Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Software

Desarrollo de Software Seguro

Laboratorio 1 - Tercer Parcial

Análisis de código estático con SonarQube para la identificación y corrección de vulnerabilidades de seguridad en aplicación web full-stack.

Docente: Geovanny Cudco
Estudiantes: [Nombre 1, Nombre 2, Nombre 3]
Grupo: [Número de Grupo]

Fecha: 8 de enero de 2026

Duración	2 horas
Integrantes por equipo	3 estudiantes
Puntaje máximo	20 puntos
Fecha de entrega	[08-01-2026]

Índice

1. Tema **1**

2. Objetivo **1**

3. Contexto **1**

4. Descripción de la actividad **1**
4.1. Configuración inicial (15 minutos) 1
4.2. Primer análisis (30 minutos) 1
4.3. Configuración de Quality Gates (20 minutos) 2
4.4. Corrección de vulnerabilidades (40 minutos) 2
4.5. Verificación final (15 minutos) 2

5. Entregables **2**

6. Criterios de evaluación **3**
6.1. Notas de Evaluación: 4

7. Anexos **4**
7.1. sonar-project.properties 4
7.2. Ejemplo de reglas de QUALITY GATES 5
7.3. Ejemplo de vulnerabilidades 5
7.4. Recursos adicionales 5

1. Tema

Análisis de código estático con SonarQube para identificación y corrección de vulnerabilidades de seguridad en aplicación web full-stack (Node.js/Express + ExtJS + PostgreSQL).

2. Objetivo

Identificar, analizar y corregir vulnerabilidades de seguridad y problemas de calidad de código en una aplicación full-stack mediante el uso de SonarQube, aplicando principios SOLID y patrones de diseño.

3. Contexto

Se ha desarrollado una aplicación de gestión de parqueaderos con arquitectura cliente-servidor. Sin embargo, el código presenta múltiples vulnerabilidades de seguridad y anti-patrones de desarrollo. Su equipo debe:

1. Analizar el código con SonarQube
2. Configurar quality gates personalizadas
3. Identificar vulnerabilidades críticas
4. Corregir al menos 5 vulnerabilidades aplicando buenas prácticas

4. Descripción de la actividad

4.1. Configuración inicial (15 minutos)

1. Clonar el repositorio con la aplicación vulnerable - `https://github.com/agcudco/taller-sonar-qube.git`
2. Verificar que SonarQube y SonarScanner estén instalados
3. Crear archivo `sonar-project.properties` con la configuración del análisis
4. Configurar SonarScanner para analizar todo el proyecto

4.2. Primer análisis (30 minutos)

1. Ejecutar análisis estático de código completo
2. Identificar al menos 10 vulnerabilidades (5 backend, 5 frontend)
3. Clasificar vulnerabilidades por tipo y severidad
4. Documentar hallazgos en informe inicial

4.3. Configuración de Quality Gates (20 minutos)

1. Definir reglas de quality gates enfocadas en:
   - Seguridad del código
   - Calidad del diseño
   - Mantenibilidad
2. Configurar estas reglas en SonarQube
3. Ejecutar análisis con quality gates activadas

4.4. Corrección de vulnerabilidades (40 minutos)

1. Corregir al menos **5 vulnerabilidades críticas**, aplicando:
   - **Principios SOLID**: SRP, OCP, LSP, ISP, DIP
   - **Patrones de diseño**: Factory, Singleton, Repository, MVC
   - **Buenas prácticas**: validación de entrada, manejo seguro de errores, etc.

4.5. Verificación final (15 minutos)

1. Ejecutar análisis final con SonarQube
2. Verificar que las vulnerabilidades corregidas ya no aparezcan
3. Confirmar cumplimiento de quality gates
4. Generar reporte final

5. Entregables

1. Archivo `sonar-project.properties` configurado
2. Lista de vulnerabilidades identificadas (antes/después)
3. Capturas de pantalla de dashboards de SonarQube
4. Código fuente modificado con las correcciones
5. Informe final (seguir formato de informes en LaTeX) que incluya:
   - Vulnerabilidades identificadas
   - Quality gates definidas
   - Explicación de correcciones aplicadas
   - Principios SOLID y patrones utilizados

