Las notificaciones deben mostrarse en un tiempo inferior a **300 ms** después de la acción.
  - Las notificaciones deben desaparecer automáticamente después de un tiempo configurable (entre 3 y 5 segundos).

**MOGE_NF18: Latencia en actualización de interfaz**

- **Descripción:** El sistema debe reflejar en la interfaz de usuario los cambios generados por eventos del sistema en un tiempo adecuado que garantice una experiencia en tiempo real.
- **Criterios de aceptación:**

- Las actualizaciones provenientes de eventos en tiempo real deben reflejarse en la interfaz en menos de **1 segundo**.
- El sistema debe mantener consistencia visual entre los usuarios conectados.
- No deben presentarse inconsistencias temporales visibles en la interfaz.

## 4.2 Proceso de trabajo y marcos de trabajo ágil

Para la gestión y organización del desarrollo del microservicio, se adoptó el marco de trabajo ágil *Scrum*, implementado mediante la plataforma JIRA. Scrum propone un proceso iterativo e incremental en el que el producto se construye a través de entregas parciales llamadas *sprints*, guiadas por un conjunto de historias de usuario priorizadas en el *product backlog*.

De acuerdo con Schwaber y Sutherland (2020), Scrum permite gestionar proyectos complejos mediante roles claramente definidos (Scrum Master, Product Owner y Equipo de Desarrollo), eventos periódicos como reuniones diarias (daily stand-up) de máximo diez minutos y la entrega frecuente de incrementos funcionales del producto.

En este proyecto, al tratarse de un equipo pequeño, los roles fueron distribuidos entre los participantes, asumiendo porcentajes de responsabilidad según la afinidad y las competencias de cada integrante. La siguiente tabla resume la distribución de roles y su nivel de participación o afianzamiento dentro del proceso de trabajo:

**Tabla 3**
*Distribución de roles y participación en Scrum*

| **Integrante** | **Scrum Master** | **Product Owner** | **Desarrollo Back-End** | **Desarrollo Front-End** | **QA / Bases de Datos** |
|---|---|---|---|---|---|
| Andrés Felipe Olivar Gutiérrez | 70 % | 0 % | 40 % | 60 % | 50 % |
| Danna Valentina Prada Briceño | 30 % | 0 % | 60 % | 40 % | 50 % |
| Luis Ignacio González (Director) | 0 % | 100 % | 0 % | 0 % | 0 % |

## 4.3 Diferencias entre épicas, historias de usuario y sprints

Dentro del marco de trabajo ágil Scrum, es fundamental comprender la relación jerárquica y funcional entre las épicas, las historias de usuario y los sprints. Aunque estos elementos interactúan entre sí, cada uno cumple un propósito distinto dentro del proceso de desarrollo y planificación del producto.

### *4.3.1 Épicas*

Una épica es una funcionalidad amplia, compleja o de alto nivel, que representa un objetivo significativo dentro del sistema. Debido a su magnitud, una épica no puede completarse en un único sprint, por lo que se divide en varias historias de usuario más pequeñas y manejables. Las épicas permiten organizar el *product backlog* en bloques funcionales coherentes y facilitan la visión estratégica del desarrollo.

Ejemplo de una épica en este proyecto:

- **Épica:** Gestión de grupos de estudio.
- **Historias asociadas:** Crear grupo, programar horarios, visualizar miembros, editar información del grupo, entre otras.

### *4.3.2 Historias de usuario*

Las historias de usuario representan unidades de trabajo pequeñas, específicas y orientadas a la entrega de valor. Cada historia describe una necesidad puntual desde la perspectiva del usuario, y es lo suficientemente acotada como para ser desarrollada dentro de un solo sprint. Las historias asociadas a una épica permiten completar dicha funcionalidad de manera incremental.

### *4.3.3 Sprints*

Un sprint es un ciclo de trabajo de duración fija (usualmente entre una y cuatro semanas), en el cual el equipo selecciona un conjunto de historias de usuario del *backlog* y se compromete

a entregarlas como un incremento funcional del producto. Los sprints no contienen épicas, sino historias de usuario, y pueden abarcar partes de múltiples épicas.

### *4.3.4 Relación entre épicas, historias y sprints*

**Tabla 4**
*Comparativo entre épicas, historias de usuario y sprints*

| Concepto | Qué es | Duración | Qué contiene |
|---|---|---|---|
| **Épica** | Funcionalidad grande o módulo completo. | Varios sprints. | Historias de usuario. |
| **Historia de usuario** | Requisito pequeño que entrega valor incremental. | Un sprint. | Tareas o actividades. |
| **Sprint** | Ciclo de trabajo de duración fija. | 1–4 semanas. | Historias de usuario seleccionadas para el incremento. |

En total, se planificaron cuatro sprints para este proyecto, distribuidos según el calendario del proyecto y orientados a entregar un incremento funcional al finalizar cada periodo. La versión beta del microservicio se programó para ser entregada al cierre del último sprint.

### *4.3.5 Historias de usuario y configuración del trabajo en JIRA*

Las historias de usuario constituyen la unidad mínima de trabajo dentro de Scrum. Estas permiten descomponer los requerimientos funcionales del sistema en tareas manejables, orientadas a las necesidades concretas del usuario final. Un solo requerimiento funcional puede dividirse en una o múltiples historias de usuario, permitiendo una planificación más granular y controlada del desarrollo.

Cada historia de usuario se redactó utilizando la estructura recomendada por Cohn (2004)

- **Como** *[actor]*,
- **quiero** *[funcionalidad]*,
- **para** *[beneficio u objetivo]*.

A esta estructura se le añadieron criterios de aceptación claros, siguiendo el formato *Given-When-Then*, que permite validar la correcta implementación de cada historia.

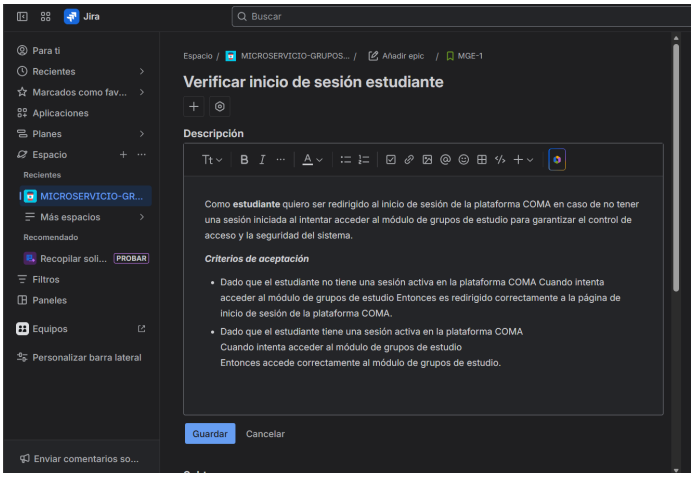
Además, cada historia recibió un puntaje de complejidad determinado en reunión conjunta del equipo. Dicho puntaje se asignó con base en la serie de Fibonacci (1, 2, 3, 5, 8, 13...), una práctica común en planificación ágil recomendada por Schwaber y Sutherland (2020) para estimar el esfuerzo relativo del trabajo. En este proyecto, un puntaje de 3 equivalía aproximadamente a dos días de trabajo.

JIRA permitió gestionar las historias, asignarlas a los desarrolladores correspondientes, dar seguimiento a su avance y organizarlas dentro de los sprints.

A continuación, en la Figura 3 se muestra la plataforma JIRA con una de las primeras historias de usuario y uno de los formatos utilizados. Este formato se usó para historias de usuario que no incluían acciones con atributos, y solo se muestran los criterios de aceptación. Una vez documentadas las historias, fueron llevadas al backlog para su posterior puntuación y asignación.

**Figura 3**

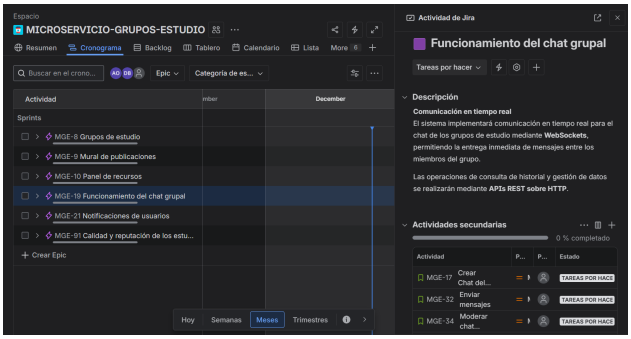
*Vista desde el panel JIRA de una historia de usuario sencilla*



La historia de usuario presentada anteriormente fue una de las más sencillas del proyecto. A continuación, utilizando todas las herramientas que ofrece la plataforma, se muestra cómo fue posible desglosar un requerimiento complejo en una historia de usuario con subtareas y comentarios para los desarrolladores. La siguiente secuencia de imágenes muestra el proceso de documentación utilizado en JIRA, pasando por todas las épicas y enfocándonos específicamente en el componente del chat grupal. El lector puede asumir que todas las demás épicas fueron tratadas de manera similar, por lo cual no es necesario mostrar cada una de ellas individualmente, sino el proceso general aplicado.

**Figura 4**

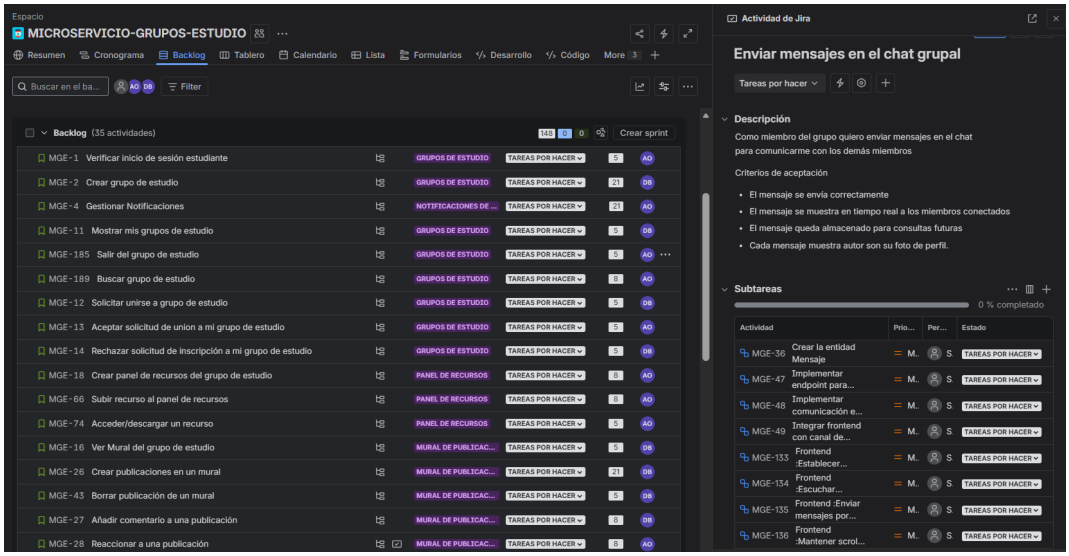
*Épica documentada del chat del grupo de estudio*




Como se observa en la figura anterior, cada épica cuenta con una descripción general cuyo propósito es comunicar al desarrollador la idea principal del componente. En esta sección se resume el funcionamiento esperado y las tecnologías involucradas. Para el caso particular del chat grupal, se menciona el uso de WebSockets como mecanismo para la comunicación en tiempo real.

Tal como se explicó previamente, una épica está compuesta por varias historias de usuario, y en conjunto todas ellas conforman el backlog del proyecto. En la siguiente imagen se presenta el estado final del backlog, en el cual se evidencia la asociación entre cada historia de usuario y su respectiva épica.

**Figura 5**
*Vista del backlog en Jira*




En la figura se observa una muestra del backlog completo. Cada historia de usuario incluye su identificador (por ejemplo, MGE-1), el nombre de la épica asociada resaltada en color morado, su estado actual que consiste en uno de los tres estados existentes en jira: “Por hacer”, “En curso” y “Finalizado”, la complejidad estimada y finalmente la persona encargada de dicha historia.

Como se había mencionado al comienzo de este capítulo, durante una reunión con el equipo de desarrollo se acordó utilizar la serie de Fibonacci para estimar la complejidad de las historias de usuario. A partir de ello, se construyó una tabla de conciliación donde, por ejemplo, una historia con puntaje de 3 tiene una duración máxima estimada de 1.5 días (puntaje/2), y así sucesivamente, tal como se muestra en la siguiente tabla.

**Tabla 5**
*Relación entre calificación Fibonacci y días máximos de desarrollo*

| Calificación | Días de desarrollo máximo |
|---|---|
| 3 | 1.5 |
| 5 | 2.5 |
| 8 | 4 |
| 13 | 6.5 |

En la parte derecha de la figura 5 también se aprecia la historia de usuario denominada “Enviar mensajes en el chat grupal”, la cual contiene varias subtareas. Aunque estas puedan parecer nuevas historias de usuario, en realidad corresponden a actividades técnicas específicas que nosotros como desarrolladores debemos realizar para implementar la historia principal. Por ejemplo, una subtarea asociada a esta historia consiste en la creación de la entidad *Mensaje*.

En este sentido, la relación entre los elementos puede resumirse así: la épica proporciona una visión general del componente y su propósito; la historia de usuario describe qué debe hacer el sistema desde la perspectiva del usuario; y las subtareas indican cómo se implementará técnicamente dicha funcionalidad.

Finalmente, una vez completado el backlog, se procedió a la planificación de los sprints del proyecto y a ajustar la complejidad de las tareas con el fin de equilibrar los tiempos estimados de desarrollo con la carga real de trabajo de los desarrolladores.

También fue necesario considerar el cronograma general del proyecto, ya que se buscaba respetar la planificación presentada inicialmente. Por ello, una vez finalizada la semana de documentación, y teniendo en cuenta que en ese momento aún no se contaban con los diagramas, se decidió que los sprints iniciarían una semana después, correspondiente a la fase de diseño. Posteriormente, se tendrían 6 semanas destinadas a la fase de implementación; en consecuencia, los sprints debían organizarse dentro de ese periodo de 6 semanas. Tomando esto en cuenta, se obtuvo la siguiente organización:

### *4.3.6 Primer sprint: Base del microservicio*

Como se observa en la figura 6, el primer sprint corresponde a una aproximación inicial a la aplicación. En este sprint se prioriza la construcción de la base del microservicio, enfocándose en disponer de la estructura principal del sistema, aunque aún sin todas las funcionalidades completas. Sobre esta base se desarrollarán posteriormente los componentes que complementen y den forma final al microservicio.

El objetivo de este sprint era que un usuario pudiera ingresar al microservicio de forma

correcta y autenticada, y pudiera crear un entorno grupal vacío, es decir, un cascarón de opciones y vista previa de las funcionalidades con métodos básicos. Además, también se inició una de las funcionalidades transversales del microservicio correspondiente a las notificaciones, ya que algunas debían ser asíncronas y otras en tiempo real.

En este primer sprint surgieron los primeros detalles y fallos en la planificación. Si bien el sprint se pudo terminar a tiempo, no todas las historias de usuario incluidas en él pudieron completarse. Un ejemplo son las notificaciones, las cuales se completaron hasta el final del quinto sprint debido a la diversidad de funcionalidades en las que estaban involucradas, por lo que debían ir modificándose en cada sprint hasta ser finalizadas.

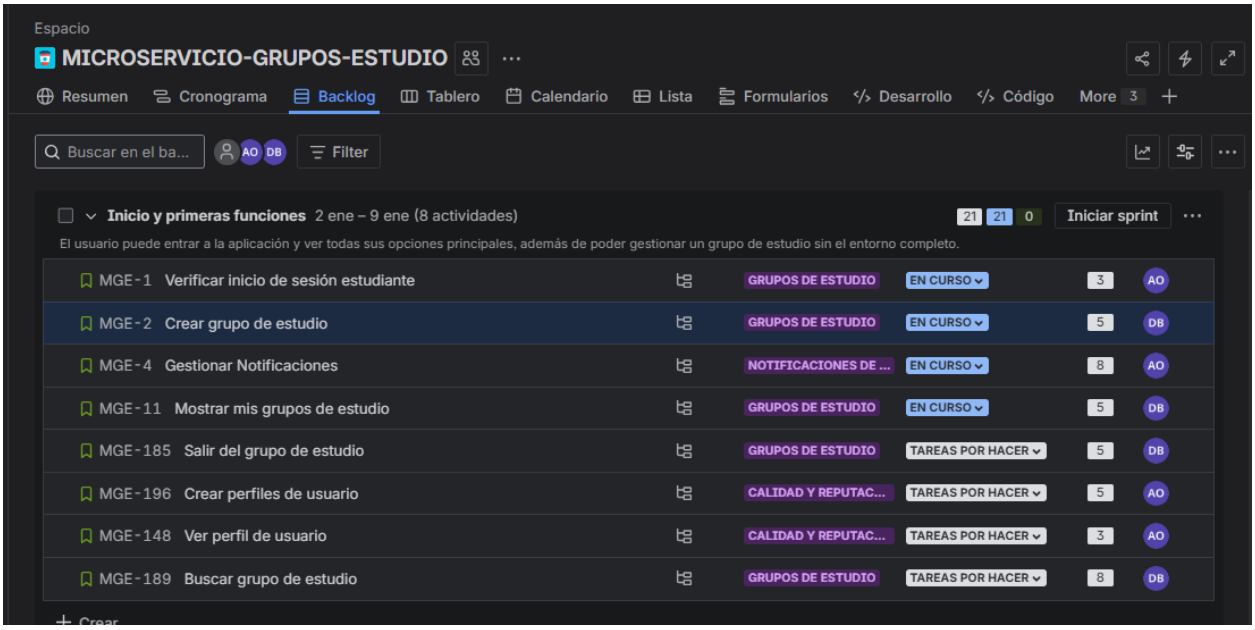
Algo que añadir en este sprint es que ya se contaba con una estructura armada desde antes de iniciarlo. De no haber sido así, es muy probable que no se hubiera logrado completar el primer sprint a tiempo. En este caso, la experiencia previa utilizando el marco de trabajo Scrum permitió anticipar estos posibles inconvenientes. Por esta razón, se decidió asignar tareas de menor complejidad en este sprint, considerando que las fases iniciales suelen requerir mayor tiempo hasta alcanzar un ritmo de desarrollo más eficiente.

Para este sprint se tuvo una reunión con el product owner durante su desarrollo y una al finalizar el sprint para su revisión y retroalimentación.

En la retroalimentación de este primer sprint se discutieron los puntajes asignados a las historias de usuario y si estos estaban bien ajustados a los tiempos empleados en su desarrollo. Asimismo, se evaluaron posibles aspectos por mejorar o ajustar. Sin embargo, como se mencionó, a pesar de haber tenido que trasladar una historia al siguiente sprint, el resto de actividades se completó dentro del tiempo estimado, por lo que se decidió continuar sin realizar cambios significativos hacia el siguiente sprint. Antes de dar inicio al segundo sprint, el equipo realizó una revisión general de la arquitectura y de los componentes implementados durante esta primera iteración. Esta actividad permitió validar que la estructura base desarrollada era lo suficientemente flexible para soportar la incorporación de nuevas funcionalidades en los siguientes sprints sin requerir cambios significativos en su diseño.

**Figura 6**

*Cronograma y visualización del primer sprint*




### *4.3.7 Segundo sprint: Primera versión del chat*

Como se observa en la figura 7, el segundo sprint tenía como objetivo presentar un chat grupal en un estado inicial, permitiendo a los usuarios enviar y recibir mensajes en tiempo real.

En este sprint ocurrió una situación interesante. En el sprint anterior se consideró que los tiempos de desarrollo habían sido adecuados, y se estimó que este segundo sprint, al ser más completo, tomaría aproximadamente dos semanas, por lo cual se le asignó dicho tiempo. Sin embargo, el resultado fue que el sprint se completó casi en el sexto día. Esto evidencia que, una vez más, la estimación del tiempo de desarrollo y la complejidad de las tareas no fue completamente precisa.

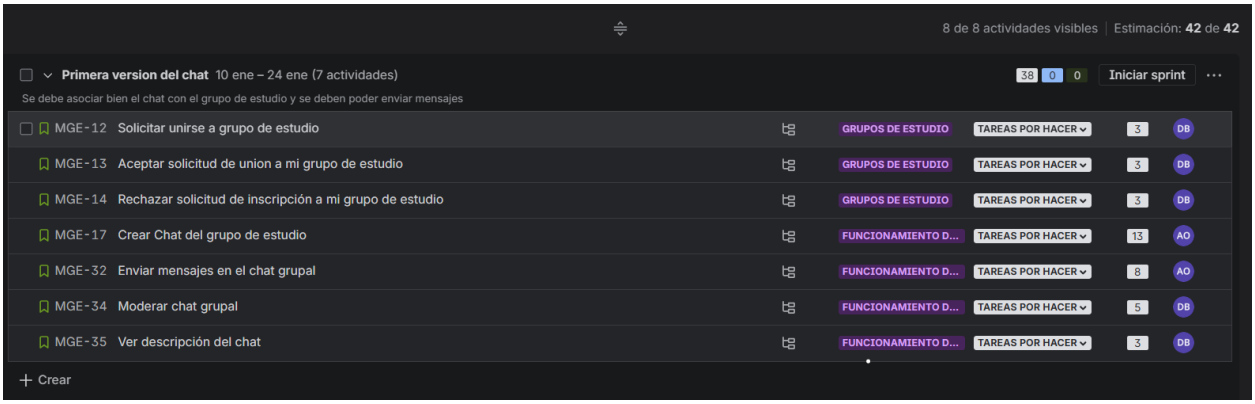
Esta situación pudo deberse a que era la primera vez que se trabajaba con tecnologías de comunicación en tiempo real mediante WebSockets, lo cual inicialmente se percibía como altamente complejo. No obstante, es importante destacar que, para este punto del proyecto, las tablas de la base de datos ya se encontraban diseñadas y estructuradas previamente. Esta planificación previa facilitó significativamente el desarrollo, permitiendo abordar un dominio desconocido con

mayor claridad y eficiencia, lo que se tradujo en una reducción aproximada de una semana en el tiempo estimado del sprint.

