11. Levantamiento de información (herramientas y conclusiones)

1. SonarQube

Tipo: Análisis estático de código (SAST).

Características: Detecta bugs, vulnerabilidades, code smells y deuda técnica.

Ventajas: Open-source en su versión Community, soporta múltiples lenguajes, integración sencilla con pipelines CI/CD.

Desventajas: Configuración inicial compleja, requiere servidores con recursos adecuados.

Aplicación: Ideal para detectar problemas de seguridad y calidad en el código fuente antes de compilar.

2. OWASP ZAP

Tipo: Análisis dinámico de seguridad (DAST).

Características: Escáner especializado en vulnerabilidades de aplicaciones web.

Ventajas: Gratuito, mantenido por la comunidad OWASP, interfaz intuitiva.
Desventajas: Puede generar falsos positivos y requiere experiencia en ciberseguridad para interpretar resultados.

Aplicación: Detecta vulnerabilidades de seguridad OWASP Top 10 (XSS, SQLi, CSRF, etc.) durante pruebas dinámicas.

3. Snyk

Tipo: Análisis de dependencias y contenedores.

Características: Escaneo en tiempo real de librerías de terceros, imágenes Docker y configuraciones.

Ventajas: Base de datos de vulnerabilidades muy actualizada, integración directa con repositorios GitHub/GitLab.

Desventajas: El plan gratuito es limitado y los costos de la versión empresarial son altos.

Aplicación: Permite prevenir despliegues con librerías inseguras o configuraciones mal implementadas.

4. Jenkins

Tipo: Integración y despliegue continuo (CI/CD).

Características: Automatiza la compilación, ejecución de pruebas y despliegues en diferentes entornos.

Ventajas: Open-source, gran cantidad de plugin, adaptable a distintos proyectos.
Desventajas: Puede ser complejo de mantener en proyectos grandes y con muchas integraciones.

Aplicación: Orquesta herramientas como SonarQube, Snyk y Selenium dentro del pipeline de desarrollo.

5. Selenium

Tipo: Automatización de pruebas funcionales.

Características: Simula interacciones de usuario en navegadores, compatible con múltiples lenguajes.

Ventajas: Open-source, muy flexible, amplia comunidad de soporte.

Desventajas: Requiere mantenimiento constante de scripts, limitado para pruebas de Backend.

Aplicación: Automatiza pruebas de interfaz gráfica (UI) para validar la funcionalidad del sistema antes de producción.

📌 Conclusiones del Levantamiento de Herramientas

El análisis realizado muestra que cada herramienta cumple un rol específico dentro del ciclo de vida del software:

SonarQube y Snyk permiten detectar vulnerabilidades de manera temprana en el código y en las dependencias.

OWASP ZAP aporta seguridad en la etapa de pruebas dinámicas, evaluando la aplicación en ejecución.

Jenkins y Selenium garantizan la automatización y la continuidad del proceso, evitando que las pruebas dependan únicamente de la intervención manual.

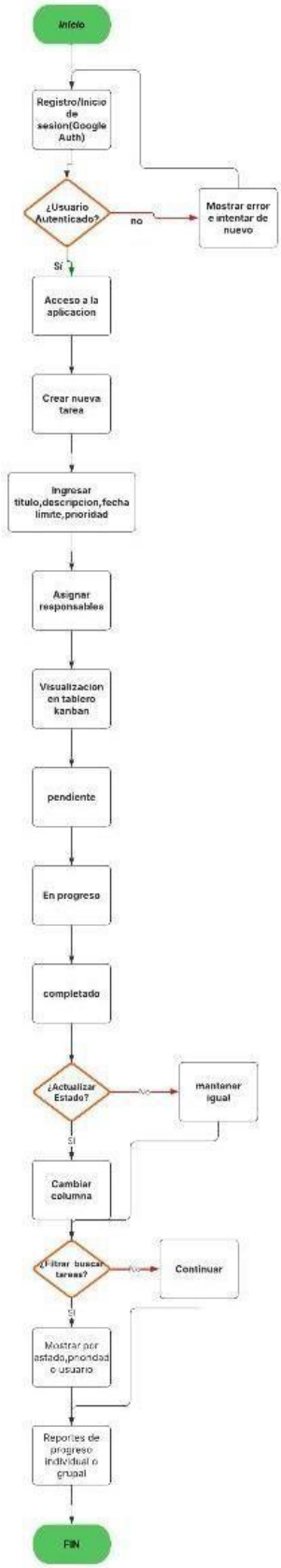
En conjunto, estas herramientas ofrecen una cobertura integral de calidad y seguridad, desde la codificación hasta el despliegue. Además, la mayoría cuenta con versiones gratuitas u open-source, lo que facilita su adopción en proyectos académicos y empresariales sin representar un alto costo.

