ACTIVIDAD 5

Estrategia de Control de Calidad y Pruebas Unitarias

Garantizar la calidad del software es un aspecto fundamental en el desarrollo de cualquier aplicación, especialmente en una plataforma de gestión de proyectos basada en inteligencia artificial. Para ello, se define una estrategia de control de calidad basada en pruebas automatizadas, revisiones de código y herramientas de análisis estático que permitan detectar errores de manera temprana y reducir la deuda técnica.

Implementación de Pruebas Unitarias

Las pruebas unitarias serán el primer nivel de validación del software. Para su implementación, se utilizará **JUnit**, uno de los frameworks más utilizados en Java. Estas pruebas permitirán verificar de manera aislada que cada módulo y componente cumple correctamente con su funcionalidad esperada. La cobertura de estas pruebas será una métrica clave para evaluar la robustez del código, asegurando que la mayor parte del sistema esté protegida contra errores imprevistos.

Cada prueba unitaria debe ser **independiente y repetible**, evitando dependencias con bases de datos o servicios externos. Para lograr esto, se emplearán **dobles de prueba** como mocks y stubs mediante herramientas como **Mockito**, permitiendo simular comportamientos de dependencias externas sin afectar el rendimiento ni la fiabilidad de las pruebas.

Además, se aplicará el principio **AAA (Arrange, Act, Assert)** en la estructuración de las pruebas unitarias, asegurando claridad y mantenimiento en los test. Este principio se basa en organizar los datos de entrada y el estado inicial (Arrange), ejecutar la funcionalidad a probar (Act) y verificar que el resultado obtenido es el esperado (Assert).

Integración Continua y Automatización de Pruebas

Para garantizar que los cambios en el código no introducen regresiones, se establecerá un sistema de integración continua mediante herramientas como **Jenkins, GitHub Actions o GitLab CI/CD**. Esto permitirá ejecutar automáticamente la batería de pruebas cada vez que un desarrollador realice un commit en el repositorio, detectando errores de forma temprana y evitando que defectos se acumulen en la base del código.

Además de las pruebas unitarias, se integrarán pruebas de regresión automatizadas para asegurar que nuevas funcionalidades no alteran el correcto funcionamiento de las existentes. Estas pruebas podrán ejecutarse en entornos aislados utilizando contenedores con **Docker**, lo que permitirá replicar con precisión diferentes configuraciones de ejecución y detectar fallos específicos de cada entorno.

También se incorporarán pruebas funcionales con herramientas como **Selenium** para evaluar la interacción del usuario con la interfaz, garantizando que la plataforma cumpla con los requisitos de usabilidad y experiencia de usuario.

Revisiones de Código y Análisis Estático

Un aspecto fundamental en la estrategia de control de calidad será la realización de **revisiones de código periódicas** en las que los desarrolladores del equipo analizarán las contribuciones de sus compañeros. Esta práctica no solo ayuda a detectar errores y mejorar la calidad del código, sino que también promueve el aprendizaje continuo y la difusión de buenas prácticas dentro del equipo.

Para complementar estas revisiones manuales, se emplearán herramientas de análisis estático como **SonarQube** o **Checkstyle**, que permitirán detectar patrones de código problemáticos, vulnerabilidades de seguridad y violaciones de principios de diseño como S.O.L.I.D. y DRY. Estas herramientas ayudarán a mantener un código limpio, modular y fácil de mantener, reduciendo la deuda técnica y evitando errores difíciles de corregir en fases avanzadas del desarrollo.

Se implementará una guía de estilos y convenciones de código basada en estándares de la industria para asegurar consistencia en la escritura del código. Esto contribuirá a facilitar la colaboración entre desarrolladores y reducir la probabilidad de errores humanos derivados de discrepancias en el formato del código.

Reducción de la Deuda Técnica y Beneficios

El conjunto de prácticas descritas permitirá mitigar la acumulación de deuda técnica, que ocurre cuando se prioriza la velocidad en el desarrollo a costa de la calidad del código. Mediante pruebas automatizadas, revisiones periódicas y un enfoque en la calidad desde las primeras etapas del desarrollo, se garantizará que la plataforma sea escalable y mantenible a largo plazo.

Además, estas estrategias no solo mejorarán la calidad del software, sino que también optimizarán el proceso de desarrollo. Al detectar errores temprano, se reduce el tiempo necesario para corregirlos en etapas posteriores, disminuyendo costos y mejorando la eficiencia del equipo.

En conclusión, la estrategia de control de calidad propuesta asegurará que la plataforma cumpla con altos estándares de fiabilidad y mantenibilidad. La combinación de pruebas unitarias, integración continua, análisis estático y revisiones de código permitirá desarrollar un producto sólido, evitando problemas derivados de una mala calidad en el código y facilitando su evolución futura.

Se fomentará una cultura de calidad dentro del equipo de desarrollo, incentivando la adopción de metodologías ágiles y asegurando que cada entrega de software sea evaluada desde una perspectiva técnica y funcional antes de ser desplegada en producción. Además, se realizarán reuniones periódicas para revisar la efectividad de las estrategias implementadas y proponer mejoras según las necesidades cambiantes del proyecto. Esto garantizará que el equipo se mantenga alineado con los objetivos de calidad y que el software evolucione de manera sostenible en el tiempo.