ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

Construcción de Software – GR2SW

PROYECTO SEGUNDO BIMESTRE

Integrantes:

Evelin Rocha

Profesor: Vicente Eguez.

Fecha de entrega: 23/01/2026

Construcción y evolución de Software.

Proyecto: Plataforma Multimedia Galeto

1. Introducción: Contexto, Problemática y Propósito

La plataforma **Galeto** surge como una solución tecnológica avanzada ante la creciente necesidad de espacios digitales que integren de manera cohesiva la gestión visual y la experiencia auditiva. En el panorama actual, los usuarios se enfrentan a plataformas fragmentadas; Galeto propone un ecosistema unificado donde la fotografía y la música convergen bajo estrictos estándares de seguridad, moderación y retroalimentación social dinámica.

El propósito fundamental de este despliegue técnico es demostrar cómo una arquitectura desacoplada, soportada por un ciclo de vida de desarrollo automatizado en Azure DevOps, puede evolucionar desde un módulo básico de autenticación hasta un sistema complejo de interacción en tiempo real. Este documento detalla la ingeniería detrás del carrusel dinámico, la seguridad basada en tokens y la gestión de requerimientos a través de cuatro sprints incrementales.

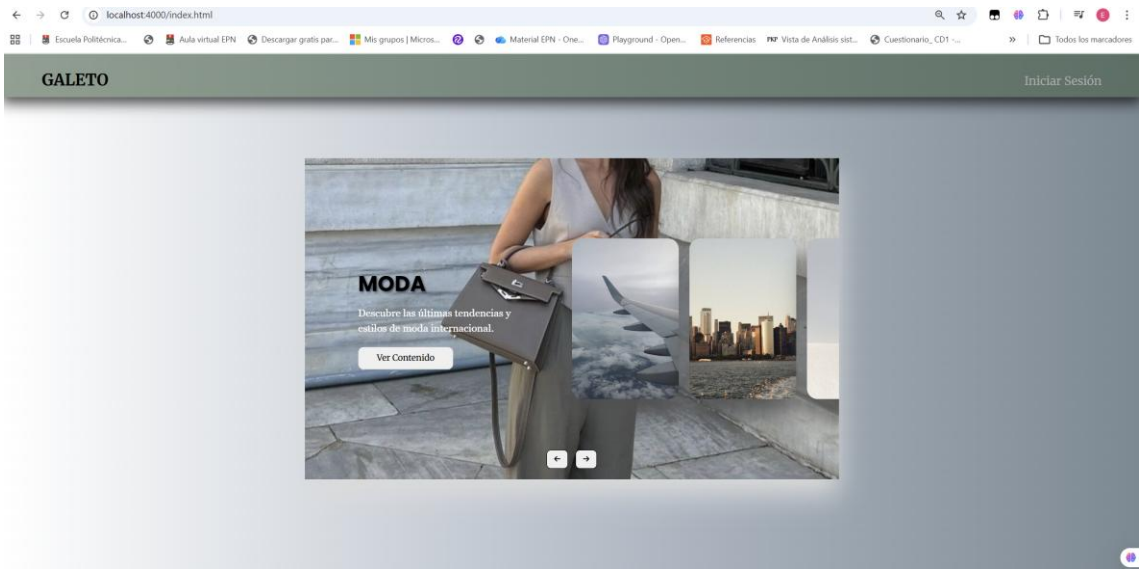

Ilustración 1. Ventana de inicio Galeto.

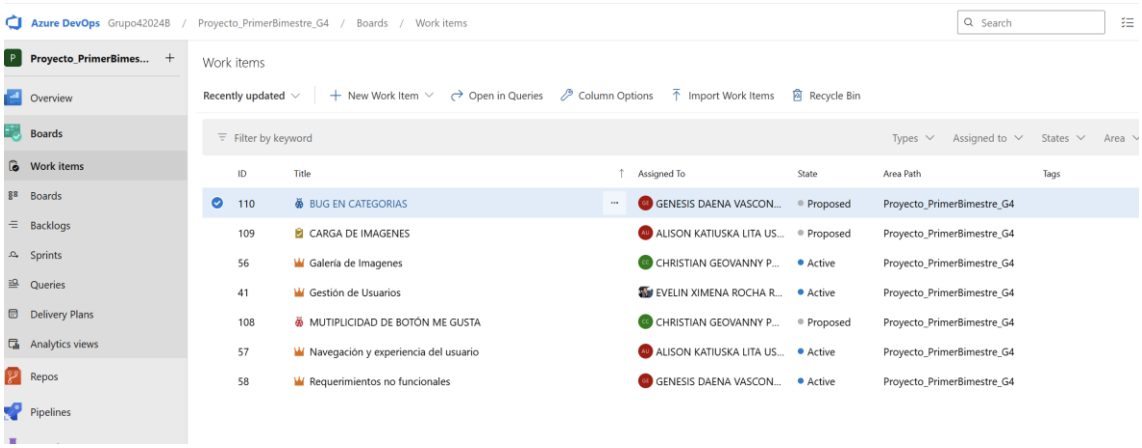

Ilustración 2. Gestión de Galeto en azureDevops..