En conclusión, este sprint se completó antes de lo previsto, lo que permitió disponer de una semana de margen que pudo ser redistribuida en sprints posteriores. Durante este sprint se realizó una reunión intermedia con el Product Owner y una reunión final para la validación y retroalimentación del avance.

**Figura 7**

*Cronograma y visualización del segundo sprint*



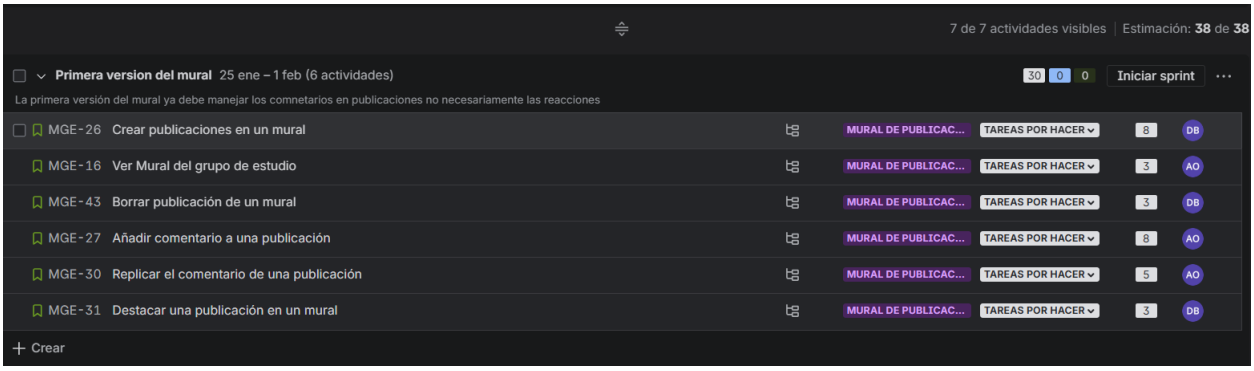

### *4.3.8 Tercer sprint: Primera versión del mural*

Como se observa en la figura 8, el tercer sprint correspondía a la primera versión del mural, un centro de publicaciones y recursos compartidos con información del grupo y algunas funcionalidades básicas extra. Realmente no sonaba tan complejo en su momento, pero esta acumulación de funcionalidades dentro de otras, como lo son hacer una publicación, agregar comentarios y permitir respuestas a dichos comentarios, terminó siendo muy compleja.

Recordamos este sprint como el más agotador de todos; realmente queríamos terminarlo a tiempo y, debido a que eran momentos en los que estábamos de vacaciones, podíamos pasar horas y horas intentando finalizarlo. Aun así, faltando un par de días para terminarlo, nos reunimos ambos desarrolladores y consideramos dividir el sprint en dos, dejando para el cuarto sprint una mezcla del panel de recursos con algunas historias de las publicaciones.

El tercer sprint se logró “terminar” (delegando tareas al siguiente sprint) a tiempo, y hubo una reunión virtual con el product owner al finalizar.

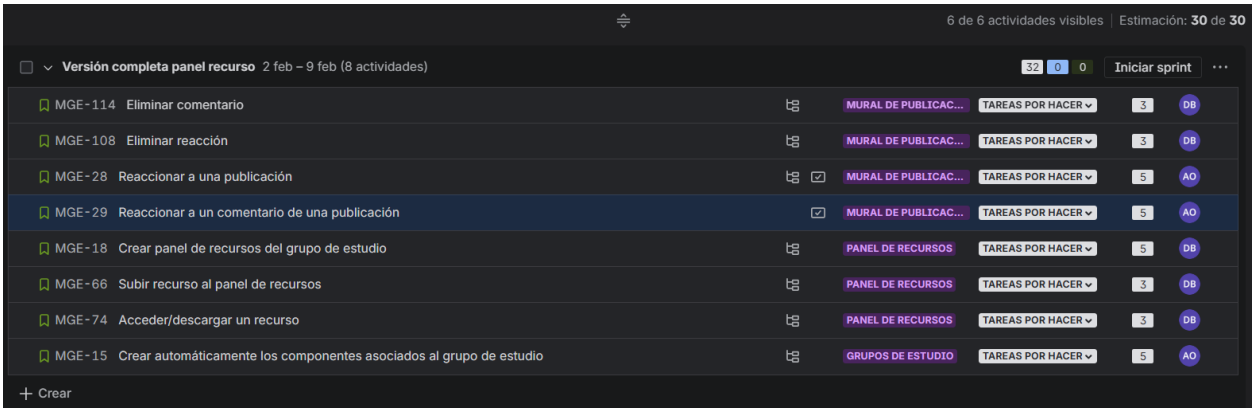
**Figura 8**
*Cronograma y visualización del tercer sprint*




### *4.3.9 Cuarto sprint: Versión completa panel de recursos*

Como se observa en la figura 9, el cuarto sprint ahora correspondía al panel de recursos compartidos y a algunas historias de usuario migradas desde las publicaciones del tercer sprint. Para este sprint ya teníamos más experticia en el desarrollo y la velocidad de implementación estaba incrementando, por lo que no se tuvieron problemas para culminarlo y se logró completar a tiempo. En este sprint se tuvo una reunión virtual con el product owner al finalizarlo.

**Figura 9**
*Cronograma y visualización del cuarto sprint*



### *4.3.10 Quinto sprint: Funciones de calificación*

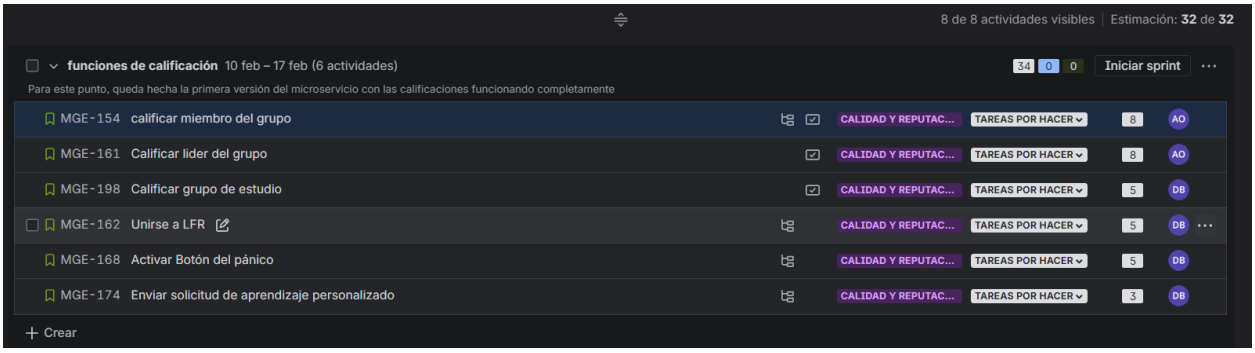
Como se observa en la figura 10, el quinto sprint correspondía a las calificaciones, reseñas y funcionalidades nuevas como el botón del pánico y el LFR (Looking for a Random), que pasó a llamarse BDP (Buscador de parches) tras charlas con el product owner.

Teníamos muchas ganas de llegar a este sprint porque íbamos a poder desarrollar estas funcionalidades más dinámicas; además, ya estábamos sin tareas pendientes de sprints anteriores y, por ende, podíamos centrarnos en funcionalidades específicas. Sin embargo, estas nuevas funcionalidades resultaron más tediosas de lo esperado debido al nivel de creatividad y lógica que hay detrás de ellas, lo que hizo el código más difícil de implementar.

Aun así, se lograron desarrollar, aunque debemos admitir que esta primera versión de las funcionalidades no cumplió completamente con nuestras expectativas, por lo que debían mejorarse en las fases de pruebas y validación, o incluso destinando una semana adicional para ajustes generales. Al fin y al cabo, estos cinco sprints correspondían a una fase beta, cuyo objetivo es ser desplegada en un entorno de pruebas para su posterior mejora.

En este sprint se tuvo una reunión con el product owner al finalizar.

**Figura 10**
*Cronograma y visualización del quinto sprint*




Finalmente podemos ver un resumen de lo hecho en el panel de JIRA

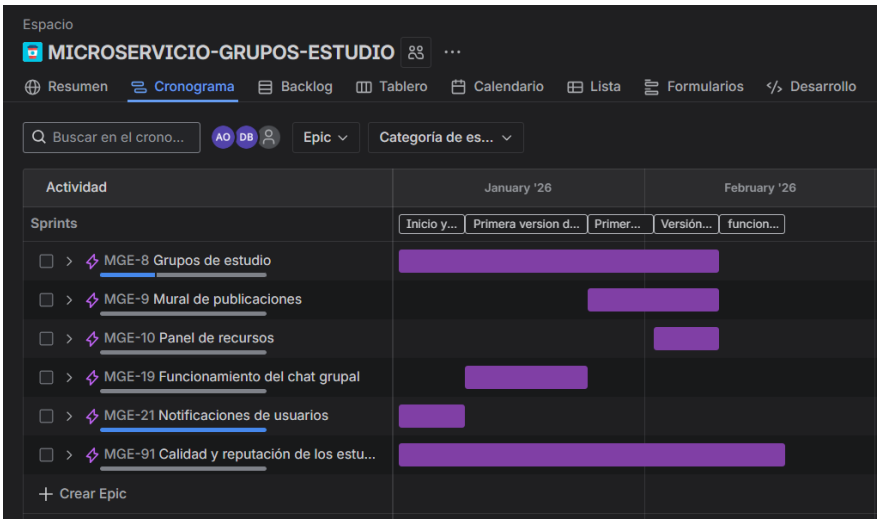
**Figura 11**

*Actividades por hacer en los sprints: resumen en jira*


También podemos ver el cronograma grafico desde JIRA con respecto a las épicas y su relacion en cada sprint:

**Figura 12**

*Cronograma y relación Épicas-Sprints*



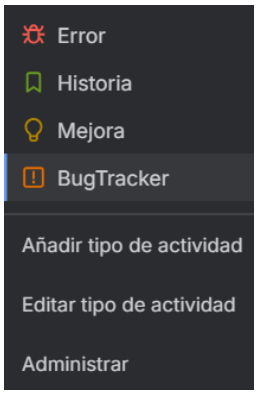
### *4.3.11 Trabajo posterior a los sprints*

Una vez finalizados los sprints se tenía una fase beta del servicio. Había errores o bugs que conocíamos y otros que seguramente irían surgiendo, por lo que se pensó que, antes de subirlo al servidor de pruebas, lo mejor era implementar un BugTracker.

La idea era la siguiente: tomar una semana para tratar de pulir los errores que conocíamos y también para implementar una herramienta con la cual los usuarios de prueba pudieran reportar de forma fácil y rápida los bugs que encontraran a medida que probaban el servicio. Luego tomaríamos esos reportes y empezaríamos a armar un nuevo backlog en JIRA con ellos.

Primero se crearon nuevas etiquetas véase la figura 13.

**Figura 13**
*Etiquetas JIRA para el backlog*




**Error:** Se cataloga como error si funciona de forma incorrecta, no funciona o hace que otros componentes dejen de funcionar.

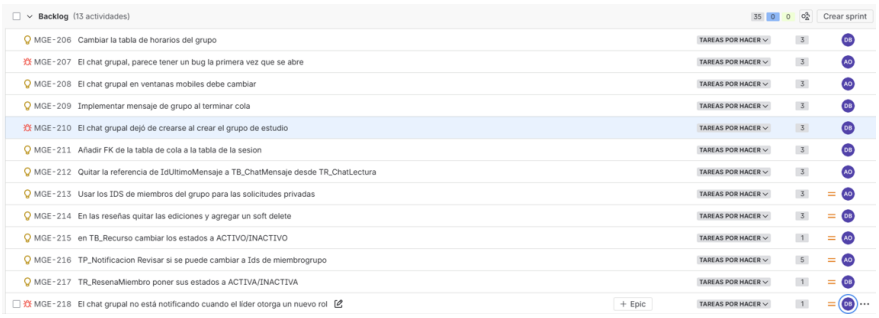
**Historia:** Historia de usuario común, usadas anteriormente.

**Mejora:** Se cataloga como mejora si no falla, pero su forma de funcionar se siente deficiente.

**BugTracker:** Reportes directos del usuario en fase beta.

Luego, poco a poco, el backlog se iba llenando de reportes (Ver Figura 14) y mejoras para el servicio, lo cual nos facilitó mucho el trabajo.

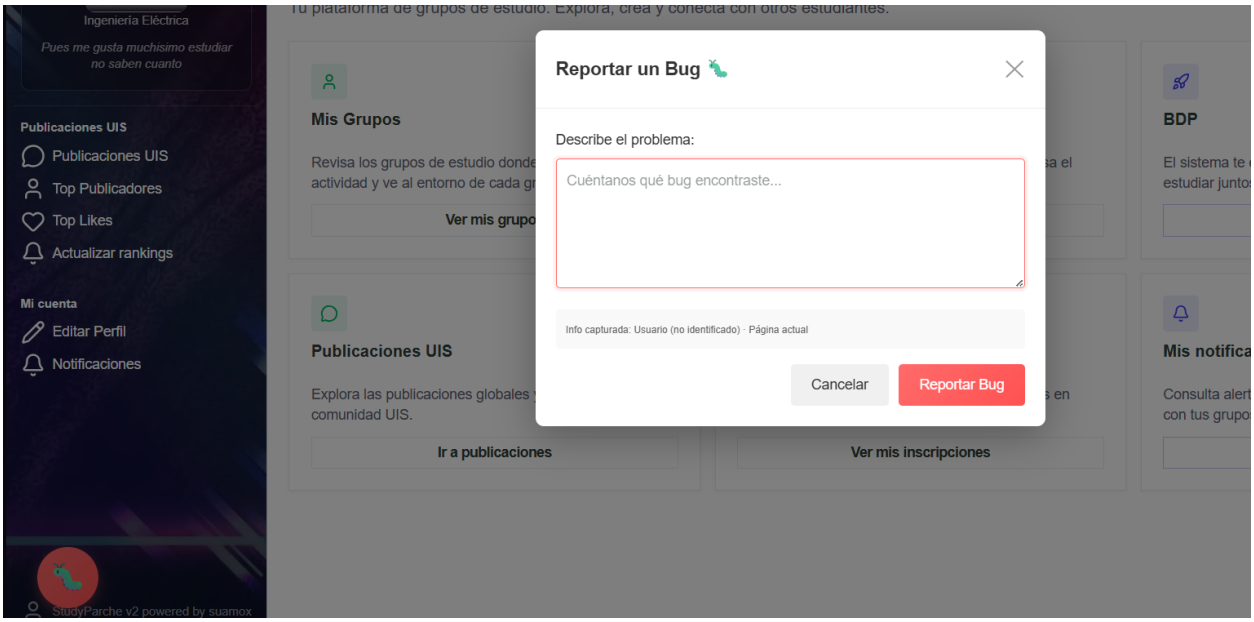
**Figura 14**
*Backlog del trabajo posterior a los sprints*




El flujo era el siguiente:

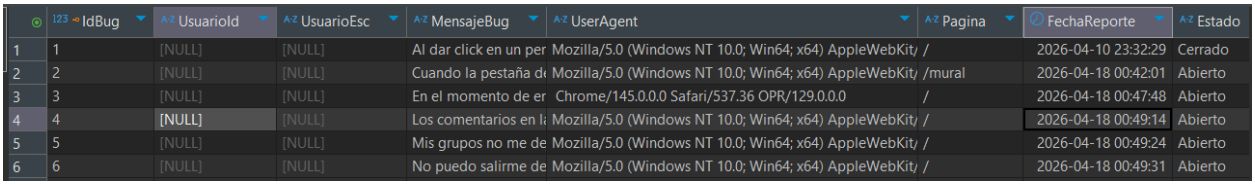
1. El usuario reporta el bug usando el BugTracker.

**Figura 15**
*Reporte desde el BugTracker*




2. La base de datos nos proporciona el reporte.

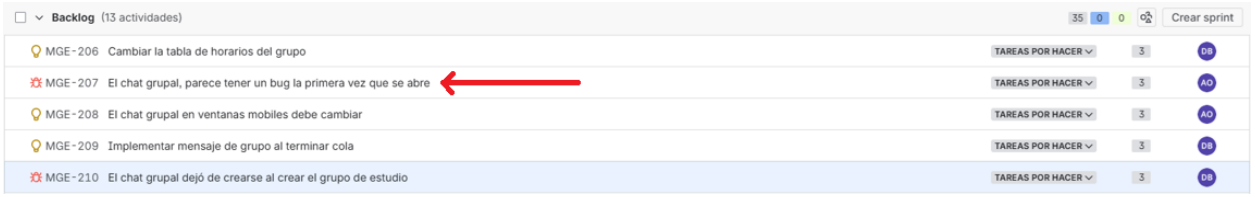
**Figura 16**
*Reportes en la base de datos*



| | IdBug | UsuarioId | UsuarioEsc | MensajeBug | UserAgent | Pagina | FechaReporte | Estado |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | [NULL] | [NULL] | Al dar click en un per | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/ | / | 2026-04-10 23:32:29 | Cerrado |
| 2 | 2 | [NULL] | [NULL] | Cuando la pestaña de | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/ | /mural | 2026-04-18 00:42:01 | Abierto |
| 3 | 3 | [NULL] | [NULL] | En el momento de er | Chrome/145.0.0.0 Safari/537.36 OPR/129.0.0.0 | / | 2026-04-18 00:47:48 | Abierto |
| 4 | 4 | [NULL] | [NULL] | Los comentarios en la | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/ | / | 2026-04-18 00:49:14 | Abierto |
| 5 | 5 | [NULL] | [NULL] | Mis grupos no me de | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/ | / | 2026-04-18 00:49:24 | Abierto |
| 6 | 6 | [NULL] | [NULL] | No puedo salirme de | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/ | / | 2026-04-18 00:49:31 | Abierto |

3. Llevamos el reporte al backlog.

**Figura 17**
*Reporte del BugTracker en JIRA*




4. Usamos el reporte para crear una nueva rama en GitHub, darle solución y finalmente hacer un merge de las ramas.

**Figura 18**
*Rama del reporte en github*

### Resumen del trabajo en JIRA

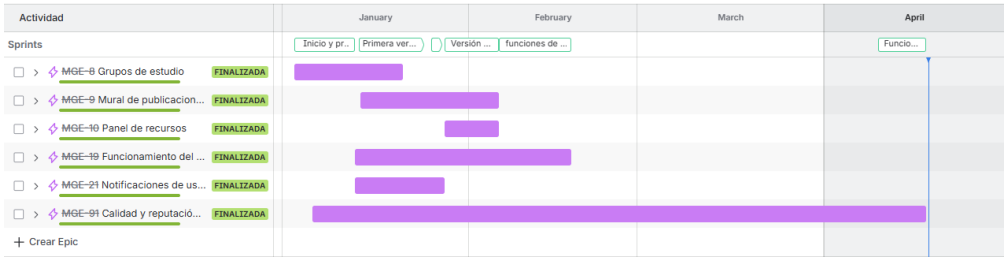
Finalmente, en JIRA se reportaron 219 tareas finalizadas (219/219), distribuidas de la siguiente manera: 10 errores, 6 reportes del BugTracker, 39 historias de usuario, 152 subtareas y 12 mejoras (Véase Apéndice B).

**Tabla 6**
*Distribución de tareas finalizadas en JIRA*

| Tipo de tarea | Cantidad |
|---|---|
| Errores | 10 |
| Reportes del BugTracker | 6 |
| Historias de usuario | 39 |
| Subtareas | 152 |
| Mejoras | 12 |
| **Total** | **219** |

Finalmente, todas las épicas fueron completadas, y el trabajo usando el marco de trabajo SCRUM dio buenos resultados. Se logró aprender mucha organización base sobre la documentación de incidencias y cómo ajustar los tiempos de un desarrollador. También logramos entender más sobre la medición de la complejidad de las implementaciones y a darle validez e importancia a desglosar tareas en subtareas más pequeñas. En la imagen se ve el calendario final de los sprints y las épicas finalizadas en Jira en su respectivo tiempo.

**Figura 19**
*Estado final de las épicas en JIRA*



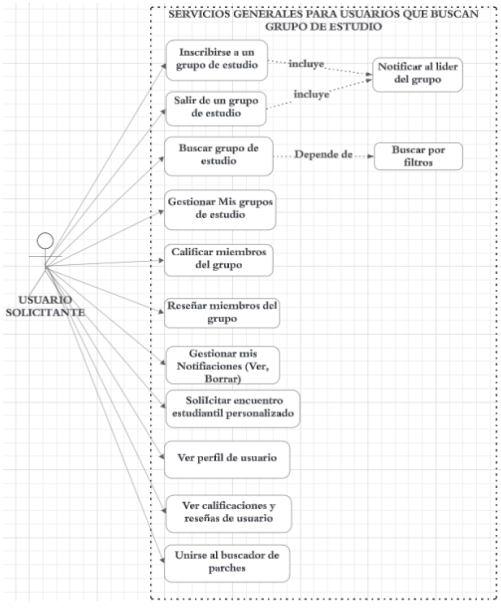
# 5. Diseño

## 5.1 Diagramas de caso de uso

Aquí vamos a ver los diagramas de caso de uso, que básicamente muestran cómo se comporta el sistema dependiendo de quién lo use. En cada situación participa uno o dos actores, por ejemplo el solicitante o el líder, y se explica qué puede hacer cada uno dentro de este entorno. Luego se detalla el flujo principal, y también los flujos alternativos. Al final se listan las precondiciones (lo que tiene que pasar antes) y las postcondiciones (lo que el sistema garantiza después de ejecutar la acción).