12. Diagrama de flujo de solución selecciona.

Link diagrama de flujo (https://lucid.app/lucidchart/cd6052ec-e03e-4620-ad47-951309399339/edit?viewport_loc=277%2C443%2C1218%2C506%2C0_0&invitat

ionId=inv_1987536e-fb51-4d4a-9965-267ab221240d)

Diagrama de flujo



Historias de Usuario en el tablero de la metodología Ágil seleccionada. Link (https://trello.com/invite/b/68d59cb7461e85b65a97f558/ATTI1329032e17cca957fadf00003137851317D2A48E/proyecto-de-software)

Autenticación y Acceso

1. Como usuario, quiero iniciar sesión en el sistema usando mi cuenta personal de Google para acceder de forma rápida y segura sin crear una nueva contraseña.
2. Como usuario, quiero cerrar sesión desde el sistema para proteger mi información cuando ya no necesite usar la aplicación.
3. Como administrador, quiero asegurar que solo los usuarios autenticados con Google puedan acceder al sistema para evitar accesos no autorizados.
4. Como administrador, quiero definir roles (administrador, miembro de equipo, observador) tras la autenticación para controlar qué acciones puede realizar cada usuario.
5. Como usuario, quiero que mi sesión permanezca activa mientras trabajo para no tener que iniciar sesión constantemente.
6. Como administrador, quiero poder revocar el acceso de un usuario autenticado para mantener la seguridad en caso de que alguien ya no deba usar la aplicación.

Gestión de tareas

7. Como usuario, quiero **crear** tareas con título, descripción, fecha límite y prioridad para organizar mis actividades de forma clara.
8. Como miembro de un equipo, quiero **asignar** tareas a usuarios específicos para distribuir responsabilidades y evitar confusión.

9. Como usuario, quiero **cambiar el estado** de mis tareas (**pendiente, en progreso, completada**) para reflejar el avance de mi trabajo.
10. Como usuario, quiero visualizar mis tareas en un tablero Kanban para tener una vista general de mi progreso.
11. Como miembro de un equipo, quiero ver el progreso grupal para evaluar el avance del proyecto en conjunto.
12. Como usuario, quiero **filtrar y buscar tareas** por estado, prioridad o usuario para encontrar fácilmente la información que necesito.
13. Como usuario, quiero **comentar dentro de las tareas** para mejorar la comunicación y resolver dudas sin depender de otras herramientas.
14. Como usuario, quiero **adjuntar documentos** a las tareas para centralizar la información necesaria para cumplirlas.
15. Como usuario, quiero **consultar el historial** de cambios de una tarea para saber quién hizo modificaciones y en qué momento.

13. Definición de RQF-RQNF manejar un formato para la documentación.

Requisitos Funcionales. (RF)

ID	Requerimiento	Descripción
RF01	**Análisis de calidad del código**	El sistema deberá permitir realizar análisis de calidad del código fuente para detectar errores y malas prácticas.
RF02	**Detección de vulnerabilidades de seguridad**	El sistema deberá identificar posibles vulnerabilidades de seguridad en la aplicación.
RF03	**Ejecución de pruebas automáticas**	El sistema deberá ejecutar pruebas

		automáticas que validen el correcto funcionamiento de las principales funciones del software.
RF04	**Generación de reportes básicos**	El sistema deberá generar reportes básicos con los resultados de los análisis y pruebas realizados.
RF05	Impedir paso a producción con fallas críticas	El sistema deberá impedir el paso a producción de versiones que presenten fallas críticas.

Requerimientos No funcionales. (RNF)

ID	Requerimiento	Descripción
RNF01	**Facilidad de uso**	El sistema deberá ser fácil de utilizar por el equipo de desarrollo, con una interfaz clara y comprensible.
RNF02	**Integración con repositorios de control de versiones**	El sistema deberá integrarse de manera sencilla con repositorios

		de control de versiones como GitHub o GitLab.
RNF03	**Disponibilidad continua**	El sistema deberá estar disponible de forma continua durante el ciclo de desarrollo del proyecto.
RNF04	**Claridad en los reportes**	El sistema deberá presentar reportes con información clara y comprensible para los programadores.
RNF05	**Escalabilidad**	El sistema deberá ser escalable, de modo que pueda adaptarse a futuros proyectos sin grandes modificaciones.