2. Arquitectura del Proyecto y Estrategia de Integración

La robustez de Galeto reside en su arquitectura modular, diseñada para minimizar la deuda técnica y maximizar el rendimiento en el renderizado del lado del cliente.

2.1. Frontend: Ingeniería de Interacción y Manipulación del DOM

El frontend ha sido concebido bajo el paradigma de **JavaScript Vanilla (ES6+)**, priorizando la velocidad de carga y la eficiencia de los recursos:

- **Algoritmo de Carrusel Infinito (Sprint 3 - HU O5H9):** A diferencia de las soluciones convencionales que clonan elementos consumiendo memoria adicional, se desarrolló un algoritmo de rotación física de nodos. Al interactuar con los selectores .next o .prev, el sistema captura el nodo desbordado y lo reinserta mediante appendChild o prepend en el extremo opuesto del contenedor padre. Esto garantiza que el flujo multimedia sea perpetuo sin importar la densidad de la carga de datos.

```
let next = document.querySelector('.next')
let prev = document.querySelector('.prev')

next.addEventListener('click', function () {
    let items = document.querySelectorAll('.item')
    document.querySelector('.slide').appendChild(items[0])
})

prev.addEventListener('click', function () {
    let items = document.querySelectorAll('.item')
    document.querySelector('.slide').prepend(items[items.length - 1]) // here the length of items = 6
```

Ilustración 3. Implementación del carrusel.

- **Modularidad Estética y CSS Avanzado:** El sistema visual se basa en una jerarquía de selectores de alta especificidad. Se implementó una biblioteca de variables CSS para el manejo de la paleta de colores y tipografía (Poppins/Merriweather), asegurando que cualquier cambio de marca pueda ejecutarse modificando un solo archivo raíz, sin afectar la estructura del layout.

2.2. Capa de Seguridad, Identidad y API

- **Protocolo de Autenticación JWT (Sprint 1 - HU O11H6):** La seguridad no se delega únicamente al servidor. El frontend implementa un interceptor de lógica que gestiona **JSON Web Tokens**. Tras un login exitoso contra la API en localhost:4000, el token se almacena encriptado en el localStorage. Una función dedicada (decodeJWT) extrae el perfil del usuario para habilitar o restringir componentes visuales (como el botón de borrado para administradores) en milisegundos.

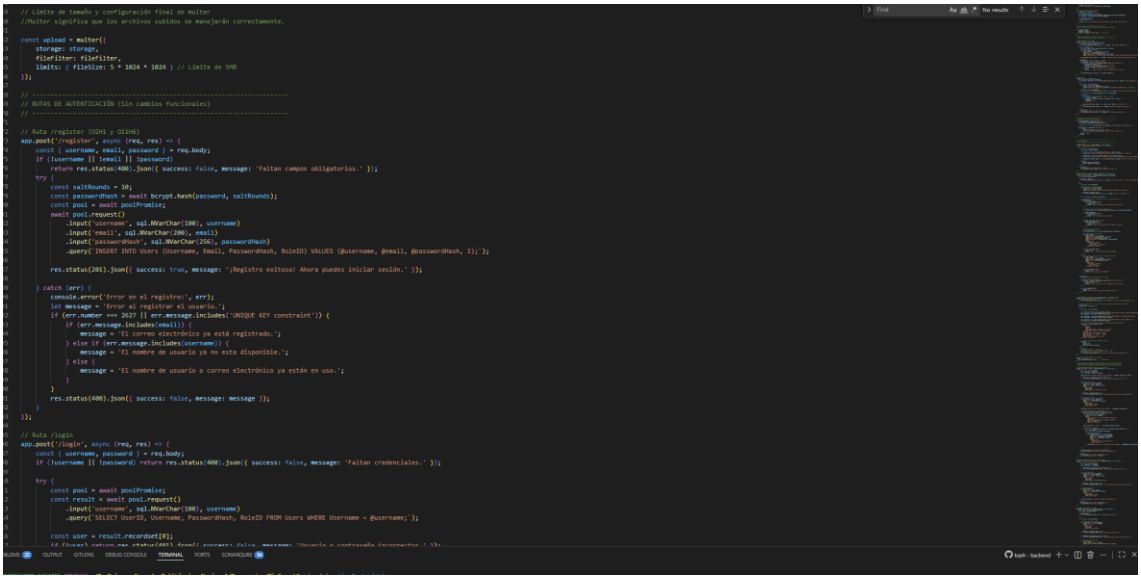

Ilustración 4. Implementación del protocolos de autenticación.

- **Filtros de Integridad en el Registro (Sprint 1 - HU O2H1):** Se diseñó un motor de validación basado en Expresiones Regulares (Regex) que actúa como primer muro de defensa. Este motor no solo valida la estructura sintáctica del email, sino que restringe el acceso a dominios de confianza (@gmail, @outlook, @hotmail), reduciendo drásticamente la creación de cuentas "spam".