6. Criterios de evaluación

CRITERIO DE EVALUACIÓN	EXCELENTI (5 puntos)	BUENO (4 puntos)	REGULAR (3 puntos)	DEFICIENT (2 puntos)	INSUFICIENTE (1 punto)
1. ANÁLISIS INICIAL CON SONARQUBE (5 puntos)	Identifica y documenta >= 10 vulnerabilidade correctamente clasificadas por tipo, severidad y ubicación (frontend/back	Identifica 8-9 vulnerabilidade con clasificación mayoritariame correcta.	Identifica 6-7 vulnerabilidade con algunas clasificaciones incorrectas.	Identifica 4-5 vulnerabilidade con clasificación deficiente.	Identifica <4 vulnerabilidades o documentación muy pobre.
2. CONFIGURACIÓN DE QUALITY GATES (5 puntos)	Define >=5 reglas relevantes, bien enfocadas en seguridad y calidad, y las configura correctamente en SonarQube.	Define 3 reglas relevantes y las configura correctamente.	Define 2 reglas, algunas no totalmente relevantes o con errores en configuración.	Define 1 regla o tiene errores significativos en configuración.	No define reglas relevantes o no configura los Quality Gates.
3. CORRECCIÓN DE VULNERABILIDAI (8 puntos)	Corrige >=5 vulnerabilidade aplicando principios SOLID y patrones de diseño correctamente. Todas pasan análisis posterior.	Corrige 4 vulnerabilidade aplicando buenas prácticas. La mayoría pasa análisis posterior.	Corrige 3 vulnerabilidade con soluciones parciales. Algunas persisten en análisis posterior.	Corrige 1-2 vulnerabilidade con soluciones básicas, sin aplicar buenas prácticas.	No corrige vulnerabilidades o las correcciones no son efectivas.
4. CALIDAD DEL CÓDIGO Y DOCUMENTACIÓN (2 puntos)	Código limpio, bien estructurado, documentado y con mejoras arquitectónicas evidentes. Reporte completo.	Código funcional pero con documentación limitada y pocas mejoras en estructura.	Código funcional pero sin mejoras significativas. Documentación básica.	Código con cambios mínimos. Documentación incompleta.	Código no funcional o sin cambios. Sin documentación.

DESGLOSE DE PUNTUACIÓN POR CRITERIO	PUNTOS
Análisis Inicial con SonarQube	5 puntos
Configuración de Quality Gates	5 puntos
Corrección de Vulnerabilidades	8 puntos
Calidad del Código y Documentación	2 puntos
TOTAL	**20 puntos**

6.1. Notas de Evaluación:

- Se considerará **vulnerabilidad corregida** solo si desaparece del reporte final de SonarQube.
- La aplicación de **principios SOLID** debe ser demostrable en el código (ej: clases con única responsabilidad, interfaces definidas).
- Los **patrones de diseño** deben estar correctamente implementados (ej: Repository para acceso a datos, Factory para creación de objetos).
- El **reporte final** debe incluir capturas de SonarQube que muestren la evolución del análisis.
- La **colaboración en equipo** será evaluada indirectamente a través de la completitud del trabajo.

7. Anexos

7.1. sonar-project.properties

```
# SonarQube Configuration for Parking App
sonar.projectKey=parking-app-vulnerable
sonar.projectName=Vulnerable Parking App
sonar.projectVersion=1.0

# Source code directories
sonar.sources=backend,frontend

# Language
sonar.language=js

# Exclusions
sonar.exclusions=**/node_modules/**,**/*.min.js,**/lib/**

# Test directories
sonar.tests=backend/test
sonar.test.inclusions=**/*.test.js,**/*.spec.js
```

```
# Coverage
sonar.javascript.lcov.reportPaths=coverage/lcov.info

# Analysis settings
sonar.sourceEncoding=UTF-8
sonar.host.url=http://localhost:9000
sonar.login=admin
sonar.password=admin

# JavaScript specific
sonar.javascript.file.suffixes=.js

# Quality Gate
sonar.qualitygate.wait=true
```

7.2. Ejemplo de reglas de QUALITY GATES

Regla 1: **Seguridad Crítica**: "No deben existir vulnerabilidades críticas relacionadas con inyección SQL, XSS o manejo inseguro de autenticación"

Regla 2: **Calidad de Código**: "La deuda técnica no debe superar los 30 minutos por cada 100 líneas de código"

Regla 3: **Mantenibilidad**: .El código duplicado no debe exceder el 5 % del total del código fuente"

Regla 4: **Confiabilidad**: "La cobertura de pruebas debe ser al menos del 70 % para componentes críticos"

Regla 5: **Buenas Prácticas**: "No deben existir anti-patrones como código comentado, funciones demasiado largas (¿50 líneas) o complejidad ciclomática alta (¿10)"

7.3. Ejemplo de vulnerabilidades

Backend	**Frontend**
Inyección SQL en consultas nativas	URLs hardcodeadas en código
CORS mal configurado (permite *)	XSS potencial en renderización
Exposición de stack traces	Manejo inseguro de datos sensibles
Falta de límites en body-parser	Falta de validación de entrada
Credenciales en código plano	Código ofuscado/anti-patrones

7.4. Recursos adicionales

- Documentación oficial de SonarQube: `https://docs.sonarqube.org/latest/`
- Guía de instalación de SonarScanner: `https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/`

- Repositorio de la aplicación vulnerable: `https://github.com/agcudco/taller-sonar-qube`
- Tutorial de principios SOLID: `https://refactoring.guru/es/design-patterns/solid-principles`
- Guía de patrones de diseño: `https://refactoring.guru/es/design-patterns`