### *5.1.1 Sistema general para usuario solicitante*

**Figura 20**
*Sistema general para usuario solicitante*




***Flujo general:*** El flujo general inicia cuando el usuario solicitante accede al módulo de grupos de estudio. Desde este módulo, el usuario puede realizar la búsqueda de grupos mediante una interfaz que permite aplicar filtros por tema, carrera, modalidad o nivel de conocimiento. Los

resultados de búsqueda se muestran mediante tarjetas informativas que incluyen descripción del grupo y opciones para solicitar ingreso.

Si el grupo es público, el usuario puede unirse directamente mediante el botón “Unirse”. Si el grupo es privado, el usuario debe enviar una solicitud de ingreso. Esta solicitud genera una notificación que podrá ser aceptada o rechazada por el líder del grupo o sus moderadores.

Una vez que el usuario es miembro de uno o más grupos, podrá visualizar los mismos en la sección “Mis grupos”, donde se muestran las tarjetas de cada grupo con su nombre, líder, horarios de reunión e imagen representativa. Desde esta vista, el usuario puede ingresar al grupo o abandonarlo mediante la opción “Dejar grupo”.

Además, el usuario solicitante puede calificar al grupo y al líder. La calificación al grupo se realiza desde el panel principal del grupo, mientras que la calificación al líder se realiza desde su perfil, mediante el botón “Calificar”.

El usuario también dispone de un panel de notificaciones en el cual puede consultar solicitudes enviadas y recibidas, respuestas a solicitudes pendientes y otros eventos relacionados con los grupos de estudio. Estas notificaciones pueden visualizarse en detalle y eliminarse cuando sea necesario.

Finalmente, el usuario solicitante puede utilizar la opción de “búsqueda rápida grupal (LFR)”, en la cual registra un tema de estudio y disponibilidad horaria. El sistema empareja automáticamente hasta tres estudiantes con intereses similares y asigna un lugar y hora de encuentro dentro de la universidad.

***Precondiciones:***

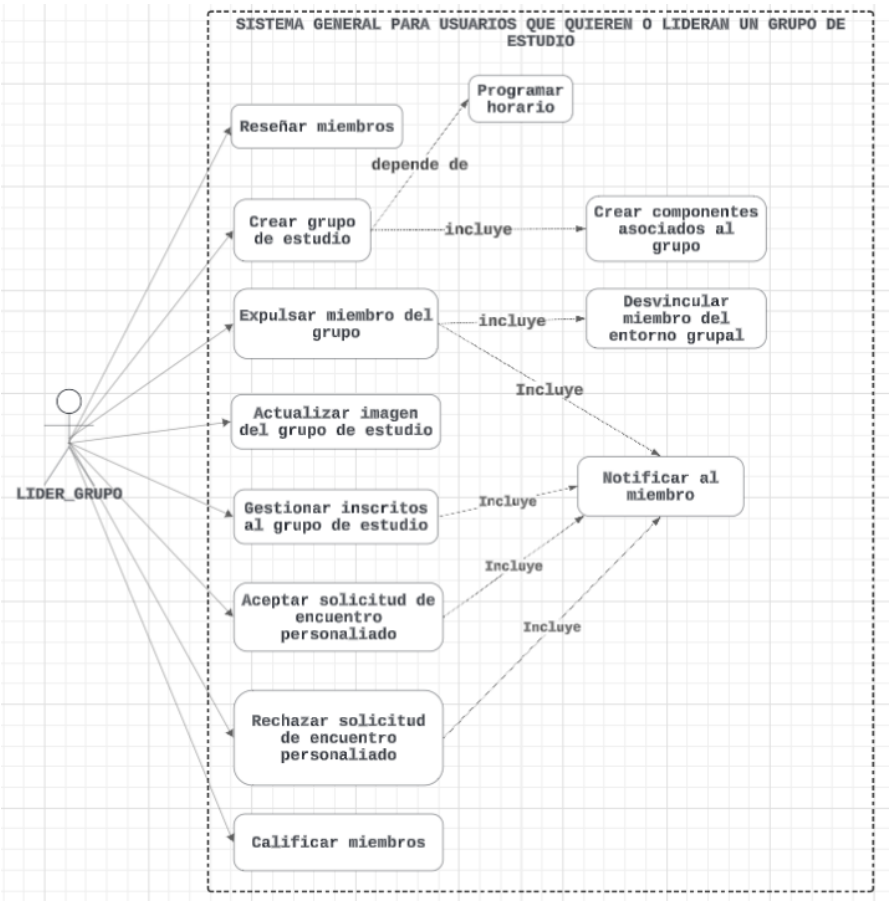
- El usuario debe estar autenticado en la plataforma.
- El usuario debe tener rol de miembro interesado o miembro activo.
- Deben existir grupos de estudio registrados en el sistema.
- Para unirse a un grupo privado, el usuario no debe haber enviado previamente una solicitud pendiente al mismo grupo.

***Postcondiciones:***

- El usuario queda registrado como miembro del grupo (si fue aceptado o se unió a uno público).
- Se genera o actualiza el estado de la solicitud de ingreso.

### *5.1.2 Sistema general para líder de grupo*

**Figura 21**
*Sistema general para lider de grupo*




***Flujo general:***

El flujo general inicia cuando el usuario aspirante a líder ingresa al módulo de grupos de estudio. Desde este módulo, el usuario puede seleccionar el botón “crear grupo”, el cual lo redirige

a un formulario donde debe completar los campos asociados al grupo, tales como el nombre del grupo, la carga de una imagen representativa, una descripción, el nombre de la materia a la que pertenece y la configuración de los horarios de reunión semanal, así como la selección del tipo de grupo (público o privado).

Una vez que el usuario se convierte en líder de un grupo, puede ingresar al grupo específico y, desde la sección de chat, gestionar a los miembros del grupo mediante la opción “expulsar” ubicada en la columna de miembros. Esta acción genera una notificación que puede ser visualizada desde el panel de notificaciones del usuario expulsado.

El usuario líder del grupo puede actualizar la imagen del grupo de estudio las veces que lo desee. Asimismo, puede gestionar los miembros del grupo en caso de que este sea de tipo privado. Estas acciones generan notificaciones que pueden ser visualizadas desde el panel de notificaciones de los usuarios aceptados como miembros.

En caso de que el usuario líder del grupo reciba una solicitud de encuentro personalizado, puede aceptarla o rechazarla desde el panel de solicitudes, mediante la selección del botón correspondiente de color verde o rojo. En ambos casos, el sistema notificará al miembro que realizó la solicitud a través del panel de notificaciones.

El usuario líder puede calificar a los miembros de su grupo con etiquetas propuestas por el sistema, mediante una opción en el panel de perfil del miembro seleccionado.

***Precondiciones:***

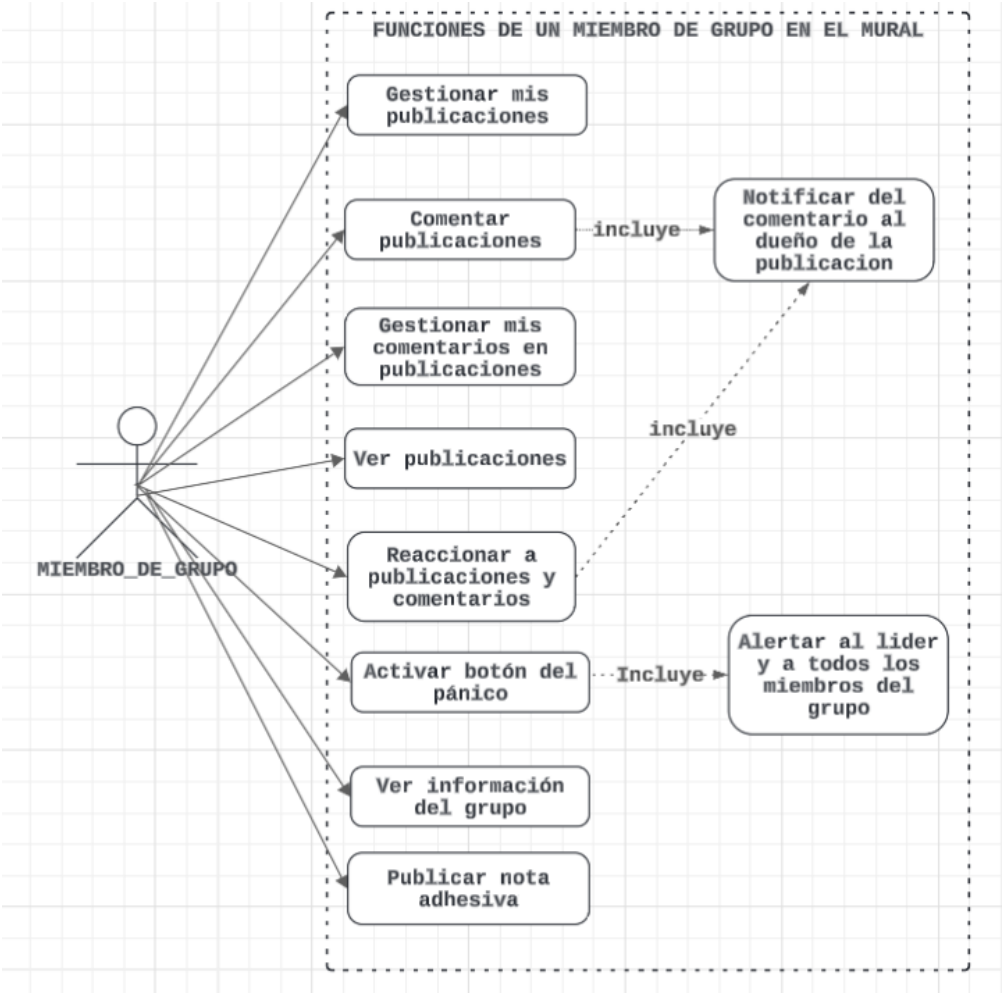
- El usuario aspirante a líder debe estar autenticado en la plataforma.

***Postcondiciones:***

- El usuario queda registrado como líder del grupo de estudio.
- El grupo queda registrado en el sistema como grupo de estudio activo.

### *5.1.3 Funciones de un miembro de grupo en el mural*

**Figura 22**
*Casos de uso de los miembros en el mural*




***Flujo general:***

El flujo general inicia cuando el miembro ingresa al módulo de grupos de estudio, en la sección de mural. Desde este módulo, el miembro puede gestionar sus publicaciones mediante la opción “Mis publicaciones” ubicada en la barra superior, lo que le permite eliminar aquellas que ya no desea o añadir nuevas publicaciones. Al ingresar a esta sección, se visualiza la vista general de las publicaciones del grupo junto con las publicaciones propias del miembro.

Dentro del mural, al seleccionar una publicación mediante la tarjeta correspondiente, el

miembro accede a la sección de comentarios ubicada en la parte inferior de la página. Al seleccionar la opción “Comentar”, el sistema permite escribir un texto de hasta 200 caracteres. Asimismo, el miembro puede reaccionar a la publicación mediante las opciones “Me gusta” o “No me gusta”, de acuerdo con su preferencia, así como reaccionar a otros comentarios utilizando las mismas opciones.

Estas acciones generan una notificación que es enviada al panel de notificaciones del miembro autor de la publicación o del comentario, según corresponda.

Los miembros del grupo también podrán publicar notas en el mural como parte de una comunicación asíncrona y ver al detalle la información del grupo en sus respectivas opciones dentro del mural.

***Precondiciones:***

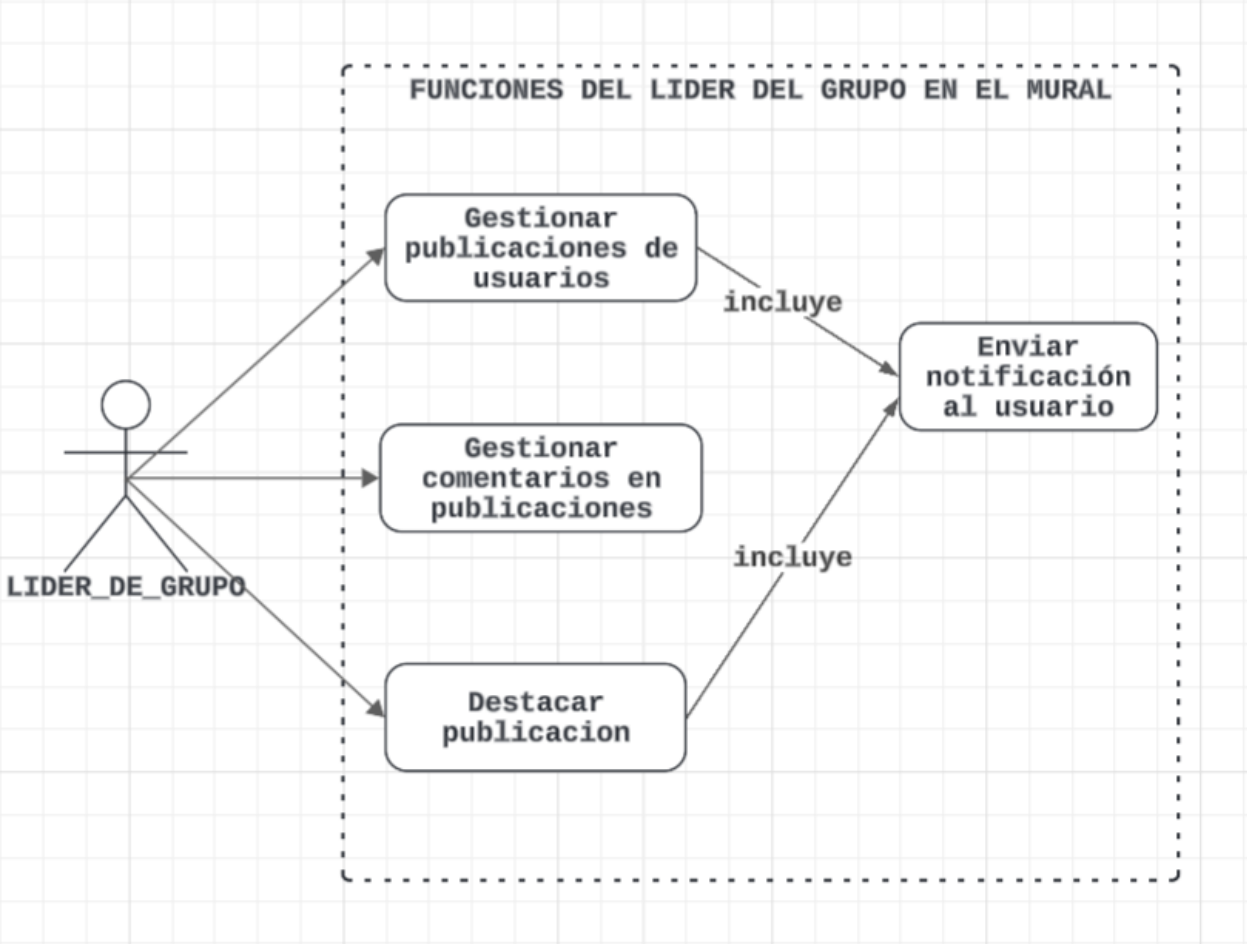
- El usuario debe estar autenticado en la plataforma.
- El usuario debe pertenecer al grupo.
- El usuario pertenece a un rol específico.
- El usuario no se encuentra vetado.
- El usuario no se encuentra silenciado.

***Postcondiciones:***

- Las publicaciones, comentarios y reacciones quedan registradas en el sistema.
- El sistema actualiza el mural con las publicaciones añadidas más recientes.
- El sistema notifica al autor si alguien reacciona o comenta su publicación.
- El sistema notifica en tiempo real al autor cuando un miembro reacciona o responde a su comentario y persiste la acción en base de datos.

### *5.1.4 Funciones del líder en el mural*

**Figura 23**
*Funciones del líder en el mural*




***Flujo general:***

El flujo general inicia cuando el líder del grupo ingresa al mural. Desde la vista principal, puede gestionar las publicaciones seleccionando la opción de los tres puntos ubicada en la parte superior derecha de la tarjeta de cada publicación, donde se despliegan las opciones disponibles, entre ellas eliminar la publicación, así como gestionar los comentarios asociados.

El usuario líder del grupo puede destacar publicaciones del mural mediante la opción *"Destacar"*, disponible en el menú de los tres puntos ubicado en la parte superior de la tarjeta de la

publicación.

Cualquiera de estas acciones genera una notificación que es enviada al miembro autor de la publicación o del comentario, según corresponda.

***Precondiciones:***

- El usuario debe estar autenticado en la plataforma.
- El usuario debe pertenecer al grupo y contar con el rol de líder.

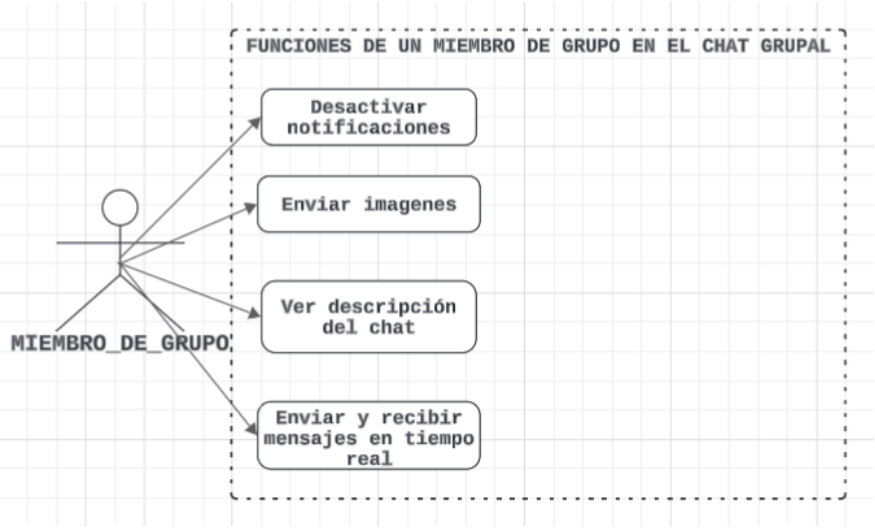
***Postcondiciones:***

- Las publicaciones destacadas quedan marcadas como destacadas en el sistema.
- Se actualizan las publicaciones o comentarios gestionados en el mural del grupo.
- El sistema envía la notificación correspondiente al autor de la publicación o comentario.
- El sistema actualiza y muestra correctamente la cantidad de comentarios y reacciones asociadas a la publicación.

### *5.1.5 Funciones de un miembro en el chat grupal*

**Figura 24**

*Funciones de un miembro en el chat grupal*



***Flujo general:***

El flujo general inicia cuando un miembro del grupo ingresa al chat grupal, donde puede enviar y recibir mensajes en tiempo real. Cada mensaje enviado genera una notificación sonora que es recibida por los demás miembros del grupo.

El miembro del grupo puede enviar mensajes de texto, imágenes e incluso audios. Si necesita mas información sobre las normas del chat puede ir al icono d información para ver la descripción y los participantes.

***Precondiciones:***

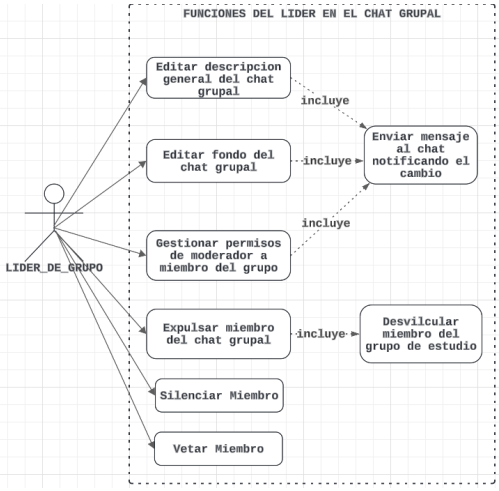
- El usuario debe estar autenticado en la plataforma.
- El usuario debe pertenecer al grupo de estudio.

***Postcondiciones:***

- Los mensajes enviados quedan registrados y actualizados en el sistema.
- El sistema envía las notificaciones correspondientes a los mensajes y alertas generadas.

### *5.1.6 Funciones del líder en el chat grupal*

**Figura 25**
*Funciones del líder en el chat grupal*



***Flujo general:***

El flujo general inicia cuando el líder del grupo ingresa al chat grupal. Desde este espacio, puede editar la descripción del chat, modificar el tono del fondo y gestionar los permisos de los demás miembros. Estas acciones generan notificaciones que son enviadas a los miembros del chat.

El líder del grupo puede expulsar miembros del chat grupal, acción que desvincula al miembro tanto del chat como del grupo de estudio.

