Documento de Arquitectura para la Kata: Revisa Tu Tarea

Valeria Rudas Ruiz

Mariana Cruz Gomez

Lilian Estefanía Maradiago Correa

Materia: Arquitectura de software

Profesor: Julio Robles



Universidad San Buenaventura

Facultad de Ingeniería

Programa Académico Ingeniería de Sistemas

Santiago de Cali, Colombia

2025

Contenido

[1. Contexto 3](#_Toc198745590)

[2. Identificar los Interesados 3](#_Toc198745591)

[3. Requerimientos Funcionales 3](#_Toc198745592)

[4. Requerimientos No Funcionales 5](#_Toc198745593)

[5. Atributos de Calidad 6](#_Toc198745594)

[6. Interesados VS Requerimientos No Funcionales (Ponderación) 6](#_Toc198745595)

[7. Los Atributos de Calidad más Importantes (Drivers Arquitectónicos) MAX=5 7](#_Toc198745596)

[8. Identificar las Tácticas para Atacar los Atributos de Calidad 7](#_Toc198745597)

[8.1. Protección de Datos y Trazabilidad 7](#_Toc198745598)

[8.2. Seguridad y Aislamiento 7](#_Toc198745599)

[8.3. Compatibilidad con Sistemas Externos 7](#_Toc198745600)

[8.4. Fiabilidad y Recuperación ante Fallos 7](#_Toc198745601)

[8.5. Adecuación Funcional y Validación 8](#_Toc198745602)

[9. Los Patrones Arquitectónicos que van a Utilizar 9](#_Toc198745603)

[9.1. Protección de Datos y Trazabilidad 9](#_Toc198745604)

[9.2. Seguridad y Aislamiento 9](#_Toc198745605)

[9.3. Compatibilidad con Sistemas Externos 9](#_Toc198745606)

[9.4. Fiabilidad y tolerancia al fallo 9](#_Toc198745607)

[9.5. Adecuación Funcional y Mantenibilidad 9](#_Toc198745608)

[10. Definir la Estrategia (Táctica con los Patrones Seleccionados) 10](#_Toc198745609)

[11. Generar las Vistas 10](#_Toc198745610)

[11.1. Escenarios Identificados 11](#_Toc198745611)

[11.2. Vista Lógica 11](#_Toc198745612)

[11.3. Vista de Implementación 11](#_Toc198745613)

[11.4. Vista de Procesos 11](#_Toc198745614)

[11.5. Vista Física 12](#_Toc198745615)

# Contexto

Kata: Revisa tu tarea

Una universidad que ha ampliado su oferta de cursos en ciencias de la computación busca automatizar la calificación de tareas de programación sencillas para atender a más de 300 estudiantes por año. El sistema debe permitir la subida, ejecución y calificación de código fuente de forma persistente y auditable. Además, debe incluir detección de plagio mediante comparación entre entregas y el servicio externo de TurnItIn, integrarse con el sistema de gestión del aprendizaje (LMS) de la universidad, y respetar fechas límite definidas por los profesores. Todo esto debe desarrollarse en un contexto con bajo presupuesto para el área de sistemas y con un LMS basado en un mainframe difícil de modificar.

# Identificar los Interesados

* **Estudiantes**

Son los usuarios principales del sistema. Suben sus tareas de programación, reciben calificaciones automáticas y pueden enviar múltiples intentos antes de la fecha límite.

* **Profesores**

Definen las tareas, criterios de evaluación y fechas de entrega. Usan el sistema para revisar los resultados, configurar pruebas automáticas y detectar plagio.

* **Personal administrativo**

Se encarga de la auditoría de notas y reportes académicos. Necesita acceso a los historiales de calificaciones y entregas para cumplir con regulaciones estatales.

* **Equipo TI**

Encargado del desarrollo, mantenimiento e integración del sistema con la infraestructura tecnológica existente, incluyendo el LMS basado en mainframe.

* **Entidades estatales de auditoría**

Revisan anualmente las calificaciones y procesos académicos. No usan directamente el sistema, pero dependen de que este sea auditable y confiable.

* **Proveedor de servicio externo (TurnItIn)**

Provee TurnItIn como herramienta para la detección de plagio. El sistema debe conectarse a sus servicios para complementar la validación de originalidad del código.

