**CORPORACIÓN UNIVERSITARIA MINUTO DE DIOS – UNIMINUTO**

**Pedro David Fernández – ID 694841**

**Yeison David Negrete –ID 688627**

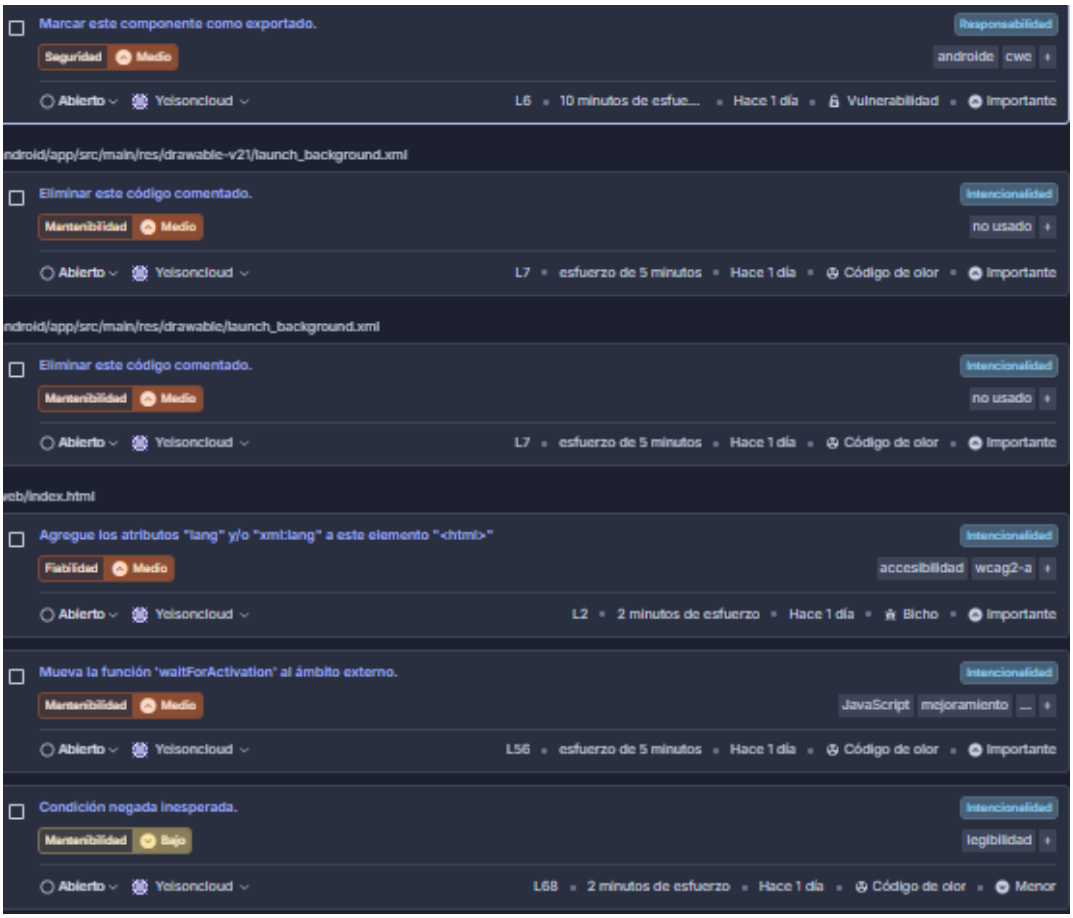
**Desarrollo de software seguro**

**Edwin Alberto Ramos Villamil**

**Bogotá 1/02/2026**

Imagen 1


Imagen 2



**1.** Análisis del Caso

- **Entorno:** Multiplataforma utiliza el SDK de Flutter. Se emplea un flujo de trabajo que integra herramientas de análisis estático de código (SAST) (2026 Black Duck Software, Inc. Todos los derechos reservados, s.f.) para la detección temprana de vulnerabilidades.
- **Producto:** Aplicación móvil y web orientada a la eficiencia operativa, que maneja configuraciones nativas de Android y estructuras HTML5.
- **Contexto:** El proyecto se encuentra en una fase de aseguramiento de calidad donde se busca mitigar la "deuda técnica" (ibm, s.f.) y fortalecer la postura de seguridad antes del despliegue final.

**2.** Clasificación Detallada de Amenazas

- **Amenazas de Seguridad (CWE-276):** Según (MITRE, 2026) El componente en AndroidManifest.xml sin el atributo android:exported definido explícitamente representa un riesgo de exposición de componentes internos a aplicaciones maliciosas.
- **Amenazas de Mantenibilidad:** El código comentado y funciones mal encapsuladas aumentan la complejidad cognitiva, dificultando futuras actualizaciones y facilitando la introducción de errores.
- **Amenazas de Fiabilidad y Accesibilidad (WCAG 2.1):** La omisión del atributo lang en el archivo index.html afecta la interpretación de la página por tecnologías de asistencia y motores de búsqueda. (W3C, s.f.)

**3.** Justificación de la Metodología de Desarrollo Seguro

- **Justificación:** Al integrar el análisis estático desde la codificación como se observa en el informe que nos da (sonarQube), se reduce el costo de mitigación. La seguridad deja de ser una fase final para convertirse en un proceso continuo que garantiza la integridad de los datos desde el primer "commit".

**4.** Plan de Integración desde los Requerimientos

- **Requerimientos No Funcionales:** Establecer que todo componente de Android debe seguir el principio de menor privilegio.

- **Criterios de Aceptación:** Definir que el código no debe ser aprobado para despliegue si presenta olores de código code smells (Skobeleva, 2026) de criticidad media o superior.
- **Automatización:** Configurar pipelines de CI/CD que ejecuten automáticamente las reglas de análisis estático vistas en la imagen.

**5.** Mapa de Buenas Prácticas por Fase

- **Diseño:** Definición de arquitectura de componentes cerrados (evitar exported="true" innecesarios).
- **Desarrollo:** Aplicación de principios sólidos y limpieza de código (eliminar comentarios y refactorizar lógica negada).
- **Pruebas:** Ejecución de escaneos de vulnerabilidades y pruebas de accesibilidad web.
- **Despliegue:** Verificación de metadatos y encabezados de seguridad en el entorno web.

**6.** Matriz Metodológica

Tabla 1

| **Riesgo Detectado** | **Fase Afectada** | **Solución Técnica** | **Impacto** |
|---|---|---|---|
| Exposición de componentes | Configuración | Definir android:exported="false" | Seguridad Alta |
| Deuda técnica (Código muerto) | Construcción | Eliminación de bloques comentados | Mantenibilidad |
| Inaccesibilidad Web | Interfaz (UI) | Inserción de atributo lang="es" | Cumplimiento Legal |
| Lógica compleja (Negación) | Codificación | Refactorización a lógica positiva | Legibilidad |

# Referencias

*2026 Black Duck Software, Inc. Todos los derechos reservados*. (s.f.).

ibm. (s.f.). Obtenido de IBM: https://www.ibm.com/es-es/think/topics/technical-debt

MITRE. (2026). *MITRE*. Obtenido de https://cwe.mitre.org/data/definitions/732.html

Skobeleva, O. (2026). *Refactoring guru*. Obtenido de https://refactoring.guru/es/refactoring/smells

W3C. (s.f.). *W3.ORG*. Obtenido de https://www.w3.org/TR/WCAG21/#informative-references