***Precondiciones:***

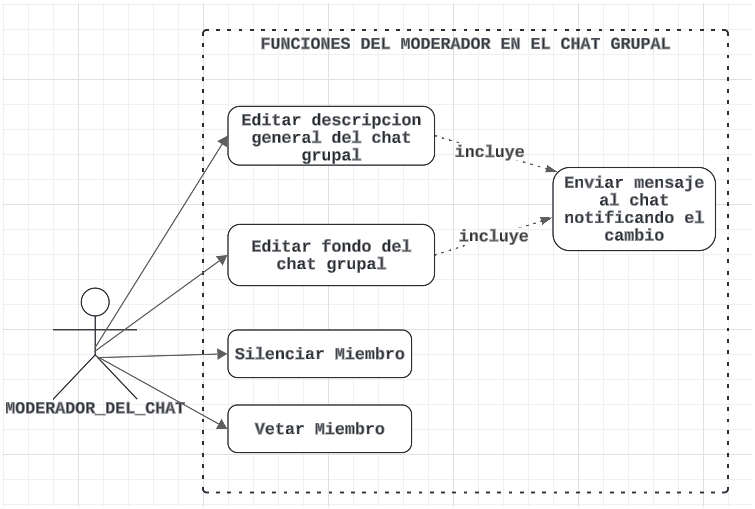
- El usuario debe estar autenticado en la plataforma.
- El usuario debe pertenecer al grupo de estudio con el rol de líder.

***Postcondiciones:***

- Los cambios realizados en la configuración del chat quedan actualizados en el sistema.
- Los estados de los miembros del chat quedan modificados.
- Los miembros expulsados quedan desvinculados del chat grupal y del grupo de estudio.

### *5.1.7 Funciones del Moderador en el chat grupal*

**Figura 26**
*Funciones del Moderador en el chat grupal*



***Flujo general:***

El flujo general inicia cuando el moderador ingresa al grupo en la sección de chat. Desde este espacio, puede editar el fondo del chat y la descripción del mismo. Estas acciones generan una notificación visible para los miembros del chat informando sobre los cambios realizados.

Por otro lado, el moderador puede expulsar miembros del chat grupal, acción que desvincula al miembro tanto del chat como del grupo de estudio.

***Precondiciones:***

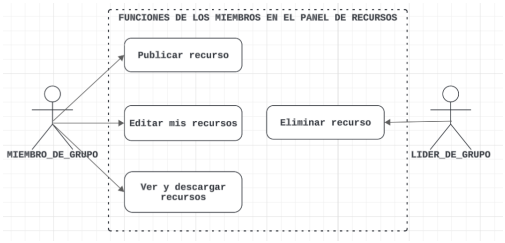
- El grupo de estudio debe encontrarse activo.
- El moderador debe tener permisos habilitados para la gestión del chat.
- El usuario debe estar autenticado en la plataforma.
- El usuario debe pertenecer al grupo de estudio con el rol de moderador.

***Postcondiciones:***

- Los cambios realizados en la configuración del chat quedan actualizados en el sistema.
- Los miembros expulsados quedan desvinculados del chat grupal y del grupo de estudio.
- El sistema registra las acciones realizadas por el moderador.
- El sistema envía notificaciones a los miembros del chat sobre las modificaciones realizadas.

### *5.1.8 Funciones de los miembros en el panel de recursos*

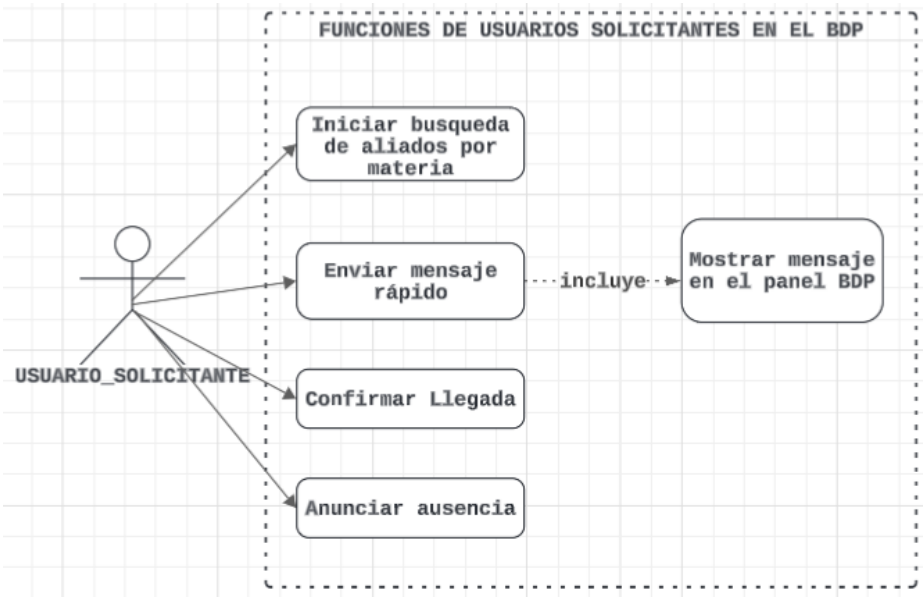
**Figura 27**
*Funciones del líder en el chat grupal*



***Flujo general:*** El flujo incia cuando el miembro del grupo ingresa al grupo en el panel de recursos, puede publicar, editar y ver o descargar los recursos. El miembro del grupo con el rol de líder puede eliminar recursos del mural.

### *5.1.9 Funciones de un usuario solicitante en el BDP*

**Figura 28**
*Funciones de un usuario solicitante en el BDP*




***Flujo general:***

El flujo inicia cuando un usuario solicitante que quiere un grupo de estudio presencial rápido accede al botón del BDP, donde puede elegir la materia y ponerse en una cola que esperará por otros dos usuarios con su mismo objetivo. Una vez que se encuentren, irán a un nuevo panel donde podrán enviar mensajes rápidos para comunicar su llegada o su ausencia.

## 5.2 Diagrama de clases

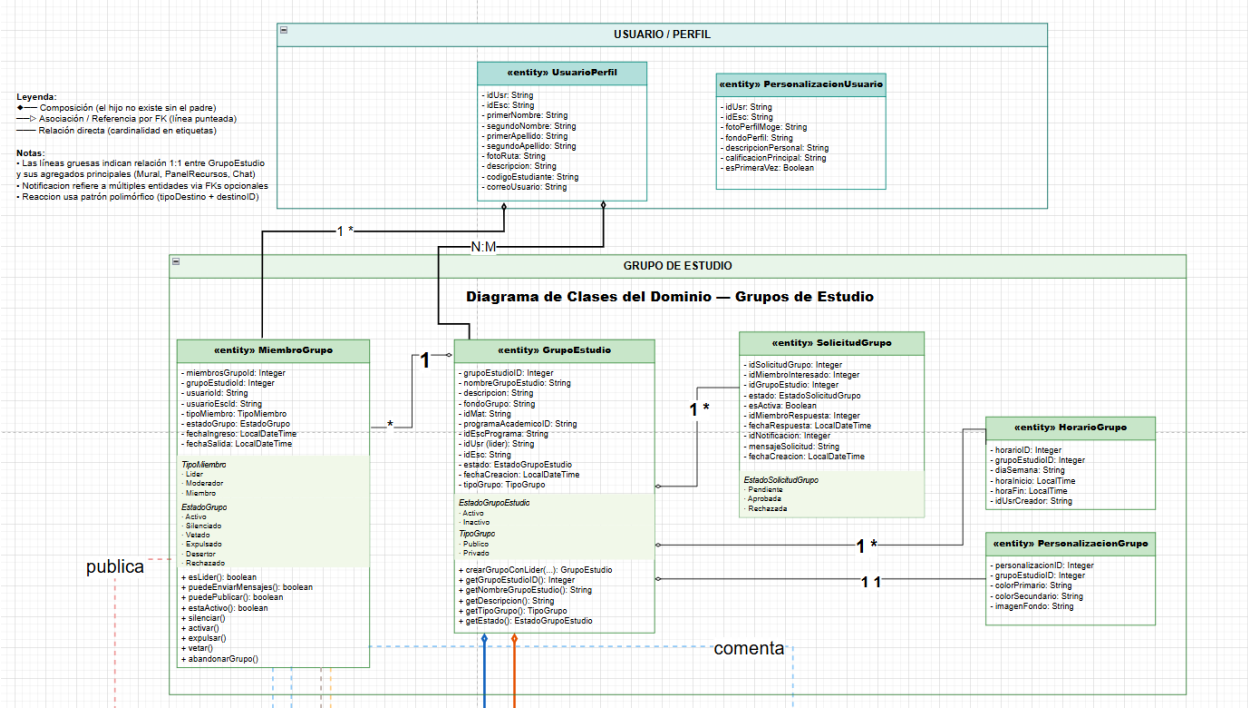
El diagrama de clase completo es muy grande y tiene muchas entidades, por lo que no cabía en una sola imagen sin perder calidad visual. Para una correcta visualización, se optó por separarlo por dominios: primero usuario/perfil con sus tablas conectadas, luego grupo de estudio,

y así sucesivamente, mostrando poco a poco las entidades y relaciones más importantes.

### *5.2.1 Dominio de Usuario/Perfil y Grupo de estudio*

**Figura 29**

*Diagrama de clases para el dominio de Usuarios y Grupos de estudio*

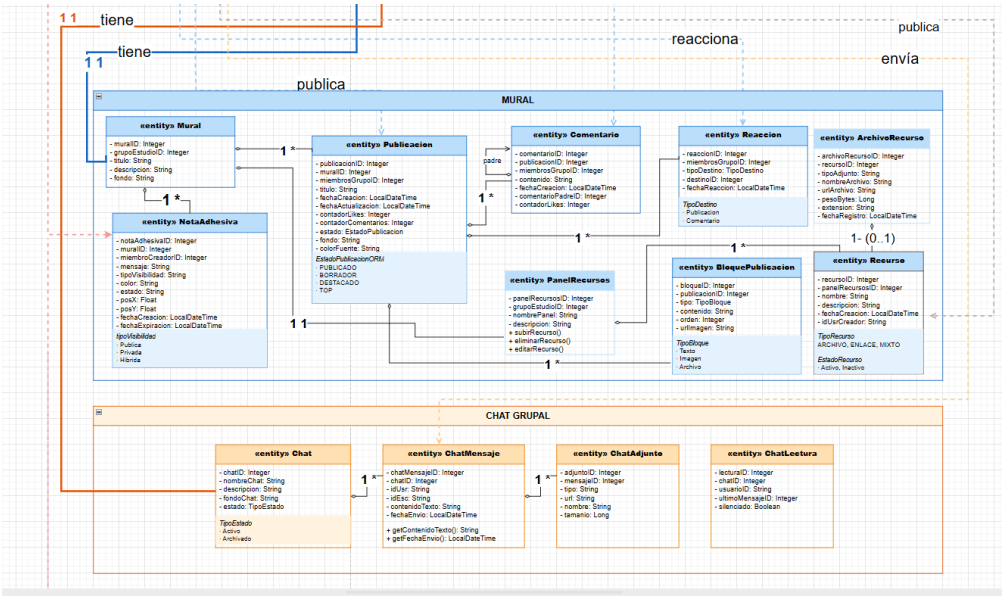



### *5.2.2 Dominio del Mural y el chat grupal*

Notese que hay lineas de color que indican que la entidad señalada es padre de las entidades donde empieza la flecha, es decir, no pueden vivir sin su padre y por ende su relación es 1:1. Además se muestran las relaciones entre clases que conllevan una acción y la clase padre que la genera, estas son representadas por lineas puntuadas. Algo interesante que podrá ver es la tabla de comentario, la cual se referencia a si misma, esto es debido a que los comentarios son padres de otros comentarios.

**Figura 30**

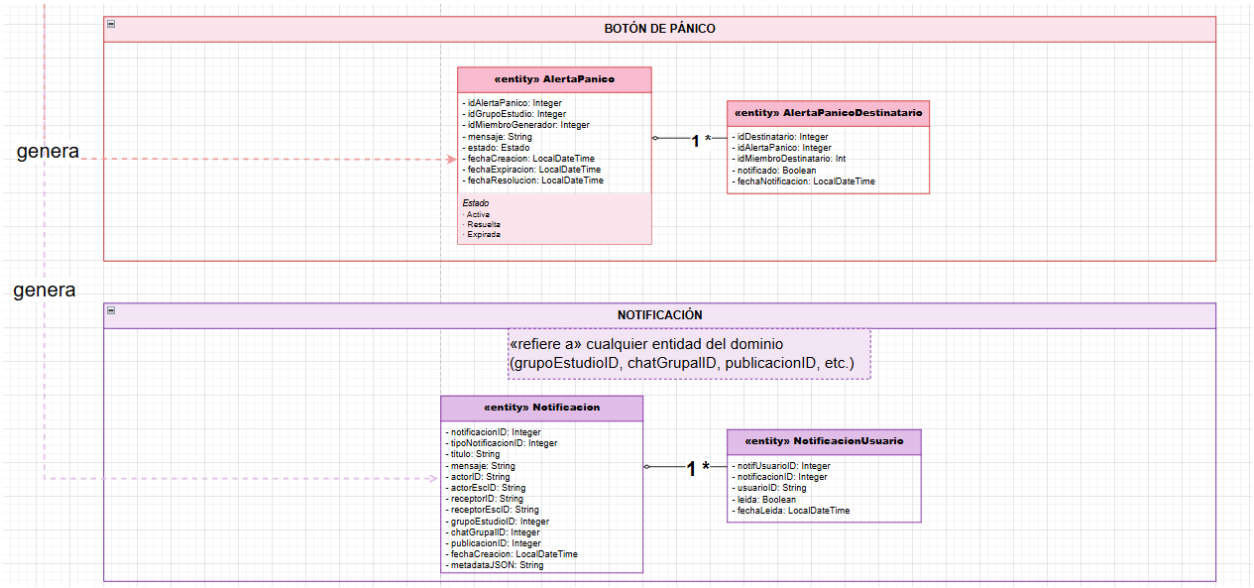
*Diagrama de clases para el dominio del Mural y Chat grupal*




### *5.2.3 Dominio del Botón de pánico y Notificación*

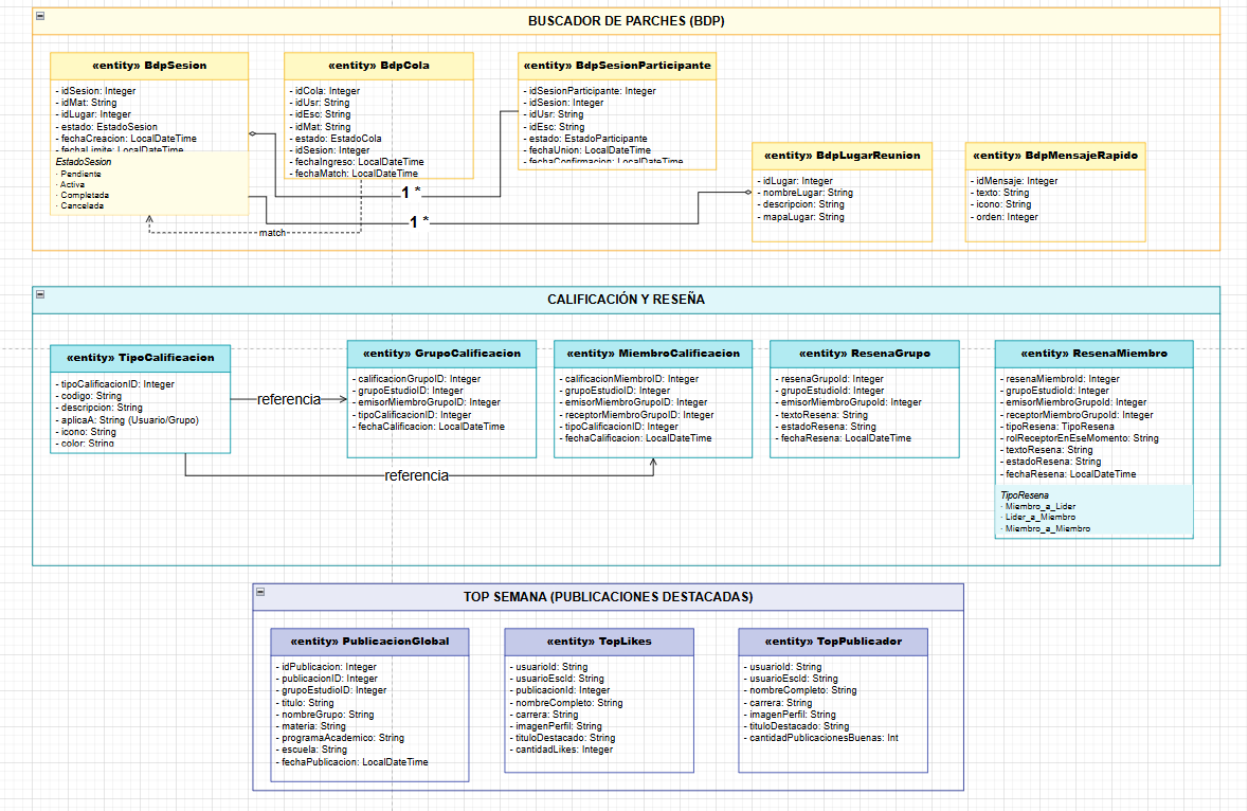
**Figura 31**

*Digrama de clases para el dominio del Botón de pánico y Notificación*



### *5.2.4 Dominio BDP, Calificaciones, reseñas y tops semanales*

**Figura 32**
*Digrama de clases para el dominio del BDP, Calificaciones, reseñas y tops semanales*



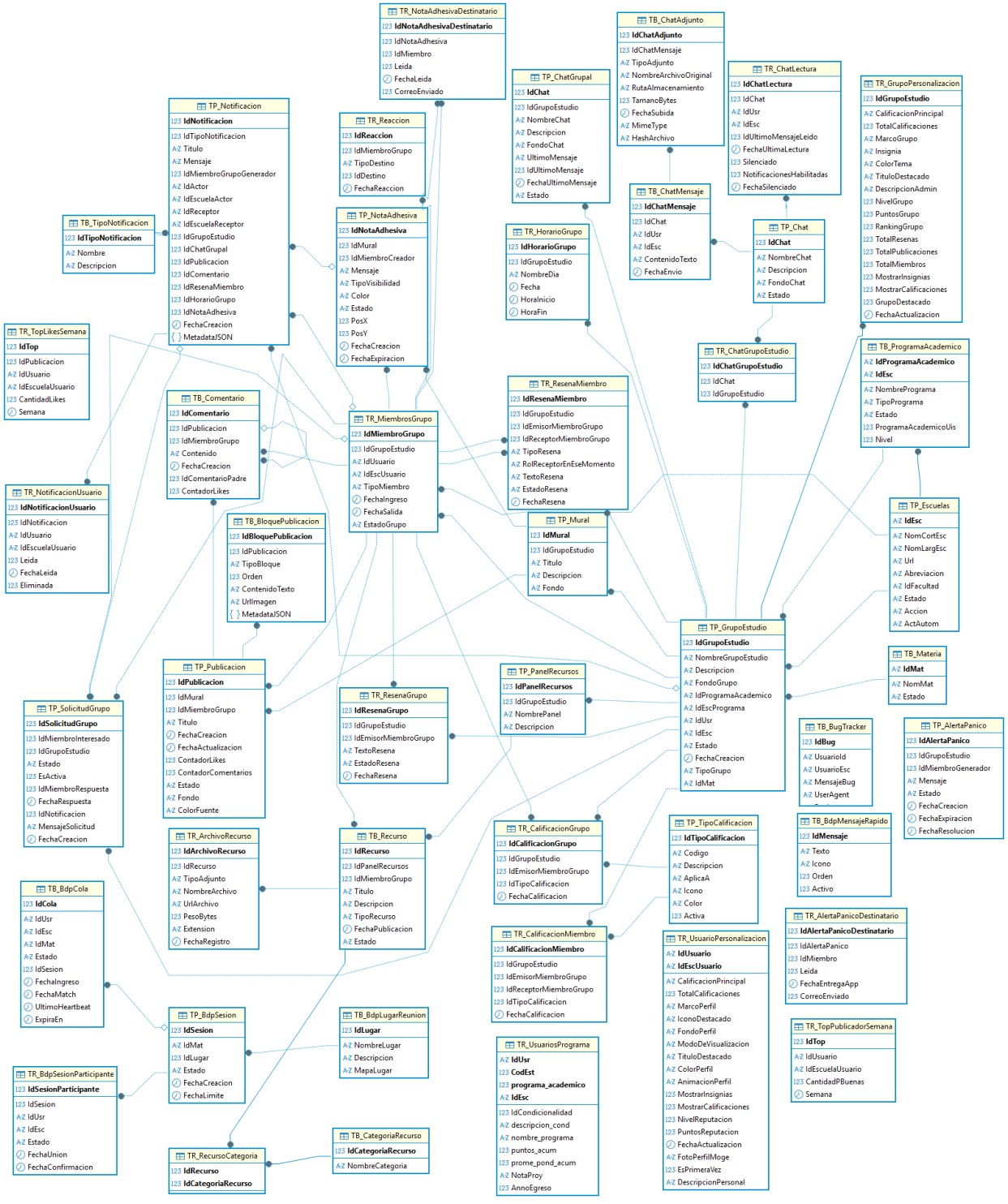

## 5.3 Modelo entidad relación físico

Este modelo Entidad–Relación fue construido a partir de las tablas existentes en la base de datos de COMA, entre ellas TP_Usuarios y TP_Materias. Estas tablas corresponden a un módulo antiguo y utilizan un motor de almacenamiento que no soporta la definición de llaves foráneas, razón por la cual no se reflejan en el MER.

Otra de las normas sugeridas fue la de respetar la integridad referencial relacionado a las tablas compartidas entre ambos sistemas. Finalmente se obtienen más de 40 tablas listas que representan el flujo funcional del sistema.