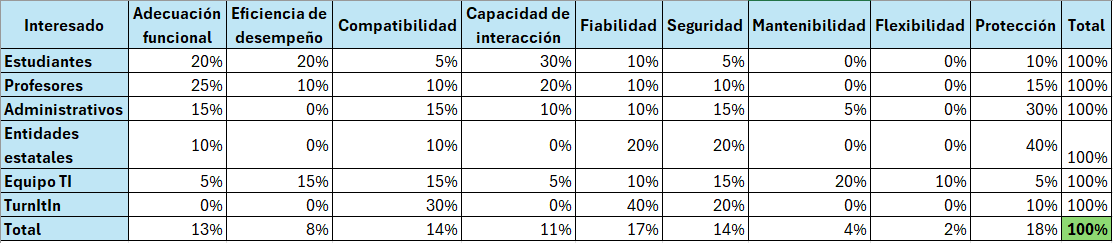
# Requerimientos Funcionales

|  |  |  |  |
| --- | --- | --- | --- |
| Código | Título | Detalle | Interesados |
| RF01 | Subida de código | El estudiante debe poder subir su código fuente para una tarea específica. | Estudiantes |
| RF02 | Ejecución del código | El sistema debe ejecutar el código en un entorno controlado y mostrar los resultados. | Estudiantes, Profesores |
| RF03 | Calificación automática | El sistema debe calificar el código según criterios definidos (pruebas, métricas). | Profesores, Estudiantes |
| RF04 | Historial de envíos | Cada envío debe guardarse con fecha, hora, código enviado y resultado. | Estudiantes, Profesores, Administrativos |
| RF05 | Múltiples intentos | El estudiante puede enviar varias versiones de su tarea antes de la fecha límite. | Estudiantes |
| RF06 | Gestión de fechas límite | El profesor puede definir una fecha y hora límite para cada tarea. | Profesores |
| RF07 | Rechazo de envíos fuera de plazo | El sistema debe rechazar entregas después de la fecha límite. | Estudiantes, Profesores |
| RF08 | Comparación entre entregas (plagio) | El sistema debe comparar entregas entre estudiantes para detectar similitudes. | Profesores, Administrativos |
| RF09 | Integración con TurnItIn | El sistema debe conectarse con TurnItIn para verificar plagio externo. | Profesores, Proveedor de TurnItIn |
| RF10 | Integración con LMS | El sistema debe enviar calificaciones y datos al LMS de la universidad. | Profesores, Administrativos, Sistemas |
| RF11 | Configuración de criterios de nota | El profesor puede definir pruebas o métricas como parte de la calificación. | Profesores |
| RF12 | Consulta de notas | Los estudiantes y profesores deben poder ver las notas y su historial. | Estudiantes, Profesores |
| RF13 | Generación de reportes auditables | El sistema debe generar reportes que permitan auditoría anual de notas y entregas. | Administrativos, Entidades estatales |

# Requerimientos No Funcionales

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Código | Título | Detalle | Métrica | Interesados |
| RNF01 | Seguridad de ejecución | El código enviado por estudiantes debe ejecutarse en un entorno aislado (sandbox). | Ninguna ejecución debe afectar el servidor o acceder a archivos externos | Equipo de sistemas, Estudiantes |
| RNF02 | Trazabilidad | Cada acción (envío, calificación, modificación) debe registrarse con fecha y usuario. | 100% de acciones relevantes registradas en logs | Administrativos, Entidades estatales |
| RNF03 | Rendimiento del sistema | El sistema debe procesar y calificar entregas de forma eficiente, incluso en picos de uso. | Tiempo de respuesta promedio < 5 segundos por envío | Estudiantes, Profesores |
| RNF04 | Alta disponibilidad | El sistema debe estar disponible durante los periodos de entrega de tareas. | Uptime ≥ 99% durante semanas de entrega | Estudiantes, Profesores |
| RNF05 | Escalabilidad | El sistema debe soportar el crecimiento anual de estudiantes sin pérdida de rendimiento. | Soporte de al menos 500 usuarios concurrentes sin fallos | Equipo de sistemas |
| RNF06 | Compatibilidad con LMS | La integración con el LMS debe funcionar sin necesidad de modificar el sistema heredado. | 100% de datos sincronizados sin intervención manual | Equipo de sistemas, Administrativos |
| RNF07 | Usabilidad | La interfaz debe ser simple y clara para usuarios no técnicos. | 85% de satisfacción en pruebas de usabilidad | Estudiantes, Profesores |
| RNF08 | Integridad de datos | Ninguna entrega ni calificación debe perderse o alterarse sin trazabilidad. | 0% de pérdida o modificación no autorizada de datos | Administrativos, Entidades estatales |
| RNF09 | Interoperabilidad | El sistema debe comunicarse correctamente con TurnItIn y otros servicios externos. | 100% de envíos compatibles y respuestas válidas desde servicios externos | Profesores, Administrativos, Sistemas |
| RNF10 | Bajo consumo de recursos | El sistema debe operar con recursos limitados del área de TI. | CPU y memoria dentro del 70% de uso promedio en servidores disponibles | Equipo de sistemas |

# Atributos de Calidad



# Interesados VS Requerimientos No Funcionales (Ponderación)

Tabla

El contenido generado por IA puede ser incorrecto.