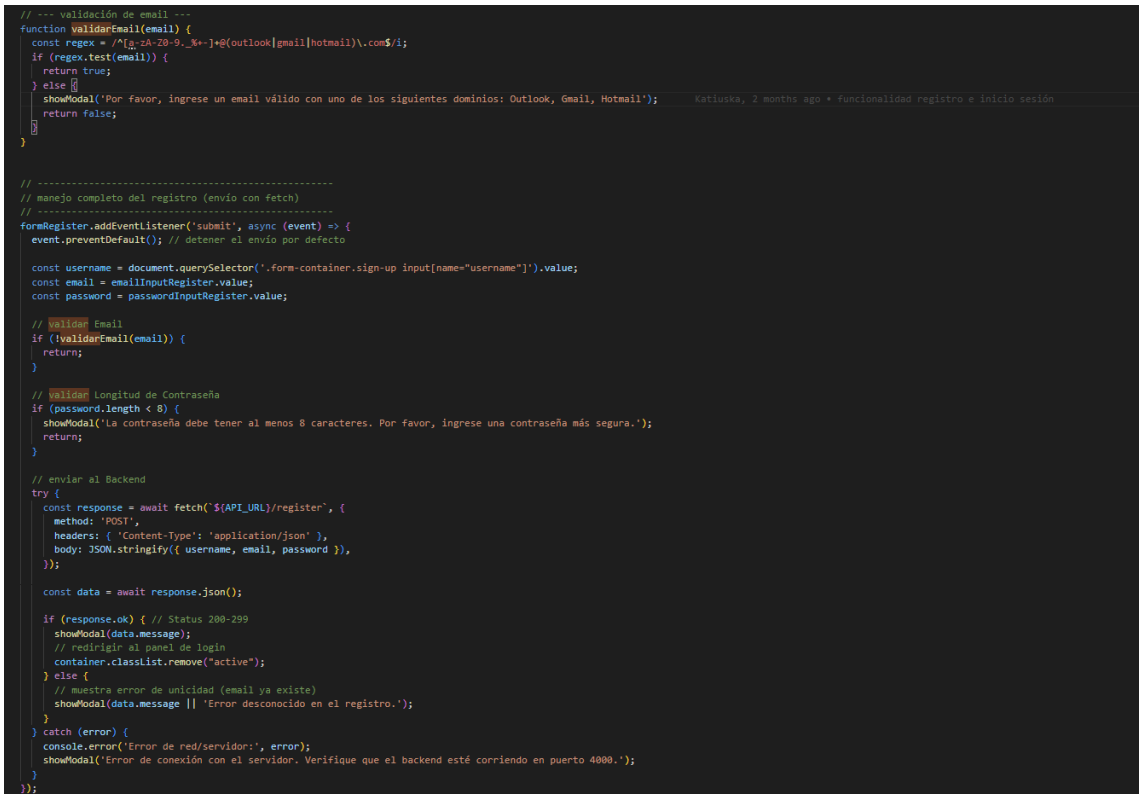

Ilustración 5. Implementación de la integridad del registro.

3. Estrategia de Pipelines y Ciclo de Vida (Azure Pipelines)

La evolución del software en Galeto se rige por un pipeline de **CI/CD** que garantiza que ningún código llegue a producción sin ser sometido a un riguroso escrutinio automatizado.

3.1. Integración Continua (CI): Calidad en el Origen

Cada vez que un desarrollador realiza un *commit* hacia **Azure Repos**, el pipeline ejecuta un flujo de trabajo de tres etapas:

1. **Análisis Estático (Linting):** Se audita el código CSS y JS para prevenir malas prácticas (como estilos "inline" o variables globales) que degraden el rendimiento.
2. **Pruebas de Validación Lógica:** Se ejecutan pruebas de caja negra sobre las funciones críticas (ej. validación de contraseña de 8 caracteres). Si una prueba falla, el build se rompe inmediatamente y se notifica al equipo.
3. **Gestión de Artefactos:** Se generan paquetes optimizados donde los scripts son minificados para reducir el ancho de banda necesario en el cliente final.