**Figura 33**

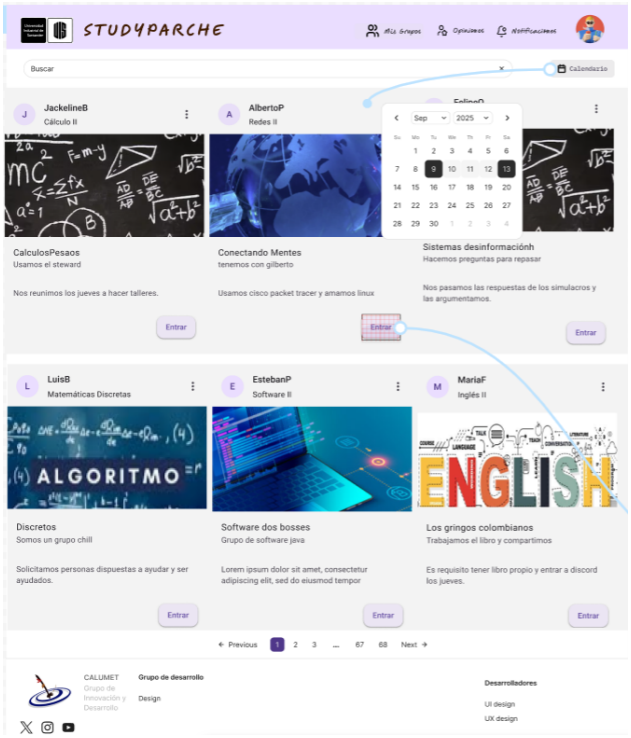
*Modelo ENTIDAD-RELACIÓN MySQL*



## 5.4 Mockups

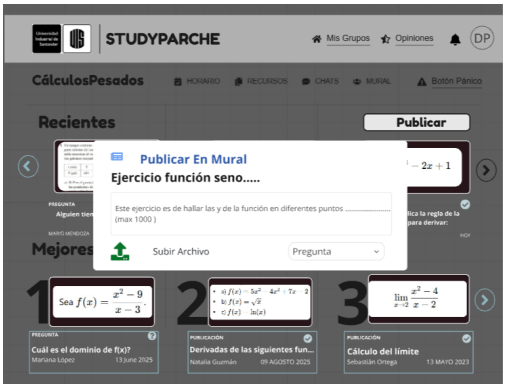
### *5.4.1 Vista de los grupos del usuario*

**Figura 34**
*Vista de grupos del usuario*



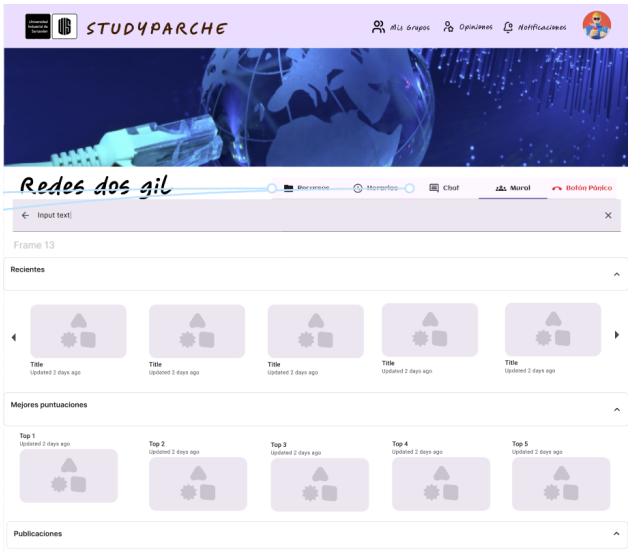

### *5.4.2 Vista del proceso de publicación*

**Figura 35**
*Vista botón publicación Mural del grupo*



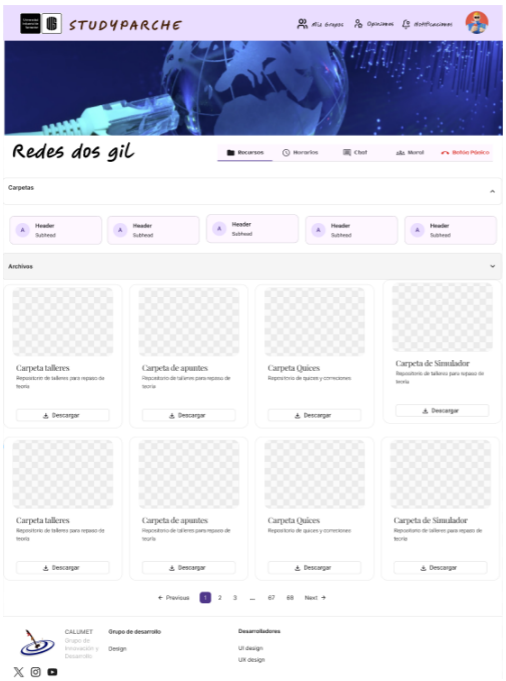
### *5.4.3 Vista del mural del grupo*

**Figura 36**
*Panel del mural del grupo*



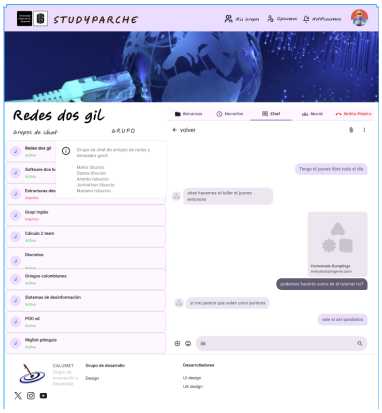

### *5.4.4 Vista del panel de recursos*

**Figura 37**
*Vista de recursos dentro de un grupo*



### *5.4.5 Vista del panel de chats y miembros de un grupo de estudio*

**Figura 38**
*Vista de chats y miembros dentro de un grupo*




## 5.5 Diagrama de arquitectura

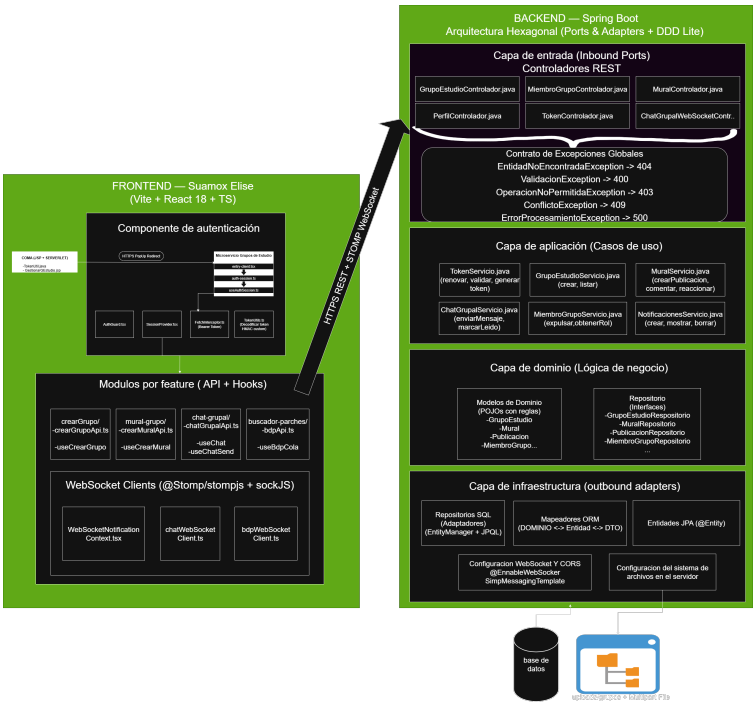
**Figura 39**
*Diagrama de arquitectura*

Para el diagrama de arquitectura, buscamos mostrar de forma general cómo se comunicarán las capas entre sí con algunos componentes, dando a entender la arquitectura que se aplicará, pero sin perder tecnicidad. El diagrama de arquitectura debe mantenerse entre lo técnico y lo informal, porque es un abrebocas tanto para quien compra un software como para quien lo vende, y esta persona puede ser un experto en software o simplemente alguien que no sabe del tema.

Como señala Brown, 2018, ”el objetivo debe ser mantener los diagramas simples y fáciles de entender en cuanto a cómo se relacionan las cosas. Que las discusiones y otros documentos proporcionen el detalle más profundo”.

## 5.6 Diagrama de componentes

**Figura 40**
*Diagrama de arquitectura de componentes*




A diferencia del diagrama de arquitectura, el diagrama de componentes es una herramienta más técnica y formal, orientada al desarrollador. Aquí se muestran interfaces, puertos, dependencias y la comunicación real entre componentes. El primero era un abrebocas para todos; este ya va

al grano para el que va a programar.

En conclusión, el diagrama de arquitectura responde al ”quéz al ”por qué”de la comunicación entre capas, mientras que el diagrama de componentes responde al çómoz al çon qué”se conecta cada pieza desde el código. Por otro lado, los diagramas de caso de uso nos ayudaron a comprender cómo interactuará el usuario final con nuestros servicios. Los mockups le dieron una cara visible al sistema. Finalmente, los diagramas de entidad-relación y de clases definen la estructura estática de los datos y sus relaciones, sentando las bases lógicas del sistema y dando paso a la implementación.

# 6. Implementación

## 6.1 Estándares de ingeniería utilizados

Para el desarrollo del microservicio de grupos de estudio se adoptaron buenas prácticas de ingeniería de software orientadas a garantizar mantenibilidad, escalabilidad y claridad arquitectónica. Entre los principales estándares aplicados se encuentran:

- **Principios SOLID**: Estos principios se aplicaron siempre que fue posible y no entraran en contradicción con otras normas del proyecto. Por ejemplo, se cumple el principio de segregación de interfaces (ISP) solo en las clases que lo requieren, debido a que una de las normas del proyecto es evitar el *boilerplate*. La forma en que se aplicaron estos principios es visible en el código. A continuación, se presenta un análisis de cada principio utilizando como ejemplo el proceso de solicitud de ingreso a grupos privados.

    1. **Principio de responsabilidad única (SRP)**: La clase `SolicitudGrupoServicio` tiene una única responsabilidad: gestionar las operaciones relacionadas con las solicitudes de grupo. No maneja la persistencia directamente (delegada a los repositorios) ni las notificaciones (delegada a `NotificacionServicio`).
    2. **Principio de abierto/cerrado (OCP)**: La clase está abierta para su extensión, pero cerrada para la modificación. Es posible agregar nuevos tipos de operaciones sobre

solicitudes sin modificar el código existente.

3. **Principio de sustitución de Liskov (LSP)**: Este principio se aplica mediante el uso de interfaces como `SolicitudGrupoRepositorio`, lo que permite que cualquier implementación que cumpla con la interfaz pueda ser sustituida sin afectar el comportamiento del sistema.

4. **Principio de segregación de interfaces (ISP)**: La interfaz *SolicitudGrupoRepositorio* define métodos específicos y cohesivos, relacionados únicamente con las operaciones del repositorio de solicitudes, sin incluir métodos innecesarios.

5. **Principio de inversión de dependencias (DIP)**: La clase *SolicitudGrupoServicio* depende de abstracciones (interfaces) como *SolicitudGrupoRepositorio*, *GrupoEstudioRepositorio*, *MiembroGrupoServicio* y *NotificacionServicio*, en lugar de depender de implementaciones concretas. Las dependencias se inyectan a través del constructor.

- **Diseño RESTful para servicios web**: Se definieron *endpoints* siguiendo los verbos HTTP estándar (GET, POST, PUT, DELETE), URLs basadas en recursos (ejemplo: `/{solicitudId}/aceptar/lider/{liderId}/escuela/{liderEscId}`) y uso de códigos de estado HTTP apropiados. En el caso de las solicitudes de ingreso a grupos, se implementó un método `aceptarSolicitud` anotado con `@PostMapping` que recibe el identificador de la solicitud, del líder y de su escuela mediante `@PathVariable`, delega la lógica de negocio en `SolicitudGrupoServicio` y retorna una respuesta HTTP 200 con el objeto DTO correspondiente.

- **Manejo de códigos HTTP**: Se utilizó una clase `GlobalExceptionHandler.java` para el manejo de los códigos de error HTTP. Por ejemplo, se definió la excepción personalizada `ValidacionException` (ubicada en el paquete `org.calumet.gruposestudio.shared.er` la cual es lanzada cuando los datos de entrada son inválidos o faltan campos requeridos; el manejador global la traduce automáticamente a una respuesta HTTP 400 (Bad Request). El paquete `org.calumet.gruposestudio` fue una nomenclatura obligatoria, establecida

como normativa por el grupo de investigación durante el desarrollo.

Estos estándares permiten desacoplar las capas del sistema (aspecto necesario en la arquitectura DDD Lite + Hexagonal), facilitar el reemplazo de componentes y favorecer la reutilización del código. El lector podrá encontrar el código completo en los Apéndices C a E.

## 6.2 Selección de herramientas

La selección de herramientas se realizó considerando compatibilidad institucional, experiencia del equipo de desarrollo y facilidad de mantenimiento a largo plazo.

### *6.2.1 Manejo de base de datos: MySQL*

La base de datos utilizada corresponde a la empleada actualmente por el sistema institucional COMA. Por este motivo, el microservicio se integró en el mismo gestor MySQL, garantizando compatibilidad y la posibilidad de establecer relaciones con tablas preexistentes como usuarios, programas académicos y escuelas.

El uso de MySQL facilita:

- Integridad referencial mediante llaves foráneas
- Consultas relacionales complejas
- Reutilización de infraestructura institucional existente
- Mantenibilidad en el grupo de investigación y coherencia

De esta forma fue posible añadir nuevas tablas relacionadas como grupos de estudio, notificaciones, reseñas y chat, sin modificar la estructura base del sistema COMA.

Inicialmente se utilizó una base de datos independiente de la plataforma, llamada AGORA, donde se alojaron todas las tablas mientras se realizaban pruebas para cumplir con el objetivo planteado. Dado que un microservicio debe ser independiente y capaz de vivir sin necesidad del servicio padre, y viceversa, se ampliará esta información en el capítulo de despliegue.

### *6.2.2 Entornos de programación: IntelliJ IDEA, NetBeans y Visual Studio Code*

Para el desarrollo del proyecto se hizo uso de diferentes entornos de desarrollo integrados (IDE), seleccionados de acuerdo con la naturaleza de cada componente:

- **IntelliJ IDEA Community Edition**: utilizado para el desarrollo del microservicio backend en Spring Boot debido a su soporte avanzado para proyectos Java.
- **NetBeans**: empleado para la integración del módulo con la plataforma COMA institucional, siguiendo las preferencias del director de proyecto y facilitando la explicación del código.
- **Visual Studio Code**: utilizado para el desarrollo del frontend del microservicio, debido a su amplio ecosistema de extensiones y herramientas como Live Server.

### *6.2.3 Framework utilizado: Spring Boot*

El microservicio fue implementado utilizando el framework Spring Boot, debido a que esta estipulado en el documento normativo para el desarrollo de microservicios nuevos en CALUMET, pero ademas este framework ofrece:

- Soporte nativo para microservicios
- Integración con Maven para gestión de dependencias
- Facilidad para construir APIs REST
- Compatibilidad con WebSocket
- Buen acoplamiento con arquitectura hexagonal (importante)

### *6.2.4 Lenguaje backend: Java*

El lenguaje utilizado para el backend es Java, debido al dominio del equipo de desarrollo y su compatibilidad con la infraestructura institucional. Entre otras ventajas del lenguaje podemos encontrar:

- Alto rendimiento y estabilidad
- Gran soporte en entornos académicos
- Comunidad amplia y bibliotecas maduras
- Compatibilidad con el grupo de investigación (Servicio principal usa JSP)

### *6.2.5 Lenguajes frontend: Herramienta interna de CALUMET Suamox-Elise*

En el plan inicial se tenía pensado utilizar únicamente HTML, CSS y JavaScript para el frontend. Sin embargo, una vez que el plan fue presentado y se dio inicio al desarrollo, una herramienta interna del grupo fue seleccionada como la única opción para el desarrollo frontend en Calumet. Para ese momento ya existía una versión mínima adelantada con las herramientas del plan. No obstante, al tratarse de un grupo de investigación, se considera óptimo mantener un contrato de herramientas estandarizadas, de modo que los nuevos integrantes no tengan que aprender múltiples tecnologías y puedan continuar con el legado establecido. Por esta razón, se decidió migrar hacia la herramienta interna y dejar atrás el trabajo realizado con las herramientas del plan. Como resultado, al final se contaba con dos frontend: uno desarrollado con las herramientas originales y otro con la migración a las nuevas herramientas internas.

Suamox es un meta-framework con SSR/SSG, enrutado por sistema de archivos y layouts, construido sobre Vite, React y Hono. Este meta-framework ya incluye la configuración y los paquetes estándar del grupo de investigación, y proporciona información clara sobre su estructura de carpetas.

La estructura principal del proyecto se compone del directorio raíz `suamox`, dentro del cual se encuentran los directorios `packages`, `examples` y `docs`. El directorio `packages` agrupa los módulos principales del framework, entre ellos `vite-plugin-pages`, `ssr-runtime`, `hono-adapter`, `head`, `router`, `cli` y `create-app`. Por su parte, `examples` contiene proyectos de referencia y `docs` la documentación técnica. Asimismo, en la raíz del proyecto se encuentran los archivos `PLAN.md` y `CONVENTIONS_v1.md`, utilizados para documentar la pla-

nificación y las convenciones de desarrollo.

En términos generales, la estructura del proyecto fue diseñada para mantener una clara separación de responsabilidades entre sus componentes.

Elise es el design system moderno adoptado para el desarrollo del frontend. Está basado en Radix UI primitives y Tailwind CSS, y se construyó como un monorepo utilizando pnpm. Este enfoque permite mantener una base de componentes consistente, accesible y reutilizable en toda la plataforma. A continuación, se describen los paquetes que componen Elise:

**Tabla 7**
*Paquetes del design system Elise*

| Paquete | Descripción |
|---|---|
| `@calumet/elise-ui` | Librería principal con 45+ componentes accesibles |
| `@calumet/elise-utils` | Utilidades: formularios (Zod), tablas (TanStack), toasts, alerts, fechas |
| `@calumet/elise-icons` | Re-export de Radix Icons |
| `@calumet/elise-linter` | Configuración compartida de ESLint y Prettier |
| `showcase` | App demo interactiva con ejemplos de todos los componentes |

Para el consumo del backend REST se utilizó la API `fetch()`, permitiendo la comunicación asíncrona entre cliente y servidor.

### *6.2.6 Herramienta de despliegue: Nginx + Docker*

El frontend construido con Suamox-Elise (React + Vite + SSR/SSG) se despliega mediante un contenedor Docker que combina:

- **Build stage**: Compilación de la aplicación React con Vite, generando archivos estáticos optimizados.
- **Runtime stage**: Servidor Nginx que distribuye el contenido estático pre-renderizado.

- **Inyección dinámica**: Script de entrypoint que configura variables de entorno en tiempo de ejecución.

**Características clave:**

- **SPA Routing**: Configuración Nginx con fallback a `index.html` para enrutamiento del cliente.
- **Separación de concerns**: Backend (Spring Boot) y frontend (Nginx) como servicios independientes en Docker Compose.
- **Configuración dinámica**: Variables de API inyectadas via `env.js` al iniciar el contenedor.

**Limitación actual**: Nginx solo sirve archivos estáticos; no actúa como reverse proxy hacia el backend.

### *6.2.7 Gestión de dependencias: Maven*

Maven se utilizó para la administración del las bibliotecas y la construcción del proyecto debido a su integración con Spring Boot. Además se uso para compilar las pruebas unitarias.

### *6.2.8 Documentación de API: Swagger*

Swagger se utilizó como herramienta de documentación automática de servicios REST, permitiendo la visualización en una interfaz detallada de nuestros endpoints organizados y con detalles y hacer pruebas rápidas manuales de consumo de api.

### *6.2.9 Protocolo de comunicación: WebSocket*

El módulo de chat en tiempo real implementa el protocolo WebSocket, el cual permite comunicación bidireccional full-dúplex entre cliente y servidor, a diferencia del modelo tradicional basado en petición–respuesta HTTP.

Este protocolo se empleó para:

- Chat en tiempo real entre miembros del grupo.
- Envío de alertas inmediatas como botón de pánico.
- Notificaciones instantáneas.
- Mensaje rápido en el BDP.

#### *6.2.10 Control de versiones: Git*

Se utilizó Git para el control de versiones del proyecto, permitiendo seguimiento estructurado de cambios, optimizar el trabajo en equipo, buena sinergia junto a SCRUM debido a su facilidad para documentar tareas y recuperación de versiones anteriores.

#### *6.2.11 Herramienta de despliegue: Docker*

Docker se utilizó como herramienta principal para la contenedorización del proyecto, permitiendo empaquetar el backend, frontend y la base de datos de prueba Agora, junto con sus dependencias en entornos aislados y reproducibles.

El uso de Docker facilitó:

- Portabilidad del sistema entre diferentes entornos (desarrollo, pruebas y producción)
- Simplificación del proceso de despliegue mediante imágenes y contenedores
- Aislamiento de dependencias, evitando conflictos entre versiones de librerías
- Escalabilidad al integrar fácilmente nuevos servicios en el ecosistema
- Integración con Docker Compose para levantar múltiples servicios (backend, frontend y base de datos) de manera coordinada

Gracias a Docker, se pudo garantizar que la aplicación se ejecutara de forma consistente en cualquier máquina, reduciendo tiempos de configuración y asegurando que las reuniones con el product owner fueran mas rápidas y al punto sin depender de nuestros dispositivos.

## 6.3 Arquitectura de la solución

Así como se mostró en el diagrama de arquitectura (Ver Diagrama de arquitectura); la solución propuesta se implementó bajo el enfoque de arquitectura hexagonal y DDD-LITE. Este enfoque permite separar la lógica de negocio del framework, de la base de datos y de cualquier tecnología externa, facilitando la mantenibilidad y evolución del sistema a futuro.

### *6.3.1 Arquitectura hexagonal*

La arquitectura utilizada es hexagonal o de puertos y adaptadores. En este modelo, el núcleo del sistema está conformado por la capa de dominio, donde residen las entidades y reglas de negocio.