# Los Atributos de Calidad más Importantes (Drivers Arquitectónicos) MAX=5

Texto

El contenido generado por IA puede ser incorrecto.

# Identificar las Tácticas para Atacar los Atributos de Calidad

## Protección de Datos y Trazabilidad

El sistema implementará un subsistema de auditoría que registrará todas las acciones críticas mediante logs estructurados en formato JSON, incluyendo metadatos obligatorios (timestamp, identificación de usuario, recurso afectado y tipo de operación). Para garantizar la integridad de la información, se aplicarán funciones hash (SHA-256) a las entregas y calificaciones, almacenándose estos valores en la base de datos. Adicionalmente, se adoptará un control de versiones inspirado en Git, permitiendo rastrear cambios entre envíos sucesivos de un mismo estudiante.

## Seguridad y Aislamiento

La ejecución de código estudiantil se realizará en entornos aislados mediante contenedores Docker efímeros, configurados con restricciones estrictas: sin acceso a red o archivos del host, y límites de recursos (CPU/memoria). La autenticación seguirá el estándar OAuth 2.0 con tokens JWT, integrados con el LMS existente, y se implementará un modelo RBAC con roles predefinidos (estudiante, profesor, administrador). Todos los datos sensibles se encriptarán en tránsito (TLS 1.3) y en reposo (AES-256).

## Compatibilidad con Sistemas Externos

Para la interoperabilidad con el LMS basado en mainframe y el servicio TurnItIn, se desarrollará una capa de adaptación que traduzca formatos heredados (XML/CSV) a modelos internos mediante el Adapter Pattern. Las comunicaciones utilizarán APIs REST/JSON documentadas con OpenAPI 3.0, y se implementarán colas de mensajería (RabbitMQ) para manejar fallos temporales en integraciones externas.

## Fiabilidad y Recuperación ante Fallos

El sistema incorporará mecanismos de resiliencia, incluyendo reintentos automáticos con backoff exponencial (hasta 3 intentos) para operaciones con servicios externos. Se configurará un patrón Circuit Breaker para desactivar temporalmente integraciones ante fallos recurrentes. Para garantizar la disponibilidad de datos, se replicará la base de datos en un servidor secundario con sincronización asíncrona.

## Adecuación Funcional y Validación

Las funcionalidades críticas, como la evaluación automática, se someterán a pruebas unitarias e integrales con una cobertura mínima del 80%. Los profesores podrán configurar límites de intentos por tarea (ej: 5 envíos máximos), y el sistema validará automáticamente cada entrega contra los criterios definidos, almacenando resultados en un historial accesible y auditable.

|  |  |
| --- | --- |
| **Atributo de Calidad** | **Tácticas Aplicadas** |
| **Protección** | - Registro de auditoría detallado (logging de acciones críticas). - Validación de integridad mediante checksum o hash.  - Control de versiones de entregas. |
| **Seguridad** | - Ejecución de código en entornos sandbox.  - Autenticación robusta y control de accesos basado en roles (RBAC). - Encriptación de datos sensibles en tránsito y en reposo. |
| **Compatibilidad** | - Uso de APIs estandarizadas (REST/JSON).  - Adaptadores para comunicarse con el LMS y TurnItIn.  - Tolerancia a protocolos heredados mediante capas de abstracción. |
| **Fiabilidad** | - Registro de errores y monitoreo continuo.  - Mecanismos de retry automático ante fallos temporales.  - Redundancia básica y respaldo de datos. |
| **Adecuación funcional** | - Validación automática contra criterios de evaluación definidos por profesores.  - Testing extensivo de funcionalidades clave.  - Soporte para múltiples intentos con control de versiones. |

# Los Patrones Arquitectónicos que van a Utilizar

Para garantizar el cumplimiento de los atributos de calidad definidos como críticos para el sistema, se han establecido los siguientes patrones arquitectónicos, fundamentados en los requisitos técnicos y las restricciones presupuestarias definidas.

## Protección de Datos y Trazabilidad

Se implementará una arquitectura por capas que aísle específicamente el subsistema de auditoría del núcleo funcional principal. El patrón Broker actuará como intermediario para el registro de eventos críticos, garantizando que los mecanismos de trazabilidad no impacten el rendimiento del sistema principal. Esta combinación permite mantener la integridad de los datos de auditoría mientras se minimiza el acoplamiento con otras componentes del sistema, facilitando además el cumplimiento de requisitos regulatorios.