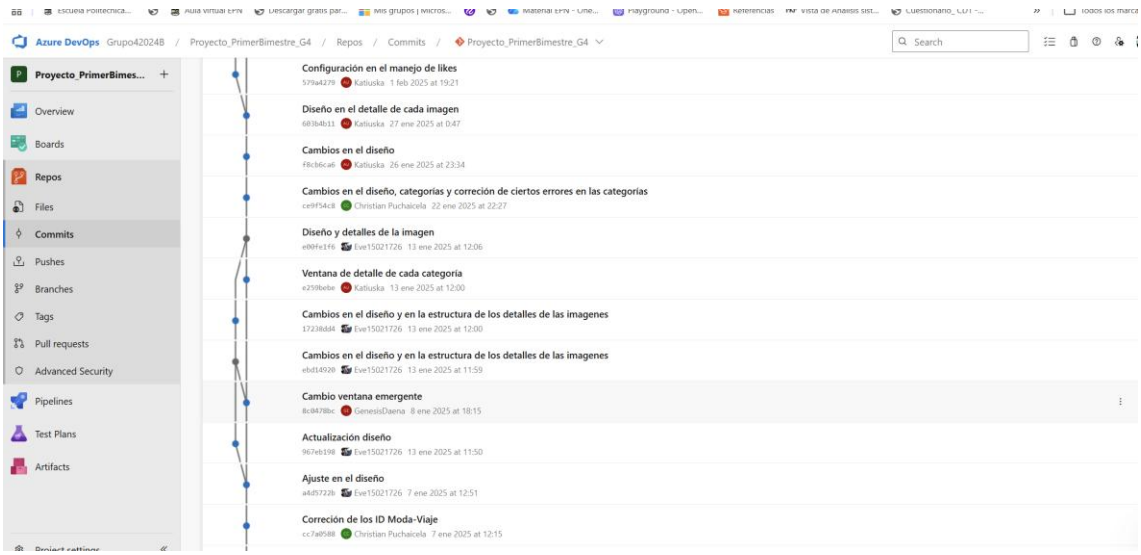

Ilustración 6. Integración continua.

3.2. Entrega Continua (CD): Despliegue Controlado

- **Ambientes de Staging:** Se utiliza un entorno espejo donde se prueban las funcionalidades multimedia antes del lanzamiento oficial. Aquí se validó la interactividad del Sprint 4.
- **Aprobaciones por Puertas (Approval Gates):** Se configuraron aprobaciones manuales obligatorias. Un líder técnico debe certificar visualmente que el diseño responsivo en 480px es funcional antes de que el pipeline libere la versión en la rama main.

4. Gestión de Historias de Usuario (Azure Boards)

La trazabilidad entre el negocio y el código es el pilar de **Azure Boards**. Cada ticket representa una necesidad real extraída de los documentos de Sprint:

Fase	Enfoque de Ingeniería	Historias de Usuario Vinculadas	Resultado Tangible
Sprint 1	**Seguridad Estructural**	O2H1, O6H3, O11H6	Registro seguro, login con JWT y perfil de usuario persistente.
Sprint 2	**Interacción y Socialización**	O7H4, O2H7, O4H8	Sistema de likes dinámico, asociación música-imagen y feed público.
Sprint 3	**Multimedia y UX**	O5H9, O7H15, O17H12	Ventanas emergentes de audio y sistema de votación blindado por ID.
Sprint 4	**Moderación y Feedback**	O7H17, O8H16, O10H17	Panel de notificaciones con estados de lectura y logout seguro.

6. Estrategia de Flujos de Desarrollo (Azure Repos)

Se adoptó un modelo de ramificación **GitFlow** avanzado para permitir el trabajo en paralelo de múltiples funcionalidades:

- **Feature Branching:** Cada funcionalidad se aisló en su propia rama. Por ejemplo, la rama feature/notificaciones permitió iterar sobre el panel dinámico sin poner en riesgo la estabilidad del login.
- **Pull Requests (PR) y Code Review:** Las fusiones a la rama develop requieren que el código sea revisado por otro desarrollador. Se busca optimización de

selectores, eliminación de código muerto y cumplimiento de la guía de estilos de Galeto.

- **Protección de Ramas:** La rama main es inviolable; solo permite fusiones mediante PRs que hayan superado exitosamente el Pipeline de CI.

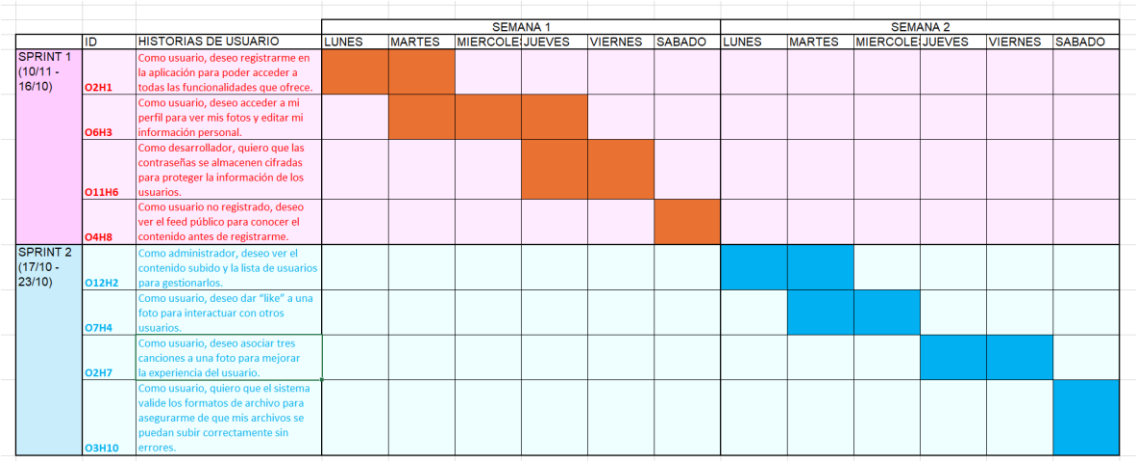

Ilustración 7. Planificación del sprint 1 y 2.