La comunicación entre el dominio y las tecnologías externas (como base de datos, web, mensajería o interfaz gráfica) se realiza mediante puertos definidos como interfaces. Los adaptadores implementan dichas interfaces, además en este proyecto los adaptadores de salida se ubican en la infraestructura y son los encargados de la comunicación con la base de datos mientras que los adaptadores de entrada, responsables de la comunicación con el cliente se ubican en una carpeta aparte como si fueran una capa extra llamada `Api/rest/adaptadorControlador.java`.

### *6.3.2 Domain-Driven Design (DDD-lite)*

El proyecto también adopta un enfoque **DDD-lite**, una versión simplificada de Domain-Driven Design. En este modelo, el sistema se organiza por *features* o subdominios, manteniendo separados los modelos de dominio de la infraestructura técnica. Cada feature se divide en capas internas (`dominio`, `aplicación`, `infraestructura`), lo que asegura que las reglas de negocio permanezcan independientes de detalles como JPA, REST o seguridad. Este enfoque permite claridad inmediata en la ubicación de cada responsabilidad y facilita la evolución hacia DDD completo cuando la complejidad del negocio lo requiera.

- claridad en la separación de responsabilidades

- uso de un lenguaje ubicuo alineado con el negocio
- reducción de acoplamiento entre capas y subdominios
- simplicidad en casos CRUD sin sobrecargar el modelo

La implementación de DDD-lite responde a un estándar establecido por el grupo **Calumet**, de esta manera, se garantiza que la arquitectura utilizada esté alineada con las buenas prácticas y lineamientos definidos por el equipo responsable de la plataforma.

### *6.3.3 Estructura del Backend*

El backend está organizado siguiendo los principios de la arquitectura hablada anteriormente, y por ende veremos una clara separación por módulos/features. Cada uno de estos módulos aplica las cuatro capas que se hablaron en la arquitectura y se mostraran en la siguiente estructura.

#### Estructura General

El backend se organiza dentro de `Backend/src/main/java/org/calumet/-gruposestudio/`, partiendo de la clase principal `GruposestudioApplication.java` (punto de entrada de Spring Boot). Los paquetes de primer nivel se dividen por funcionalidades: `auth/` (autenticación y autorización), `botonpanico/` (botón de pánico), `bugtracker/` (sistema de reporte de bugs), `buscadorparches/` (buscador de parches académicos), `calific/` (calificaciones de grupos y miembros), `chat/` (chat grupal y mensajería), `configuracion/` (configuraciones globales), `filtros/` (filtros y búsquedas), `grupoestudio/` (gestión de grupos de estudio), `miembrogrupo/` (miembros de grupos), `mural/` (mural de noticias y anuncios), `notificacion/` (sistema de notificaciones), `perfil/` (perfiles de usuario), `recurso/` (recursos compartidos), `resena/` (reseñas y comentarios), `shared/` (utilidades compartidas), `solicitudgrupo/` (solicitudes de ingreso a grupos) y `topsemana/` (rankings y tops semanales).

**Patrón por feature**

Cada módulo sigue una estructura consistente de 4 capas. Tomando como ejemplo el módulo `solicitudgrupo/`, la capa `/rest/` contiene el controlador `SoliGController.java` con los endpoints HTTP. La capa `aplicacion/` aloja el servicio `SoliGServicio.java` que implementa los casos de uso. La capa `dominio/` se subdivide en `modelo/` (con la entidad de dominio `SolicitudG.java`) y `repositorio/` (con la interfaz `SoliGRepositorio.java` que define el contrato de persistencia). Finalmente, la capa `infraestructura/persist/` contiene el adaptador concreto `SolicitudGrupoRepositorioSQL.java`, la entidad JPA `SolicitudGrupoEntidad.java` dentro de `entidad/`, y el mapeador `GMapper.java` dentro de `mapeador/` para convertir entre entidades JPA y objetos de dominio.

- **Dominio**: Entidades y Agregados (ej: Usuario, Grupo, Mural)
- **Aplicación**: Servicios de aplicación (casos de uso).
- **Infraestructura**: Persistencia (JPA, Spring Data, migraciones, etc.)
- **Api**: Adaptadores de salida, implementación del repositorio.

### *6.3.4 Patrones de Diseño Implementados*

**Patrón de mappeadores**

Este patrón se usa para tener control de conversión de una entidad a otra manteniendo así la independencia entre capas. Un mappeador convierte un objeto recibido desde la base de datos en uno de dominio y también puede convertir uno de dominio en un DTO, de esta forma las capas nunca se cruzan entre ellas ya que son convertidas antes de llegar a su propio dominio.

Un ejemplo representativo es la clase `SolicitudGrupoMapper`, ubicada en el paquete `solicitudgrupo/infraestructura/persistencia/mapeador/`. Esta clase utilitaria, con constructor privado para evitar instanciación, expone dos métodos estáticos: `toDomain()` que convierte una entidad JPA en un objeto del dominio, y `toOrm(SolicitudGrupo domain)`

que realiza la conversión inversa. De este modo, la lógica de negocio en la capa de dominio nunca depende directamente de las anotaciones o relaciones de persistencia, y los cambios en la estructura de la base de datos solo afectan al mapeador sin impactar las demás capas.

**Beneficios**:

- Separación clara entre modelo de dominio, modelo de persistencia y resultados (DTO).
- Fácil mantenimiento de conversiones

**Patrón DTO (Data Transfer Object)**

Los DTO funcionan moldeando un dominio de forma tal que el cliente recibe exactamente lo que nosotros queremos que reciba, evitando así enviar entidades completas y exponer datos que no deberían. Ademas beneficia al frontend dejando listo todos los datos para simplemente ser mostrados sin tener que hacer procesos extra.

Un ejemplo representativo es el registro `SolicitudGrupoDTO`, definido en el paquete `solicitudgrupo/api/rest/dto/`, que contiene campos como el identificador de la solicitud, del miembro interesado y del grupo de estudio, el estado, la fecha de respuesta y el mensaje asociado. Este DTO es construido en el controlador `SolicitudGrupoController` mediante un método privado `toDto()` que recibe una entidad del dominio, extrae sus campos relevantes y retorna una instancia inmutable del DTO lista para ser serializada como JSON en la respuesta HTTP. De esta forma, el cliente nunca recibe objetos internos del dominio, sino una representación plana y controlada de los datos.

**Beneficios**:

- Control estricto de datos expuestos en el API.
- Evita enviar datos sensibles en el dominio.
- Optimización de payloads.

### *6.3.5 Patrón Observer (WebSocket)*

El patrón Observer se implementa mediante **Spring WebSocket** para notificaciones en tiempo real. El sistema utiliza STOMP sobre WebSocket con SockJS, permitiendo:

- **Sujeto (Publisher)**: Servicios como `NotificacionWebSocketPublisher` y `BdpWebSocketPublisher`
- **Observadores (Subscribers)**: Clientes suscritos a canales WebSocket específicos
- **Eventos notificados**:
  - Mensajes de chat en tiempo real
  - Notificaciones de sistema
  - Actualizaciones de búsqueda de parches (BDP)
  - Alertas de botón de pánico

### *6.3.6 Spring-managed Singletons*

El framework **Spring Boot** gestiona automáticamente el ciclo de vida de los componentes como singletons mediante anotaciones:

- `@Component`, `@Service`, `@Repository`: Instancias únicas por contexto de aplicación
- `@Configuration`: Beans de configuración singleton

Aunque no se implementa manualmente el patrón Singleton clásico, Spring proporciona sus beneficios de forma transparente.

**Patrón Strategy (Implícito en servicios)**

Aunque no es explícito, hay servicios como los filtros en los que se hace uso implícito de este patrón. Por ejemplo, se definió una interfaz `BusquedaStrategy` con un método

`buscar(Criterios criterios)` que retorna una lista de resultados, la cual es implementada de forma concreta por servicios como `BusquedaParchesServicio`, `BusquedaGrupos` y `BusquedaRecursosServicio`, cada uno con su propia lógica de filtrado según el tipo de entidad que gestiona.

## 6.4 Metodología de desarrollo

Como se explicó en el cuarto capítulo **Planificación y análisis**, el desarrollo se gestionó bajo el marco de trabajo SCRUM, con sprints planificados en JIRA y un backlog priorizado. La presente sección se centra en los aspectos técnicos de la implementación, específicamente en la gestión del código fuente, el flujo desde el código hasta su correcto merge en GitHub mediante pull requests y la aprobación de estos usando el sistema de control de versiones Git.

Durante el desarrollo se siguió una estrategia basada en ramas, donde cada historia de usuario o tarea asignada dentro de un sprint era implementada en una rama independiente. Esto permitió aislar los cambios realizados, facilitar las pruebas y reducir conflictos durante la integración del código.

## 6.5 Gestión del código fuente

En este proyecto, como tal, no se tenía un plan definido del flujo de trabajo después de JIRA; es decir, se dio por hecho lo que cada uno de nosotros debía hacer: tomar sus tareas asignadas, implementarlas y listo. Sin embargo, indirectamente se fue formando un flujo más estable, ya que con el pasar de las tareas que se debían completar para cada historia de usuario, se presentaron pequeños desórdenes que hicieron que se fuera adoptando un plan. Los errores nos encaminaban hacia el flujo de trabajo final.

### *6.5.1 Flujo de trabajo*

Al comienzo, las primeras tareas referentes a historias de usuario se desarrollaban en la única rama llamada `producción`. Luego empezamos a notar que definitivamente no podíamos

trabajar un proyecto con tantas tareas de esta forma. Primero, porque no todas las tareas tienen la misma duración y no siempre se terminan de forma escalonada, lo que daba paso a un flujo como: tú haces `git push`, yo hago `git pull` y empiezo lo mío, una práctica completamente lenta que solo funciona si los desarrolladores trabajan en horarios distintos, lo cual no era el caso.

Por esta razón, empezamos a definir una capa principal: toda subtarea de una historia de usuario debía tener una rama asociada, de ser posible con los siguientes convenios para la creación de ramas:

- **Fix:** Algo que debía corregirse en un componente existente.
- **MOGE/#HU:** Implementación nueva de una historia de usuario.
- **Revert/:** Punto de guardado hacia una rama estable mientras se solucionan problemas.

Luego de estos convenios el trabajo mejoró, pero esto no evitaba todos los problemas y, además, ahora había que afrontar los merges. Estos son una herramienta de GitHub que toma dos ramas y las combina para crear una con los cambios actualizados. Así que apareció una nueva capa: antes del merge se debe realizar un commit, y este commit debe ser descriptivo. Nuestra convención de ramas y comentarios puede verse en las figuras 41 y 42.

**Figura 41**
*Ejemplo de ramas creadas en GitHub*

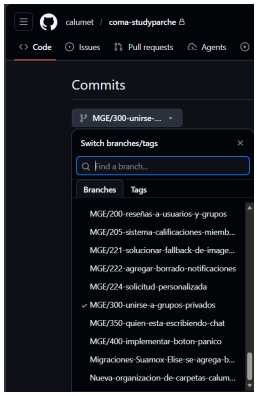
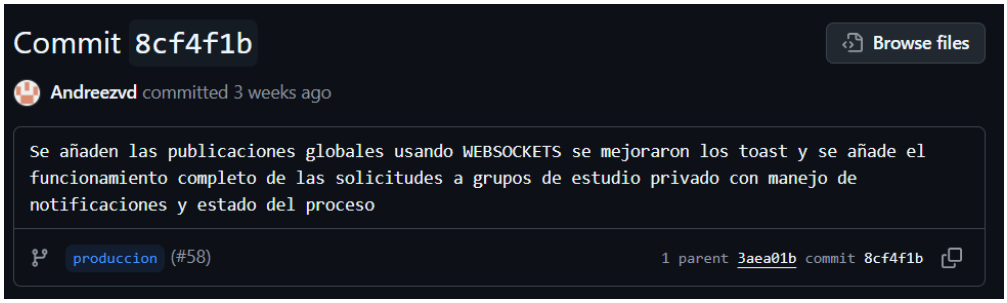


**Figura 42**
*Ejemplo de commit en GitHub*




Estos commits representan la explicación de la solución dada a una historia de usuario o una subtarea. Sin embargo, aún queda un paso más: una vez que la rama está lista y contiene todos los commits realizados, es momento de crear un pull request. El pull request indica a la persona encargada del proyecto, ya sea el Scrum Master o líder de desarrollo, que es momento de revisar el código.

En este punto se realiza una capa de revisión. Para este proyecto no se planteó actuar como QA formal, pero sí se realizaba una revisión de código, aunque no tan profunda como implementar pruebas en cada rama para su aprobación. Se revisaba la funcionalidad general y, si todo era correcto, se procedía al último paso: realizar el merge de la rama con `producción`.

Lo habitual es que las ramas sean compatibles, pero en proyectos que escalan eventualmente aparecen conflictos. En ese momento entra una etapa clave: la resolución de inconsistencias, donde se decide qué cambios deben mantenerse para lograr un merge correcto.

De esta manera, el flujo completo no fue planeado inicialmente; los errores nos guiaron hacia él. Sin embargo, al final logramos dominarlo y Git se convirtió en una herramienta esencial en el día a día.

El proceso de integración de código seguido fue el siguiente:

1. Un desarrollador creaba una rama `MOGE/#nombre-historia` desde `producción`, basada en una tarea de JIRA.

2. Se realizaban commits siguiendo la convención *describir la solución*.

3. Al finalizar, se abría un Pull Request (PR) hacia `producción`.
4. El otro miembro del equipo revisaba el código y aprobaba los cambios.
5. Se realizaba el merge a `producción`.
6. Periódicamente, se resolvían conflictos derivados del merge.

### *6.5.2 Evidencias del proceso de desarrollo*

A continuación, se presentan las estadísticas generales del repositorio en GitHub durante el desarrollo:

**Tabla 8**
*Estadísticas del repositorio Git*

| **Métrica** | **Cantidad** |
|---|---|
| Total de commits | 238 |
| Ramas creadas | 62 |
| Pull requests | 71 |
| Merges realizados | 71 |
| Desarrolladores activos | 2 |

## 7. Pruebas y validación

### 7.1 Pruebas de caja negra

Las pruebas de caja negra constituyen una técnica fundamental en la ingeniería de software, pues se centran en el comportamiento externo del sistema sin considerar la lógica interna del código. De acuerdo con (Pressman & Maxim, 2014), “Las pruebas de caja negra se centran en el comportamiento externo del software. El diseño de casos de prueba se basa en especificaciones y requisitos, no en la estructura interna del programa”. Esta perspectiva permite validar que el sistema cumple con los requisitos funcionales definidos, garantizando la calidad desde el punto de vista

del usuario final.

Asimismo, la Universidad de Santo Tomás, 2024 enfatiza que “Las pruebas de caja negra validan la funcionalidad del sistema sin considerar la lógica interna, centrándose en las entradas y salidas para asegurar el cumplimiento de los requisitos funcionales”. Esta definición respalda la estructura aplicada en los casos de prueba, donde se documentan las condiciones iniciales, los datos de entrada, los pasos de ejecución y los resultados esperados y obtenidos.

Para la organización de los casos de prueba se siguieron las recomendaciones de estándares internacionales como el IEEE 829 y las buenas prácticas del ISTQB, que establecen que cada caso debe incluir un identificador, título, módulo, precondiciones, entradas, pasos de ejecución, resultados esperados y resultados obtenidos, junto con su estado final. Esta estructura asegura trazabilidad, repetibilidad y cobertura completa de los requisitos funcionales, alineándose con las normas de documentación de pruebas de software.

### *7.1.1 Gestión de Grupos de Estudio*

Referente al CRUD de un grupo de estudio:

**Creación de grupo de estudio**

- **Precondiciones:** Usuario registrado y autenticado en la aplicación. Usuario vinculado a una escuela y programa académico.
- **Entrada:**
  - Nombre del grupo.
  - Descripción del grupo (Opcional).
  - Imagen de portada (Opcional).
  - Materia.
  - Horario semanal (Opcional).
  - Horario de reunión fija (Opcional).

- Visibilidad (Selección entre público o privado).

- **Pasos de ejecución:**

  1. Ingresar a la aplicación.
  2. Seleccionar la opción “Crear grupo”.
  3. Completar los campos requeridos y opcionales según corresponda.
  4. Confirmar la creación del grupo.

- **Resultado esperado:** El sistema muestra un mensaje de confirmación: “Grupo creado correctamente”. El nuevo grupo aparece en la lista de grupos del usuario con los datos ingresados.

- **Resultado obtenido:** El sistema mostró el mensaje de confirmación y el grupo se creó correctamente con los datos proporcionados por el usuario.

- **Estado:** Aprobada.

### *7.1.2 Unión e Inscripción a Grupos*

**Unirse a un Grupo de Estudio Público**

- **Precondiciones:**

  - Usuario registrado y autenticado en la aplicación.
  - Usuario ha identificado un grupo al cual desea pertenecer.

- **Entrada:** No se requieren datos adicionales por parte del usuario.

- **Pasos de ejecución:**

  1. Ingresar a la aplicación.
  2. Seleccionar la sección “Buscar grupos”.

3. Escribir el nombre del grupo o aplicar filtros por materia y visibilidad.
4. Pulsar el botón “Unirse” en la tarjeta del grupo.
5. Pulsar el botón “Unirse” en la pregunta de confirmación.

- **Resultado esperado:** El sistema muestra la pregunta de control, seguido el mensaje de confirmación. El grupo ahora aparece en la sección denominada “Mis grupos”.
- **Resultado obtenido:** El sistema mostró el mensaje: “¿Quieres unirte al grupo “Nombre del grupo”?”, posteriormente mostró el mensaje de confirmación: “Ya haces parte de “Nombre del grupo””. El grupo ahora apareció en la sección correspondiente “Mis grupos”.
- **Estado:** Aprobada.

**Solicitud de Unión a Grupo Privado**

- **Precondiciones:**
  - Usuario registrado y autenticado en la aplicación.
  - Usuario ha identificado un grupo privado al cual desea pertenecer.
- **Entrada:** No se requieren datos adicionales por parte del usuario.
- **Pasos de ejecución:**
  1. Ingresar a la aplicación.
  2. Seleccionar la sección “Buscar grupos”.
  3. Localizar el grupo privado por nombre o filtros.
  4. Pulsar el botón “Solicitar” en la tarjeta del grupo.
  5. Pulsar el botón “Enviar solicitud”.
- **Resultado esperado:** El sistema muestra la pregunta de control, seguido el mensaje de confirmación. El grupo ahora aparece en la sección denominada “Mis inscripciones”.

- **Resultado obtenido:** El sistema mostró el mensaje: "¿Quieres solicitar ingreso al grupo privado "Nombre del grupo"?", posteriormente mostró el mensaje de confirmación: "Solicitud enviada para "Nombre del grupo"". El grupo ahora apareció en la sección correspondiente "Mis inscripciones".

- **Estado:** Aprobada.

### *7.1.3 Publicaciones y Recursos*

**Publicar en el Mural del Grupo**

- **Precondiciones:**
  - Usuario registrado y autenticado en la aplicación.
  - Usuario pertenece al grupo de estudio en el que desea agregar una publicación.

- **Entrada:**
  - Título de la publicación.
  - Color del texto (opcional).
  - Fondo de la publicación (opcional).
  - Contenido de la publicación.

- **Pasos de ejecución:**
  1. Ingresar a la aplicación.
  2. Seleccionar la sección "Mis grupos".
  3. Seleccionar la tarjeta del grupo.
  4. En la pestaña Mural, seleccionar el botón "Nueva publicación".
  5. Completar el título, color de texto (opcional), fondo de la publicación (opcional) y contenido.

6. Seleccionar la opción “Publicar”.

- **Resultado esperado:** El sistema muestra el mensaje de confirmación y la publicación es agregada al mural del grupo correspondiente.
- **Resultado obtenido:** El sistema mostró el mensaje: “Publicación guardada. La publicación ya está disponible en el mural”. La publicación ahora está disponible en la sección “Publicadas” en el mural del grupo.
- **Estado:** Aprobada.

**Publicar Recurso en el grupo**

- **Precondiciones:**
  - Usuario registrado y autenticado en la aplicación.
  - Usuario pertenece a un grupo de estudio.
- **Datos de Entrada:**
  - Título del recurso.
  - Descripción del recurso.
  - Archivo adjunto (opcional).
- **Pasos de ejecución:**
  1. Ingresar a la aplicación.
  2. Seleccionar la sección “Mis grupos”.
  3. Seleccionar la tarjeta del grupo para acceder al entorno.
  4. Seleccionar la pestaña “Recursos”.
  5. Completar el título, descripción y adjuntar archivo si se desea.
  6. Pulsar el botón “Guardar recurso”.

- **Resultado esperado:** El sistema envía el aviso: “Recurso compartido, el recurso ya aparece en el panel del grupo”. El recurso se almacena en la sección “Recursos” del entorno del grupo. El recurso es visible para todos los miembros del grupo.
- **Resultado obtenido:** El sistema mostró el aviso: “Recurso compartido, el recurso ya aparece en el panel del grupo”. El recurso se almacenó correctamente en la sección “Recursos” del entorno del grupo. El recurso fue visible para todos los miembros del grupo sin errores en la visualización.
- **Estado:** Aprobada.