## Seguridad y Aislamiento

El núcleo del sistema se reducirá a funcionalidades esenciales, delegando la ejecución segura de código a plugins manejados como servicios independientes (sandbox). La autenticación y autorización se centralizarán en un servidor dedicado. El enfoque de microkernel limita la superficie de ataque y permite actualizar componentes de seguridad sin modificar el núcleo, mientras que el modelo cliente-servidor simplifica la gestión de accesos.

## Compatibilidad con Sistemas Externos

Se desarrollará adaptadores específicos para cada sistema externo (LMS mainframe y TurnItIn), incluyendo transformación de formatos y protocolos. Este patrón (Adapter Pattern) provee una solución eficiente y de bajo costo para integrar sistemas legacy sin requerir modificaciones en su código base, cumpliendo con la restricción de no alterar el LMS existente.

## Fiabilidad y tolerancia al fallo

El sistema generará eventos para notificar estados críticos y fallos, mientras que el patrón Circuit Breaker protegerá las integraciones externas de fallos en cascada. Esta combinación permite una respuesta ágil a incidentes mientras previene la saturación del sistema ante fallos persistentes en servicios externos.

## Adecuación Funcional y Mantenibilidad

La arquitectura separará claramente la lógica de negocio (Modelo), la interfaz de usuario (Vista) y el control de flujo (Controlador). El patrón MVC facilita el desarrollo iterativo, las pruebas automatizadas y la mantenibilidad - factores críticos dado el contexto de bajo presupuesto y necesidad de evolucionar el sistema.

|  |  |
| --- | --- |
| **Atributo de Calidad** | **Patrones Arquitectónicos Seleccionados** |
| **Protección** | - **Layered Pattern** (para aislar lógica de auditoría).  - **Broker Pattern** (para desacoplar auditoría e informes del núcleo funcional). |
| **Seguridad** | - **Microkernel Pattern** (sandbox como plugin independiente).  - **Client-Server Pattern** (seguridad centralizada en el servidor). |
| **Compatibilidad** | - **Adapter Pattern** (a nivel de integración externa). |
| **Fiabilidad** | - **Event-Driven Pattern** (para detectar y reaccionar a fallos). - **Circuit Breaker** (para prevención de fallos en cascada) |
| **Adecuación funcional** | - **Model-View-Controller (MVC)** (modularidad y separación de funciones). |

# Definir la Estrategia (Táctica con los Patrones Seleccionados)

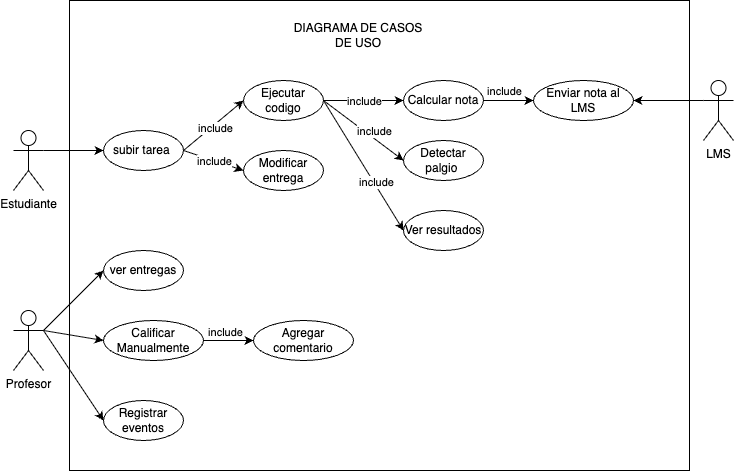
La arquitectura del sistema se diseñará aplicando un **enfoque mixto entre patrón por capas (Layered) y microservicios**, permitiendo modularizar componentes críticos (como evaluación automática, auditoría, integración con LMS, y sandbox de ejecución). Cada componente aplicará tácticas específicas:

* **Para protección**, se usará una capa de auditoría que registre todas las acciones importantes con timestamp y usuario, y se mantendrá separada mediante el patrón **Broker**.
* **Para seguridad**, se combinarán sandboxing (táctica) y un microkernel que permita cargar mecanismos de evaluación de forma segura, evitando interferencias entre módulos.
* **Para compatibilidad**, se usarán adaptadores como parte del patrón **Broker**, que actúa como mediador de servicios externos.
* **Para fiabilidad**, el sistema seguirá un enfoque **event-driven** para manejar errores y recuperación, y empleará retry automático para operaciones con terceros.
* **Para adecuación funcional**, se seguirá el patrón **MVC**, facilitando pruebas unitarias, gestión de intentos, y flexibilidad para cambios en los criterios de evaluación.