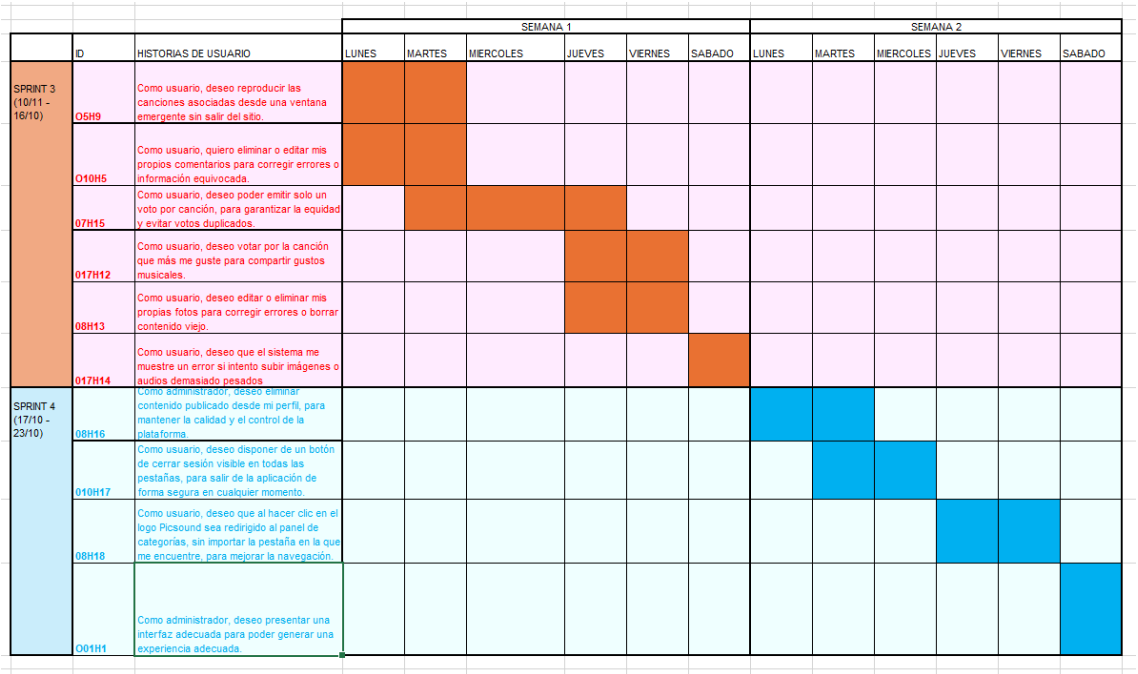

Ilustración 8. Planificación del sprint 3 y 4.

7. Control de Calidad, Revisiones y Rendimiento

El aseguramiento de la calidad (QA) en Galeto trasciende las pruebas básicas, enfocándose en la experiencia de usuario final:

1. **Auditoría de Rendimiento:** Se verificó que la manipulación del DOM en el carrusel no generara fugas de memoria (*memory leaks*), manteniendo un uso de CPU estable durante transiciones largas.
2. **Validación de UX Responsiva:** Se ejecutaron pruebas en diversos dispositivos móviles. El CSS se ajustó para que elementos críticos, como el panel de notificaciones (.notifications-panel), mantuvieran un scroll independiente y no bloquearan la navegación principal.
3. **Trazabilidad Boards-Repos:** Se implementó una política de commits donde cada cambio debe referenciar el ID del Work Item de Azure Boards. Esto permite realizar auditorías de código meses después y entender el motivo de cada cambio.

8. Herramientas, Conexiones y Evidencias Técnicas

El ecosistema de trabajo se orquestó mediante conexiones inteligentes:

- **Azure Boards ↔ Azure Repos:** Permite la vinculación automática; cada vez que se abre un PR, el estado de la Historia de Usuario cambia automáticamente a "En Revisión".
- **Postman ↔ API Testing:** Se integraron colecciones de Postman para validar los endpoints de la API (puerto 4000), garantizando que los datos JSON de las notificaciones se entreguen con la estructura correcta.
- **Anexos de Evidencia:** Se deben insertar aquí las capturas de los Sprints 1, 2, 3 y 4 que muestran los tableros de avance y las metas cumplidas.