### *7.1.4 Chat de Grupo*

**Envío de Mensajes de Texto en Chat de Grupo**

- **Precondiciones:**
  - Usuario registrado y autenticado en la aplicación.
  - Usuario pertenece a un grupo de estudio y un chat relacionado a ese grupo.
- **Datos de Entrada:** Texto escrito por el usuario (ejemplo: “Hola equipo”).
- **Pasos de ejecución:**
  1. Ingresar a la aplicación.
  2. Seleccionar la burbuja de chat o entrar al entorno del grupo.
  3. Escribir un mensaje en el campo de texto.
  4. Pulsar “Enviar” o dar Enter.
- **Resultado esperado:**
  - El mensaje aparece inmediatamente en el chat.
  - Todos los miembros del grupo reciben el mensaje de forma automática.

- El historial del chat se actualiza mostrando el nuevo mensaje junto con los anteriores.

- **Resultado obtenido:**
  - El mensaje escrito por el usuario se mostró inmediatamente en el chat.
  - Todos los miembros del grupo recibieron el mensaje de forma automática.
  - El historial del chat se actualizó correctamente mostrando el nuevo mensaje junto con los anteriores.

- **Estado:** Aprobada.

### *7.1.5 Funciones Especiales*

**Activar el Botón de Pánico**

- **Precondiciones:**
  - Usuario registrado y autenticado en la aplicación.
  - Usuario pertenece a un grupo de estudio.

- **Datos de Entrada:** No se requieren datos adicionales.

- **Pasos de ejecución:**
  1. Ingresar a la aplicación.
  2. Seleccionar la sección “Mis grupos”.
  3. Seleccionar la tarjeta del grupo para acceder al entorno.
  4. Seleccionar la pestaña “Botón de Pánico”.
  5. Pulsar el botón “Pánico”.

- **Resultado esperado:** El sistema envía una notificación automática a todos los miembros del grupo, incluyendo al líder, además envía un mensaje automático por parte del usuario al

chat grupal. El usuario puede ver que se indica en cuantó tiempo podrá volver a activarlo en ese grupo.

- **Resultado obtenido:** Al activar el botón de pánico, el sistema envió la notificación correctamente a todos los miembros del grupo y al líder, así mismo en el chat de ese grupo se envió el mensaje automático "Activé el Boton del Panico. Necesito ayuda urgente ahora" por parte de ese usuario. Se observa correctamente el mensaje de confirmación de "Alerta enviada". Se informó que podrá volver a usarlo en $x$ minutos, se mostró en pantalla sin errores la disminución en minutos.

- **Estado:** Aprobada.

**Buscador de Parches BDP Exitoso**

- **Precondiciones:**
  - Usuario registrado en la aplicación.
  - Usuario ha ingresado a la aplicación (no es necesario pertenecer a un grupo de estudio).

- **Datos de Entrada:** Materia seleccionada por el usuario para estudiar (ejemplo: "Álgebra Lineal").

- **Pasos de ejecución:**
  1. Ingresar a la aplicación.
  2. Seleccionar el botón "Unirse al BDP".
  3. Escribir la materia que desea estudiar.
  4. Confirmar la solicitud de búsqueda de compañeros pulsando "Buscar aliados en la UIS".
  5. Se inicia un cronómetro de espera denominado "Tiempo en cola".

6. El sistema busca automáticamente otros dos usuarios que hayan solicitado LFR con la misma materia.
7. Se indica que ya se encontró parche y se observa un lugar de reunión recomendado.

- **Resultado esperado:**
  - Mientras se realiza la búsqueda, el estado mostrado en el cuadro es “Encontrando”.
  - Si se encuentran dos compañeros adicionales:
    - El estado cambia a “Encontrado”.
    - Se asigna automáticamente un lugar de reunión al azar dentro de la universidad.
  - Si no se encuentran compañeros dentro del tiempo de espera:
    - El sistema finaliza la búsqueda al cumplirse el cooldown de 5 minutos.
    - Se muestra un mensaje de cierre de búsqueda.
- **Resultado obtenido:**
  - El sistema mostró el estado “Encontrando” durante la búsqueda.
  - Luego cambió a “Encontrado” al localizar dos compañeros adicionales.
  - Se asignó un lugar de reunión en la universidad.
  - La sesión BDP se inició correctamente.
- **Estado:** Aprobada.

**Solicitud Personalizada**

- **Precondiciones:**
  - Usuario registrado y autenticado en la aplicación.
  - Usuario pertenece al grupo de estudio.
- **Entrada:**

  - Mensaje de solicitud personalizada.

- **Pasos de ejecución:**

  1. Ingresar a la aplicación.
  2. Seleccionar la sección “Mis grupos”.
  3. Seleccionar el grupo de estudio correspondiente.
  4. Entrar a la pestaña “Sobre el grupo”.
  5. Pulsar sobre el círculo del perfil de un miembro.
  6. Seleccionar la opción “S. Personalizada”.
  7. Ingresar el mensaje de solicitud.
  8. Pulsar el botón “Enviar solicitud”.

- **Resultado esperado:** El sistema muestra un mensaje de confirmación indicando que la solicitud personalizada fue enviada correctamente y se registra en el historial del grupo.
- **Resultado obtenido:** El sistema muestra el mensaje de confirmación “Solicitud personalizada enviada correctamente” y la solicitud aparece reflejada en la sección de notificaciones de el usuario receptor del mensaje.
- **Estado:** Aprobada.

## 7.2 Pruebas de caja Blanca

Las **pruebas de caja blanca** son una técnica de aseguramiento de calidad que se centra en la estructura interna del software.

El objetivo principal es garantizar que todas las *sentencias*, *ramas*, *condiciones* y *caminos* posibles del programa sean ejecutados al menos una vez, donde se ejecuta una acción y comparamos el resultado esperado con el resultado obtenido, de esta forma podemos saber si se aprueba o no la funcionalidad.

En otras palabras en esta prueba se busca validar el sistema mirando por dentro, es decir, yo tengo acceso al código y por ende sé que puedo esperar y que debería hacer al pasar por esa sección de código, si todo se cumple se aprueba si algo falla se debe revisar.

- **Cobertura de sentencias:** asegurar que cada línea de código se ejecute.
- **Cobertura de ramas:** verificar que cada decisión condicional (if/else) se evalúe en todas sus posibilidades.
- **Cobertura de condiciones:** comprobar que cada condición booleana dentro de las decisiones se evalúe en verdadero y falso.
- **Cobertura de caminos:** garantizar que todos los flujos posibles de ejecución se recorran.

La documentación de los casos de prueba de caja blanca se estructuró en formato tabular, incluyendo identificador, condiciones de entrada, acción ejecutada, resultado esperado, resultado obtenido y estado de la prueba. Este formato se basa en las recomendaciones de estándares de pruebas de software como IEEE 829 e ISO/IEC/IEEE 29119, los cuales sugieren una documentación clara, trazable y verificable de los casos de prueba.

### *7.2.1 Pruebas de Grupo de Estudio*

El módulo de Grupo de Estudio constituye uno de los componentes principales del sistema, ya que actúa como elemento central para la interacción entre los usuarios y sirve como base para otros módulos asociados dentro del mural. A partir de este servicio se gestionan aspectos fundamentales como la creación, consulta y administración de grupos, así como la integración con funcionalidades complementarias desarrolladas en otros servicios.

Si bien es posible realizar pruebas generales que involucren múltiples funcionalidades relacionadas con los grupos de estudio, se optó por mantener una separación clara de las pruebas de acuerdo con la estructura definida en los endpoints y la arquitectura del sistema. Este enfoque permite evaluar cada funcionalidad de manera independiente, facilitando la identificación de errores,

la validación de los requisitos específicos y la trazabilidad de los resultados obtenidos durante el proceso de pruebas.

**Validación de campos obligatorios en creación de grupo de estudio**

**Tabla 9**
*Validación de campos obligatorios en creación de grupo de estudio*

| ID | Campo omitido | Acción realizada | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|
| CP-01 | NombreGrupoEstudio | Crear grupo sin nombre | Error indicando campo faltante | El nombre del grupo es obligatorio. | Aprobado |
| CP-02 | Descripcion | Crear grupo sin descripción | Error indicando campo faltante | La descripción del grupo es obligatoria. | Aprobado |
| CP-03 | IdMat | Crear grupo sin materia | Error indicando campo faltante | La materia es obligatoria. | Aprobado |
| CP-04 | NombreGrupoEstudio | Crear grupo con nombre vacío () | Error indicando campo inválido | El nombre del grupo no puede estar vacío. | Aprobado |
| CP-05 | Descripcion | Crear grupo con descripción vacía () | Error indicando campo inválido | La descripción del grupo no puede estar vacía. | Aprobado |

### *7.2.2 Publicaciones y recursos*

**Publicar en el mural**

**Tabla 10**
*Casos de prueba para publicar en mural*

| ID | Condición de entrada | Acción realizada | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|
| CP-19 | Publicación válida | Ejecutar publicación con título y contenido | El sistema guarda y muestra la publicación en el mural | Publicación visible en mural | Aprobado |
| CP-20 | Publicación sin título | Intentar publicar sin título | El sistema impide la acción y muestra advertencia | Titulo vacío añade un título antes de guardar la publicación | Aprobado |
| CP-21 | Publicación sin contenido | Intentar publicar sin contenido | El sistema impide la acción y muestra advertencia | Contenido vacío añade al menos un bloque antes de guardar la publicación | Aprobado |
| CP-22 | Publicación vacía | Intentar publicar sin título ni contenido | El sistema impide la acción y muestra advertencia | Campos incompletos Debes completar el titulo y el contenido antes de guardar la publicación. | Aprobado |

**Publicar Recurso**

**Tabla 11**
*Casos de prueba para publicación de recursos en grupos de estudio*

| ID | Campo ó Condición | Acción realizada | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|
| CP-24 | Recurso con archivo | Ejecutar publicación con título y archivo adjunto | El sistema guarda y comparte el recurso en el grupo | Recurso publicado correctamente | Aprobado |
| CP-25 | Recurso vacío | Intentar publicar sin título, archivo ni descripción | El sistema impide la acción y muestra advertencia de título obligatorio | Mensaje ”Debes ingresar un título”mostrado | Aprobado |

### *7.2.3 Chat de Grupo*

**Pruebas audio y mensaje chat**

**Tabla 12**
*Casos de prueba para envío de mensajes y audios en chat de grupo*

| ID | Campo ó Condición | Acción realizada | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|
| CP-37 | Mensaje válido | Ejecutar envío de mensaje con al menos un carácter | El sistema guarda y muestra el mensaje en el chat | Mensaje enviado y visible en chat | Aprobado |

### *7.2.4 Funciones especiales*

**Activar Botón de Pánico**

**Tabla 13**
*Casos de prueba para activación del botón de pánico*

| ID | Campo ó Condición | Acción realizada | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|
| CP-51 | Activación sin cooldown | Ejecutar activación del botón de pánico disponible | El sistema registra la alerta y notifica al grupo | Alerta enviada y notificación recibida | Aprobado |
| CP-52 | Activación durante cooldown | Intentar activar el botón de pánico bloqueado | El sistema impide la acción, botón deshabilitado y cursor de bloqueo visible | Botón gris, cursor de bloqueo, acción no disponible | Aprobado |

**Buscador de Parches BDP**

**Tabla 14**
*Casos de prueba para búsqueda de compañeros con buscarBDP()*

| ID | Campo ó Condición | Acción realizada | Resultado esperado | Resultado obtenido | Estado |
|---|---|---|---|---|---|
| CP-53 | Compañeros encontrados | Ejecutar búsqueda de compañeros disponibles en BDP | El sistema muestra lista de compañeros encontrados | Lista de compañeros visible en pantalla | Aprobado |

### *7.2.5 Pruebas unitarias automatizadas*

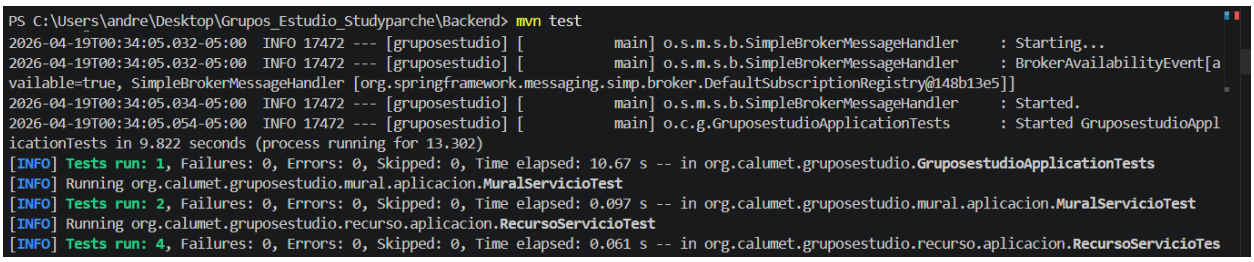
Se implementaron de manera parcial pruebas unitarias automatizadas sobre componentes críticos del sistema, utilizando herramientas de testing del entorno de desarrollo. Estas pruebas permiten validar comportamientos específicos de métodos y servicios, complementando las pruebas de caja blanca. Solo se realizaron pruebas unitarias en los módulos principales del grupo de estudio.

Se utilizó JUnit para la ejecución de pruebas unitarias y Mockito para simular dependencias como repositorios, permitiendo validar la lógica de negocio de forma aislada.

Ninguno de los dos desarrolladores había trabajado antes con pruebas unitarias; de hecho, nunca habíamos realizado pruebas más allá de caja negra y caja blanca, por lo que representó un cambio importante pasar a pruebas automatizadas. Sin embargo, se contó con la ventaja de que Spring Boot junto con Maven facilita este proceso, ya que proporciona de forma predeterminada la estructura necesaria para la implementación y ejecución de pruebas.

**Figura 43**

*Resultados de las pruebas usando mvn test*




### *7.2.6 Resultados finales del sistema*

Como resultado del proceso de desarrollo e implementación, se obtuvo un sistema funcional que integra los diferentes módulos propuestos. Si bien no son exactamente iguales a los diseños planteados en los mockups, se partió de estos para llegar al diseño actual; simplemente se les dio una modernización.

Esta modernización fue pensada para brindar una mejor experiencia a la comunidad, ofre-

ciendo una herramienta que no se sienta institucional al momento de usarla, sino una plataforma hecha para estudiantes, en la que pueden personalizar diferentes aspectos. Aunque al inicio puede parecer minimalista, finalmente se comporta como un lienzo en blanco que los usuarios pueden adaptar a sus necesidades. Además, se implementaron herramientas didácticas como el buscador de parches, orientado a encuentros informales de estudio rápidos, y funcionalidades más dinámicas como el botón del pánico.

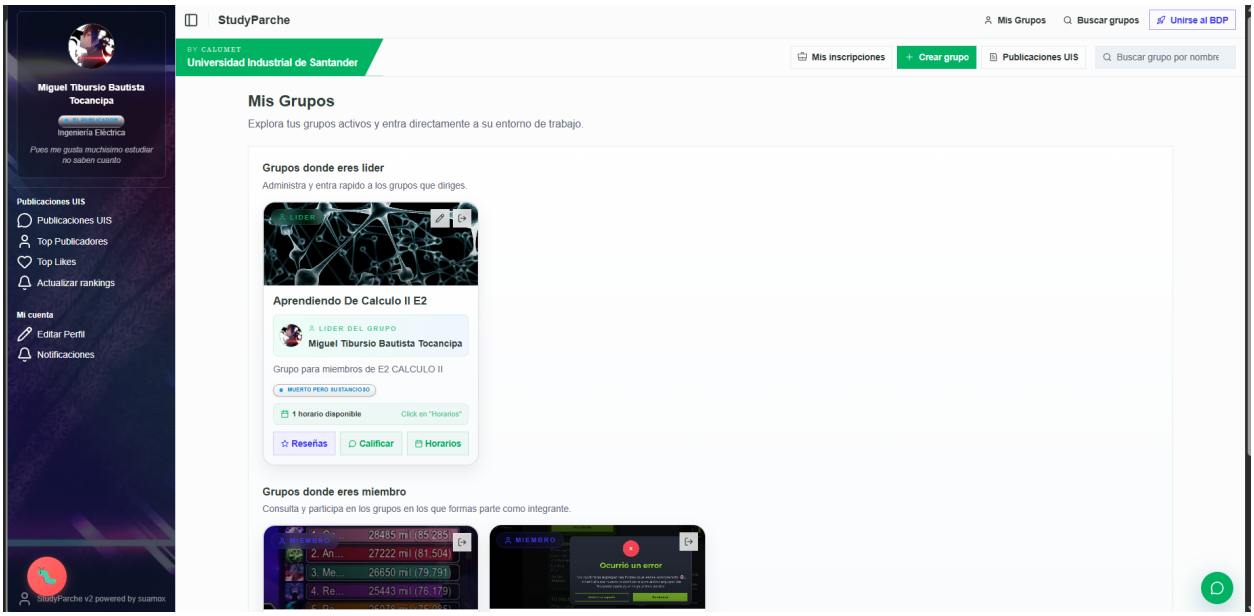
También influyó el hecho de haber cambiado de herramientas frontend y contar con Elise como biblioteca, ya que esta ofrecía un estilo completamente diferente al que se había planteado en los mockups. Sin embargo, la referencia en la disposición de los elementos se procuró mantener.

A continuación, se presentan algunas vistas finales del sistema en funcionamiento:

- **Ventana de grupos de estudio:** Permite observar grupos de estudio a los que pertenece el usuario.

**Figura 44**

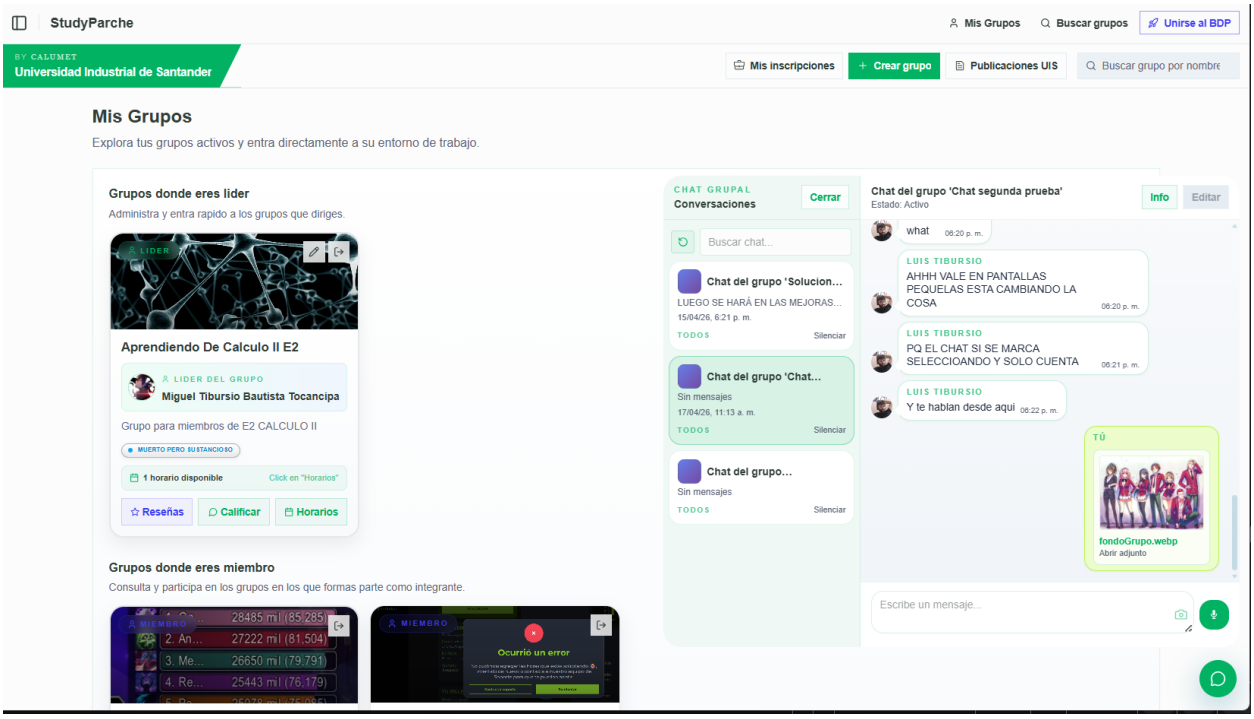
*Panel de grupos de estudio del usuario*




- **Chat grupal en tiempo real:** permite la comunicación en tiempo real entre los miembros del grupo.

**Figura 45**

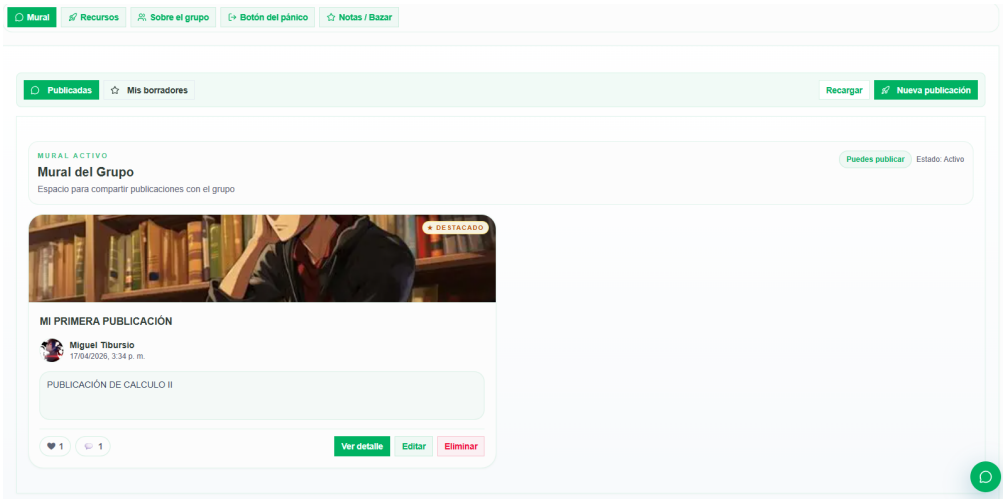
*Vista final del chat grupal*




- **Mural de publicaciones:** espacio donde los usuarios pueden compartir contenido y gestionar interacciones.