# Generar las Vistas

Los diagramas presentados a continuación están disponibles en el siguiente enlace: <https://drive.google.com/file/d/1-pc4hQOwrsD3WC5pLYZY5oVhVPrAOtKL/view?usp=sharing>

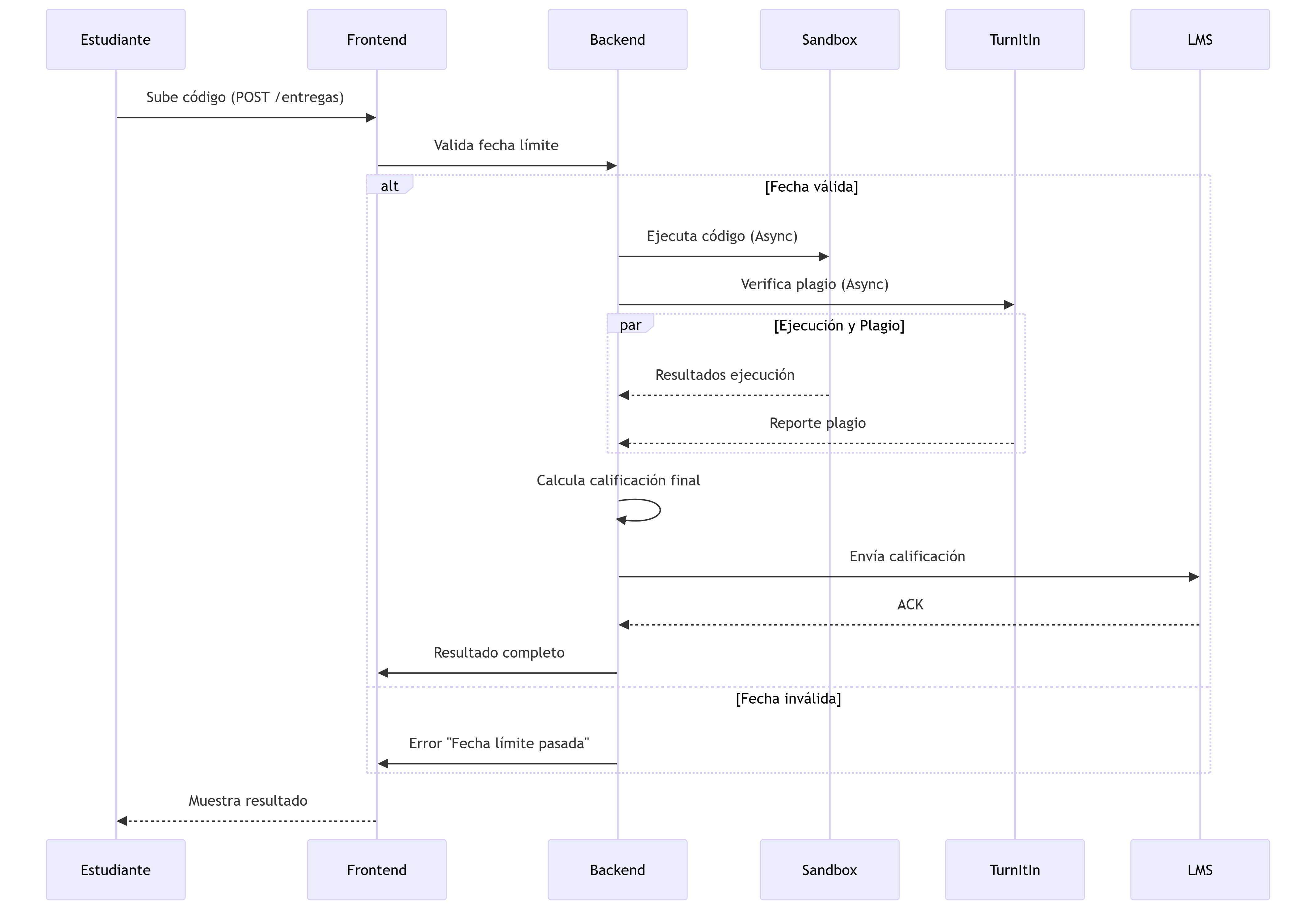
## Escenarios Identificados



## Vista Lógica

## Vista de Implementación

## Vista de Procesos



## Vista Física