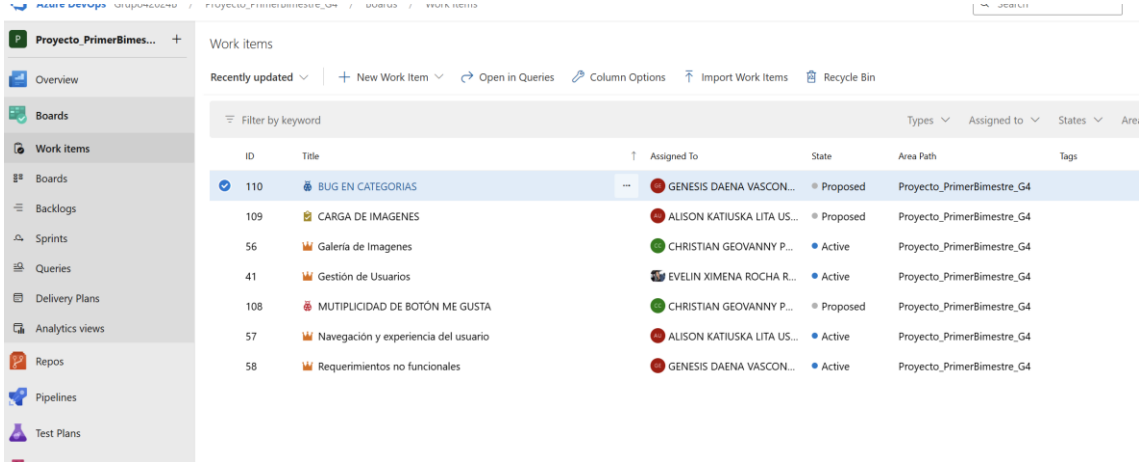

Ilustración 9. IWork items.

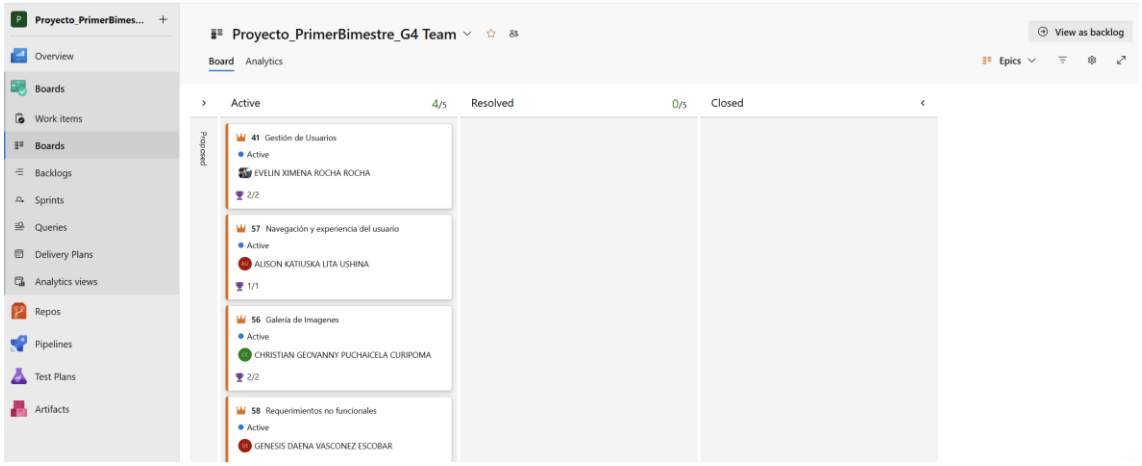

Ilustración 10. Board.

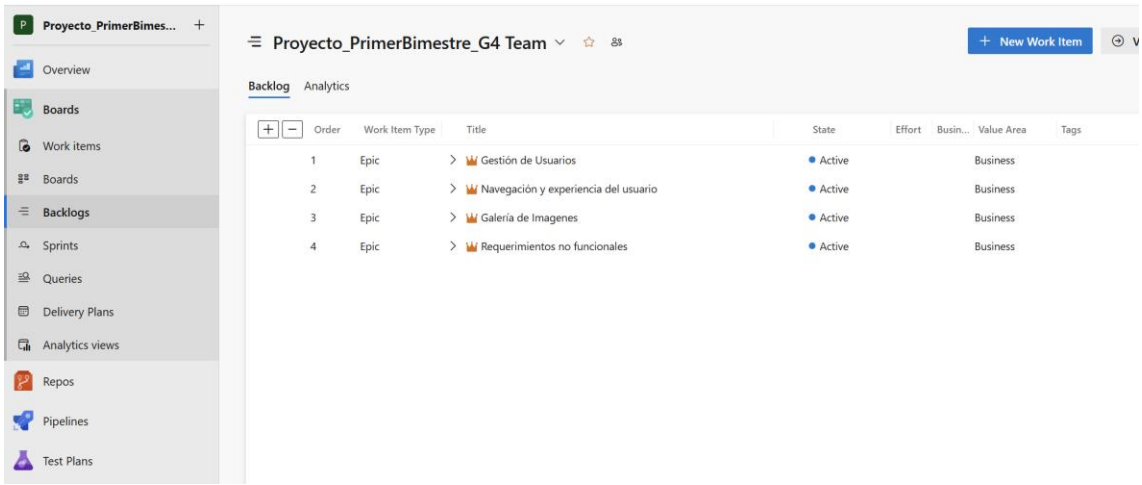

Ilustración 11. Backlogs.