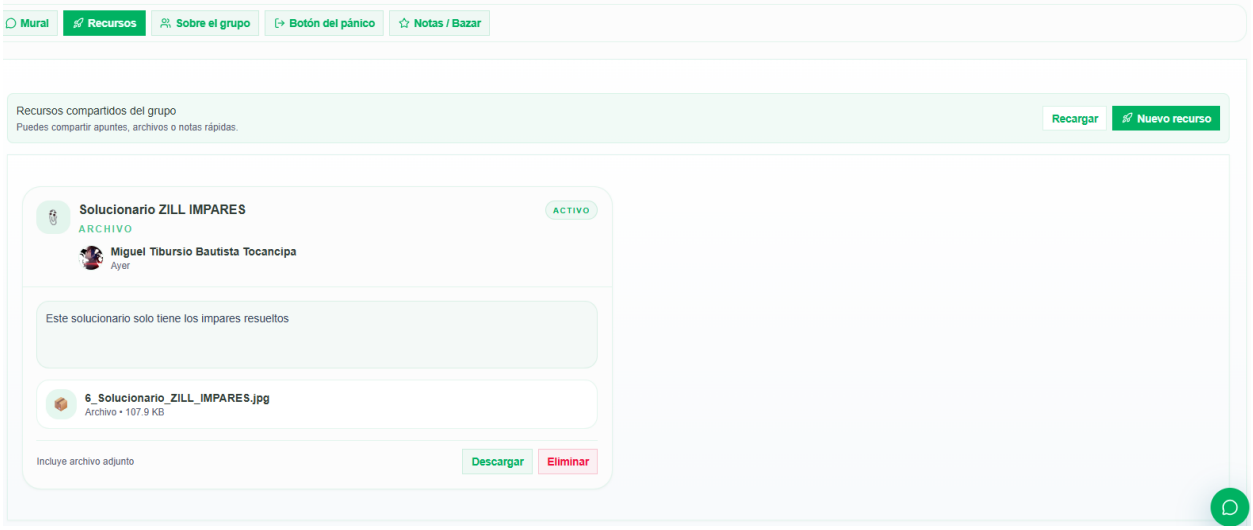
**Figura 46**

*Vista final del mural de publicaciones*




- **Panel de recursos:** permite la gestión y acceso a materiales compartidos.

**Figura 47**
*Vista final del panel de recursos*

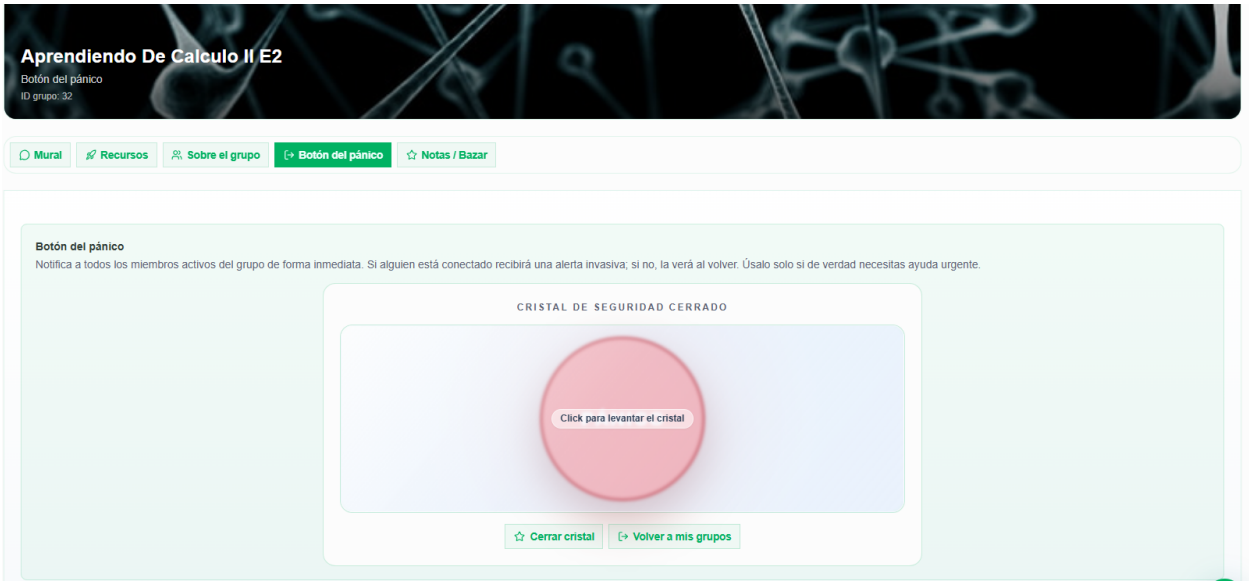



- **Buscador de parches (BDP):** funcionalidad de emparejamiento automático entre estudiantes.

**Figura 48**
*Vista del Buscador de parches*


- **Botón del pánico:** funcionalidad de alerta, que solicita ayuda a los miembros del grupo de forma molesta.

**Figura 49**
*Vista del Botón del pánico*




### 7.3 Capacitación y documentación

Finalmente, se puede encontrar la validación de calidad del sistema, la cual fue realizada en un formato de Excel siguiendo estándares de calidad reconocidos. En este proceso se procuró ser lo más objetivos posible con el fin de brindar una retroalimentación del software final. Esta información se encuentra disponible en el Apéndice C.

## 8. Despliegue

El proceso de despliegue del sistema se llevó a cabo en cuatro entornos: desarrollo local, integración mediante contenedores, pruebas institucionales y producción.

### 8.1 Entorno de desarrollo local

Durante la fase inicial, el sistema se ejecutó en un entorno local. El backend se levantaba mediante `mvn spring-boot:run`, el frontend mediante `pnpm run dev` y la base de datos en un servidor MySQL local.

En este punto se podría afirmar que el proyecto funcionaba únicamente en las máquinas

de desarrollo, lo cual es parcialmente cierto. Proyectos de este tipo suelen depender de múltiples herramientas y configuraciones adicionales, lo que dificulta su ejecución en entornos externos y la validación por parte de terceros.

Por ello, tras la primera semana de desarrollo y a partir de observaciones del product owner —quien manifestó la dificultad de visualizar avances fuera del entorno local— se decidió implementar un proceso de contenedorización utilizando Docker, con el objetivo de estandarizar el entorno de ejecución.

## 8.2 Entorno de integración con contenedores

Con el fin de garantizar consistencia entre entornos y facilitar la validación funcional, se procedió a contenerizar el sistema completo: backend con su *Dockerfile* y frontend con su respectivo *Dockerfile*.

En este entorno se integró la base de datos AGORA, utilizada durante la mayor parte del desarrollo, aunque no en el entorno de producción final debido a restricciones de infraestructura. La configuración se realizó mediante un archivo `docker-compose`, el cual define:

- Volúmenes para persistencia de datos y almacenamiento de archivos multimedia.
- Servicio de base de datos.
- Contenedor backend basado en Spring Boot.
- Servidor web (Nginx) encargado de servir el frontend.

Este enfoque permitió lograr un entorno de ejecución reproducible y desacoplado de las máquinas de desarrollo.

El archivo `docker-compose` define tres servicios principales: un contenedor backend construido desde `./Backend`, expuesto en el puerto `9090`, con volúmenes para persistencia de archivos multimedia y dependencia del servicio de base de datos; un frontend construido desde `./Frontend-Suamox-Elise`, servido via Nginx en el puerto `3000:80`, con variables de

entorno que apuntan a la URL del backend; y una base de datos MySQL 8.0 (`agora-db`) con reinicio automático, persistencia mediante un volumen nombrado `agora_data` y credenciales parametrizables mediante variables de entorno. Las variables de entorno del backend incluyen la ruta de *uploads*, el puerto del servidor, la configuración de logs JPA y los datos de conexión a la base de datos.

### 8.3 Entorno de pruebas institucional

El sistema fue desplegado en un servidor de pruebas institucional con acceso controlado. Para ello, se configuraron variables de entorno en un archivo `.env` y se utilizó una herramienta institucional de despliegue automatizado (Super Cóndor 2.0), la cual permite integrar repositorios de GitHub y ejecutar contenedores en infraestructura universitaria.

Este proceso permitió validar el sistema en un entorno más cercano a producción, sin depender de configuraciones locales.

Por motivos de seguridad, no se incluyen detalles específicos del servidor ni el enlace de acceso al entorno de pruebas.

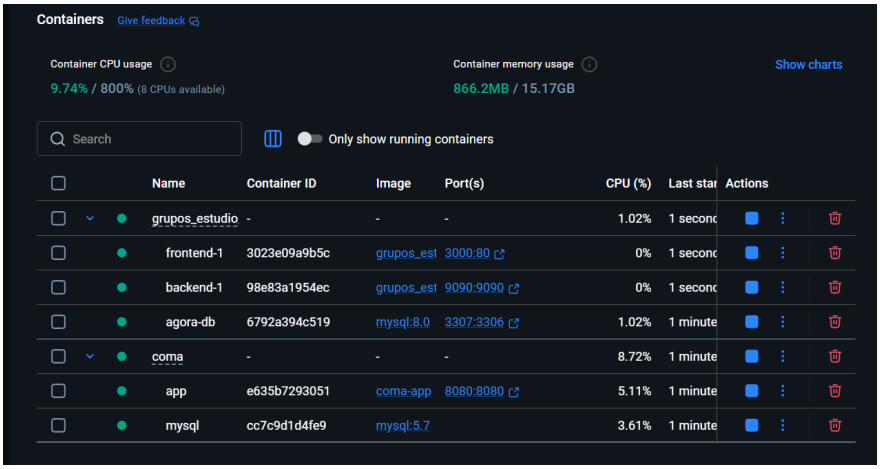
En este entorno también se realizaron pruebas de comportamiento ante fallos parciales del sistema, tales como:

- Verificar si el microservicio continúa operando ante la caída del sistema principal (COMA).
- Evaluar si el frontend puede mantener funcionalidad parcial mediante consumo de API.
- Validar la independencia del sistema principal frente a la indisponibilidad del microservicio.

Las pruebas se realizaron mediante la detención controlada de contenedores, permitiendo analizar el comportamiento del sistema ante diferentes escenarios de fallo.

Como resultado, se evidenció que los servicios pueden operar de forma independiente, cumpliendo con el principio de desacoplamiento esperado en una arquitectura basada en microservicios. Esto confirma que el sistema contribuye a la evolución de COMA hacia un enfoque distribuido.

**Figura 50**
*Contenedores de ambos servicios ejecutándose de forma independiente*




## 8.4 Entorno de producción

Finalmente, se gestionó la asignación de un dominio público para el sistema. Durante el desarrollo, el microservicio fue referido de distintas formas como “servicio”, “módulo” o “grupos de estudio”, ya que no se había definido un nombre formal.

En esta etapa se consolidó el nombre *StudyParche*, y se solicitó el dominio: `studyparche.uis.edu.co`, con el objetivo de permitir su acceso externo.

Para este entorno se realizaron configuraciones adicionales, incluyendo:

- Migración de base de datos, incluir todas las tablas de Agora en poseidon.
- Ajuste de variables de entorno específicas para producción.
- Configuración de almacenamiento y dominio; aunque Docker permite el uso de volúmenes, en el entorno institucional se requirió enlazar dichos volúmenes con el sistema de almacenamiento del servidor, garantizando así la persistencia de archivos como imágenes y recursos.
- Integración con el sistema principal de la plataforma.

Este entorno representa la versión final del sistema, preparada para su uso por usuarios

finales, garantizando acceso externo, estabilidad y persistencia de la información. Posterior al despliegue se documenta el manual de usuario el cual puede ver en el Apéndice A.

## 9. Conclusiones

El desarrollo del microservicio de grupos de estudio permitió cumplir con el objetivo principal; se logró finalmente implementar en la plataforma COMA un módulo que puede migrar a microservicio cuando la infraestructura lo permita.

Sentimos que las herramientas implementadas en StudyParche pueden mejorar la integración universitaria y el aprendizaje colectivo si se usan de forma adecuada, e incluso fortalecer lazos y dejar un "legado" a las futuras generaciones mediante publicaciones y recursos compartidos.

A lo largo de este proyecto se deja clara la evidencia de que una buena planificación inicial puede ser la mejor arma frente a nuevos retos. Tener un plan a seguir en todo momento aleja la incertidumbre sobre si lo que estamos haciendo tendrá que cambiarse. Además, el uso de SCRUM como herramienta iterativa permitió una gran flexibilidad ante los cambios, incluso cuando se requerían modificaciones en las tablas de la base de datos.

Este proyecto nos deja claro que, como ingenieros de sistemas, tenemos muchos ámbitos por explorar. Si bien no se profundizó completamente en cada uno, se abordaron diferentes roles: se trabajó en ingeniería de requerimientos, diseño como arquitectos de software, implementación, validación como Q/A, pruebas unitarias como testers, así como roles de Scrum Master, desarrollo frontend y backend, junto con la interacción constante con un product owner mediante reuniones periódicas, correcciones y retroalimentación. En conclusión, el ciclo de vida del software abarca múltiples áreas de la ingeniería que deben ser cuidadosamente abordadas, ya que todas son fundamentales en la entrega de un producto.

Se destaca la importancia de la documentación y de las herramientas que facilitan su elaboración. Como aprendizaje general, este proyecto contribuyó significativamente al fortalecimiento de competencias técnicas en el desarrollo de software.

Algo importante a destacar es la claridad del sistema en la mente del desarrollador una vez

que se pasa por un ciclo de desarrollo documentado como este. Se inicia con una idea, se construyen los requerimientos como primer acercamiento al sistema, luego se transforman en historias de usuario, posteriormente en tareas, y finalmente se diagraman. Todo esto permite formar una idea clara de cómo se implementará en código, qué hacer primero y cómo avanzar, reduciendo la incertidumbre. La documentación se convierte así en una guía constante durante todo el proceso.

## 10. Trabajo futuro

En el mural, una buena forma de mejorar las publicaciones destacadas es destacarlas por un tiempo límite para que el líder del grupo tenga menos trabajo.

En el chat, borrar comentarios puede ser añadido en el futuro como una nueva forma de evitar errores de los miembros. En el chat también se podría añadir la edición de los mensajes; se puede modernizar un poco más a futuro, como agregar vistos o replicar mensajes.

En los recursos, poner filtros por tema de recurso o algo similar.

Para el BDP, las mejoras pueden ser muchas. Por ejemplo, en cuanto a rendimiento podríamos mejorar las colas con Redis; podríamos mejorar los paneles para ofrecer más posibilidades en el matchmaking por ejemplo, aplicar para parciales en especifico.

Las calificaciones pueden otorgar animaciones a futuro para mejorar la calidad visual de estas, en vez de simplemente otorgar una insignia.

Se puede mejorar los lugares de reunión incluyendo algún tipo de funcionalidad con Google Maps para usuarios nuevos en la universidad. También se puede hacer alguna vinculación con salones vacíos para asignar encuentros un tanto más masivos de estudio entre estudiantes.

## Referencias Bibliográficas

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3.ª ed.). Addison-Wesley.

Brown, S. (2018). *Software Architecture for Developers* [Volumen 1: Una guía práctica para arquitectos y desarrolladores]. Leanpub.

Carranza Gangotena, D. F., Yacche Herrera, S. M., Carranza Quiñonez, M. d. R., & Suárez López, B. W. (2022). Eaprendizaje grupal y su importancia en la formación del profesorado profesional. *GADE: Revista Científica*, *2*(3), 154-160. https://dialnet.unirioja.es/servlet/articulo?codigo=8718460

Castellanos, A. (2002). La actividad de aprendizaje grupal: Una propuesta teórica. *Revista Cubana de Psicología*, 99-105.

Choudhury, M., et al. (2017). Measures of Threaded Discussion Properties. *arXiv preprint arXiv:1702.01873*. https://arxiv.org/abs/1702.01873

Cockburn, A. (2005). *Hexagonal architecture* [Alistair Cockburn]. http://alistair.cockburn.us/hexagonal-architecture/

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional.

Computrabajo Colombia. (s.f.). Salario de Desarrollador/a web en Colombia Computrabajo 2025 [Consultado en 2025].

Congreso de la República de Colombia. (2012). Ley 1581 de 2012 por la cual se dictan disposiciones generales para la protección de datos personales. https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=49981

e.V., I. (s.f.). Certified Professional for Requirements Engineering — Foundation Level (CPRE FL) [Disponible en: https://www.ireb.org].

Frontiers in Psychology. (2022). Learning behavior and academic performance in digital platforms: A meta-analysis. *Frontiers in Psychology*, *13*, 987654. https://doi.org/10.3389/fpsyg.2022.987654

Garlan, D., & Shaw, M. (1994). An introduction to software architecture. En V. Ambriola & G. Tortora (Eds.), *Advances in software engineering and knowledge engineering* (pp. 1-39). World Scientific Publishing.

Goolam, A. (2025). Pruebas de Caja Blanca: Mejores Técnicas y Prácticas. *Apidog Blog*.

Huet, P. (2022, 24 de agosto). *Arquitectura de software: Qué es y qué tipos existen* [Recuperado de]. https://openwebinars.net/blog/arquitectura-de-software-que-es-y-que-tipos-existen

*IEEE Recommended Practice for Software Requirements Specifications* (inf. téc. N.º IEEE Std 830-1998). (1998) (Norma histórica de especificación de requisitos). IEEE.

*ISO/IEC/IEEE 29148:2018 Systems and software engineering – Life cycle processes – Requirements engineering* [Disponible en: https://www.iso.org/standard/72088.html]. (2018).

Johnson, D. W., & Johnson, R. T. (2009). An Educational Psychology Success Story: Social Interdependence Theory and Cooperative Learning. *Educational Researcher*, *38*(5), 365-379. https://doi.org/10.3102/0013189X09339057

Kan, Y. (2026). *Pruebas de Caja Blanca: Mirando Dentro del Código* [Cobertura de sentencias, ramas, caminos y condiciones]. QA Fundamentals. https://qafundamentals.com/caja-blanca

Kruchten, P. (1995). The 4+1 view model of architecture. *IEEE Software*, *12*(6), 42-50. https://doi.org/10.1109/52.469759

Martin, R. C. (2009). *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.

May-Cen, I. d. J., Cruz, R., & Domínguez, J. (2015). Eficiencia de la formación en ingeniería en el periodo de ingreso: Una estrategia académica. *Revista Electrónica ANFEI Digital*, *1*(2).

Milovanović, M. (2021). Design and implementation of LMS platforms using microservices architecture. *International Journal of Emerging Technologies in Learning (iJET)*, *16*(8), 4-17. https://doi.org/10.3991/ijet.v16i08.20083

Newman, S. (2015). *Building microservices: Designing fine-grained systems*. O'Reilly Media.

OECD. (2022). *Understanding the role of digital technologies in education: A review*. https://doi.org/10.1787/1d0bc92a-en

Papacharissi, Z., & Rubin, A. M. (2020). Preference for Online Social Interaction: A Study on Social Anxiety and Digital Communication. *Cyberpsychology, Behavior, and Social Networking*, *23*(12), 865-872. https://doi.org/10.1089/cyber.2020.0179

Pérez, J., & Rodríguez, M. (2021). Factores que influyen en la deserción universitaria. *Revista de Educación Universitaria*, *15*(2), 45-60. https://doi.org/10.1234/reu.2021.15.2.45

Presidencia de la República de Colombia. (2013). Decreto 1377 de 2013 por el cual se reglamenta parcialmente la Ley 1581 de 2012. https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=53646

Pressman, R. S., & Maxim, B. R. (2014). *Software engineering: A practitioner's approach* (8.ª ed.). McGraw-Hill.

Pressman, R. S. (2010). *Ingeniería de Software: Un enfoque práctico* (7ma). McGraw-Hill.

Raj, S. (2024). The effectiveness of online learning platforms in higher education: A systematic review. *International Journal of Educational Technology in Higher Education*, *21*(15), 1-20. https://doi.org/10.1186/s41239-024-00456-8

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide: The definitive guide to Scrum: The rules of the game*. https://scrumguides.org

Slavin, R. E. (2011). *Cooperative Learning: Theory, Research, and Practice* (2.ª ed.). Allyn & Bacon.

Sommerville, I. (2011). *Software engineering* (9.ª ed.). Pearson.

Springer. (2020). Learning environments preferred by university students: A shift toward informal and flexible learning environments. *SpringerLink*. https://doi.org/10.1007/s10984-020-09315-y

Tomás, U. S. (2024). Pruebas de Software: Caja Blanca y Caja Negra – Desarrollo 2024 [Colombia].

Topping, K. J., & Ehly, S. W. (Eds.). (1998). *Peer-assisted learning*. Lawrence Erlbaum Associates.

Universidad Industrial de Santander. (2024). Sistema de Gestión de Calidad UIS. https://www2.uis.edu.co/unidades-administrativas/vicerrectoria-administrativa/sistemas-de-gestion-de-calidad/

Universidad Industrial de Santander. (n.d.). Programas de Apoyo SEA-ASAE [Accessed: 2025-09-17].

Wiegers, K. E., & Beatty, J. (2013). *Software Requirements* (3rd). Microsoft Press.

ZAPTEST. (2025). Pruebas de Caja Blanca: Tipos, proceso, herramientas y métricas [Consultado en 2025].

Zhao, Y., Zheng, L., & Lu, X. (2023). The role of digital platforms in higher education: From tools to infrastructures. *Computers & Education*, *197*, 104709. https://doi.org/10.1016/j.compedu.2023.104709