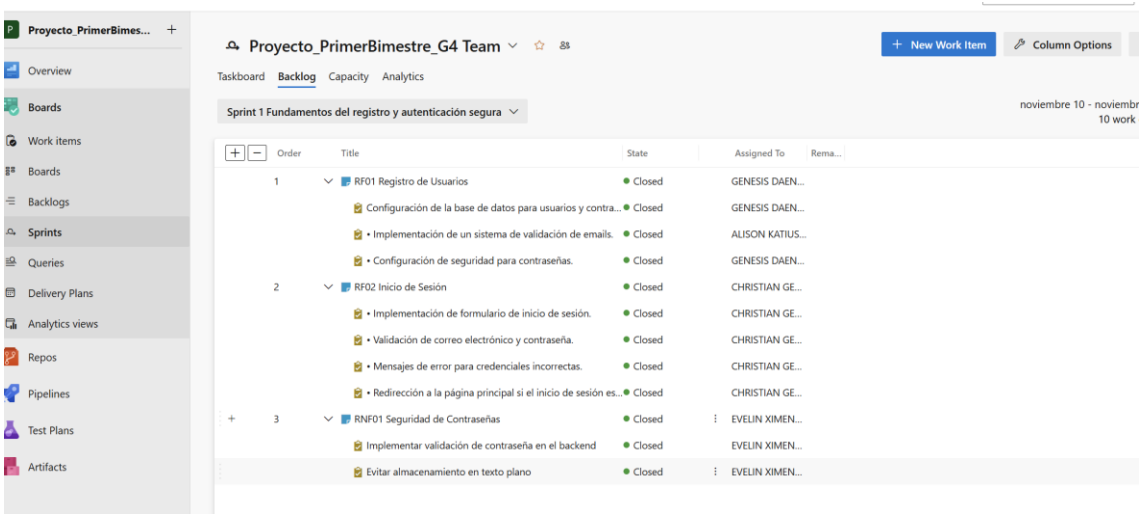

Ilustración 12. Sprint 1.

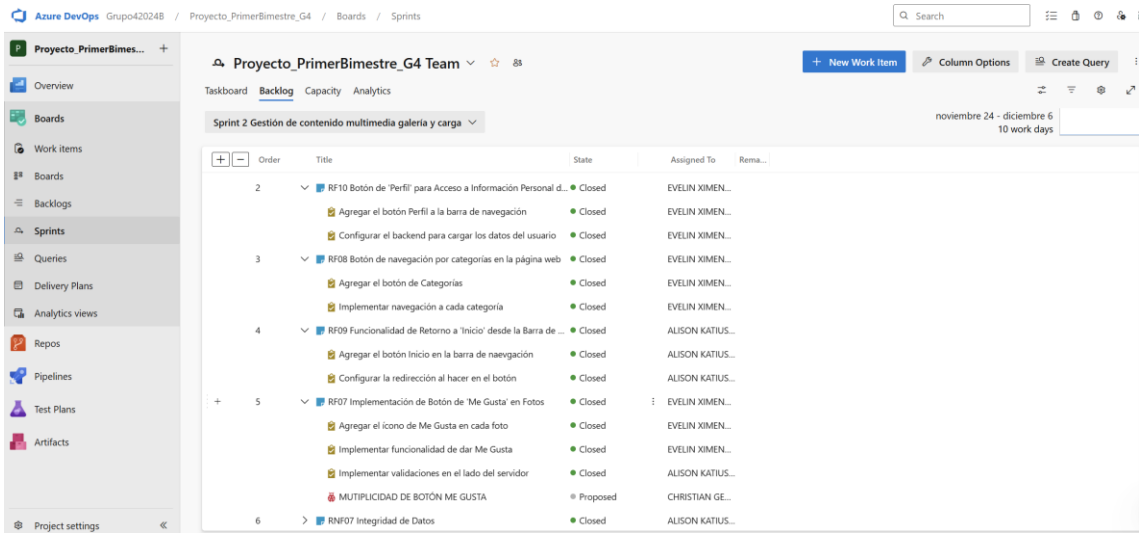

Ilustración 13. Sprint 2.

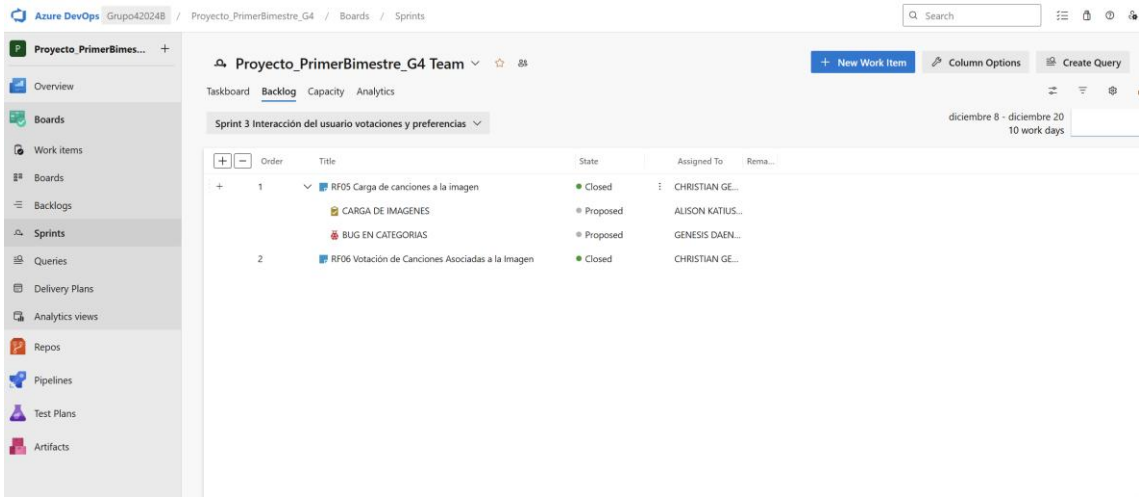

Ilustración 14. Sprint 3.

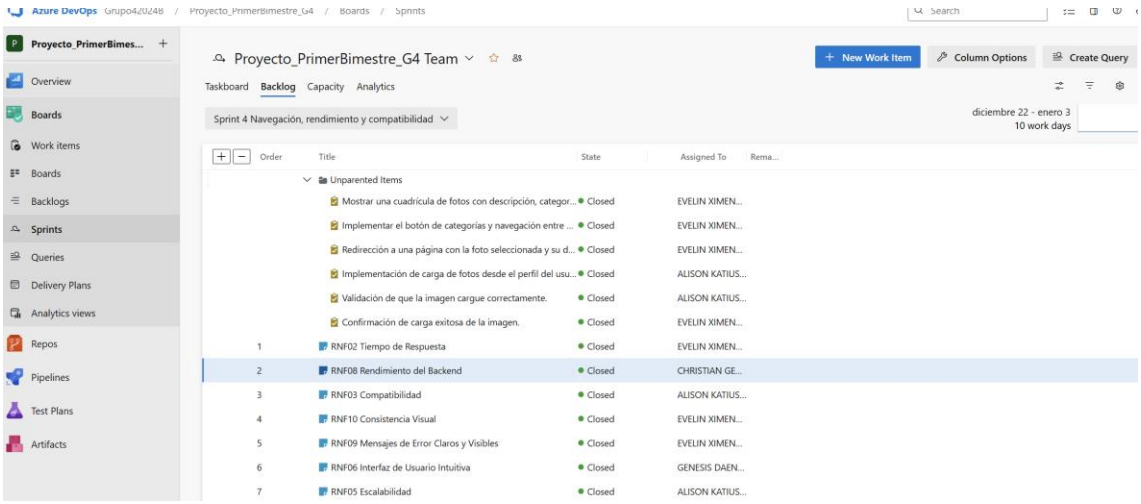

Ilustración 15. Sprint 4..

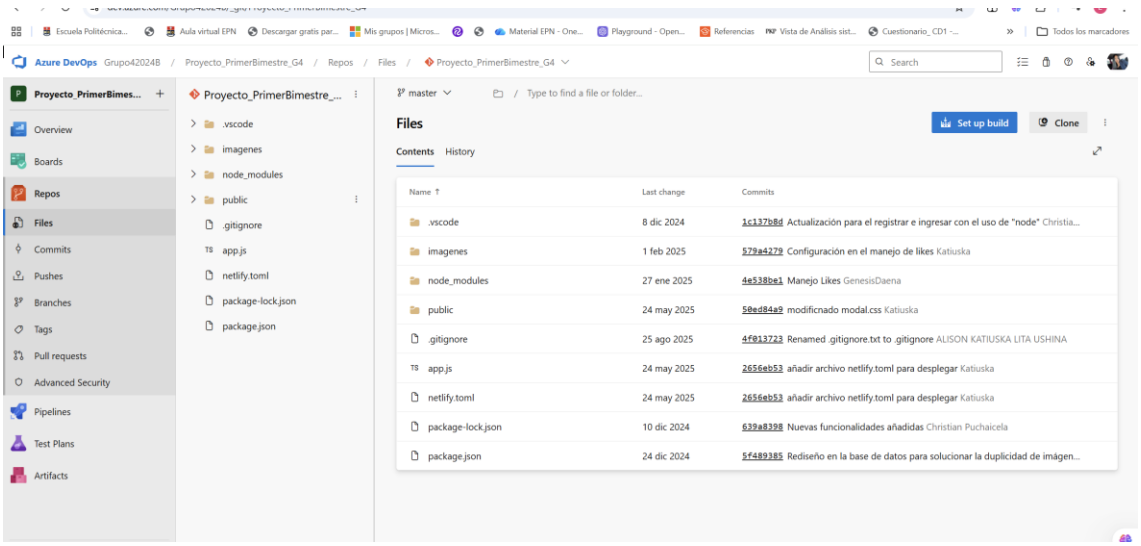

Ilustración 16. Repositorio.

9. Conclusiones y Visión Evolutiva del Software

La arquitectura técnica y la metodología aplicadas bajo el stack de Azure DevOps han transformado una idea conceptual en un producto de software robusto, escalable y mantenible. Galeto no solo cumple con las funcionalidades solicitadas en los cuatro sprints, sino que establece una base técnica sólida para futuras expansiones, como la integración de microservicios de streaming o sistemas de moderación automatizados por

IA. La trazabilidad lograda garantiza que la evolución del software sea transparente, segura y orientada siempre al valor del usuario final.