



Punto 7 Clean Code

Ingeniería de software I

Kevin David Rodriguez Riveros

Frank Sebastian Pardo Amaya

Jorge Andrés Torres Leal

Departamento de Ingeniería de Sistemas e Industrial

Universidad Nacional de Colombia - Sede Bogotá

Facultad de Ingeniería

9 de Febrero de 2025

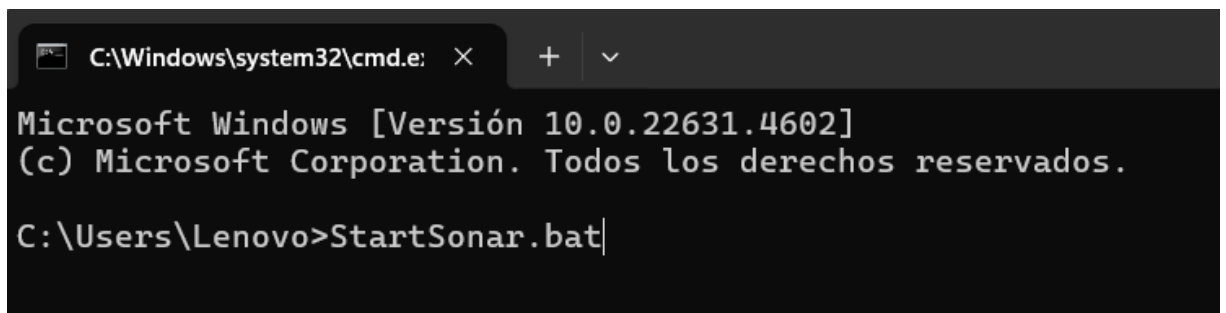
Clean Code

En esta sección se documenta el proceso de análisis de código limpio para el proyecto StockEase. Se utilizó SonarQube como herramienta principal de análisis estático, lo que permitió detectar errores, vulnerabilidades y mejorar la mantenibilidad del software.

Herramientas utilizadas: Para el análisis de calidad del código de StockEase, utilizamos **SonarQube** y **Checkstyle**: Se eligió SonarQube por su capacidad de integrarse con Maven y su funcionalidad de detección de código duplicado, vulnerabilidades de seguridad y el análisis con problemas de mantenibilidad. Y además se configuró Checkstyle en NetBeans para verificar si el código sigue las convenciones de Java.

Instalación de SonarQube

1. Se descargó **SonarQube Community Edition** desde SonarQube Downloads.
2. Se extrajo y se ejecutó el servidor (en windows) con **StartSonar.bat**



```
C:\Windows\system32\cmd.e: X + v
Microsoft Windows [Versión 10.0.22631.4602]
(c) Microsoft Corporation. Todos los derechos reservados.
C:\Users\Lenovo>StartSonar.bat|
```

3. Se accedió a la interfaz en **http://localhost:9000**.

Instalación de SonarScanner

1. Se descargó SonarScanner.
2. Se configuró la variable de entorno SONAR_SCANNER_HOME.
3. Se agregó al **PATH** para su ejecución global.

Configuración en Maven

1. Se agregó el siguiente plugin en **pom.xml**:

```
<properties>
  <sonar.host.url>http://localhost:9000</sonar.host.url>
</properties>

<build>
  <plugins>
    <plugin>
      <groupId>org.sonarsource.scanner.maven</groupId>
      <artifactId>sonar-maven-plugin</artifactId>
      <version>3.9.1.2184</version>
    </plugin>
  </plugins>
</build>
```

2. Se ejecutó el análisis con el comando:

mvn clean verify sonar:sonar

3. Se revisaron los resultados en **http://localhost:9000**.

Configuración de Checkstyle en NetBeans

1. Se instaló el plugin de Checkstyle en NetBeans desde la sección de complementos.
2. Se configuró para que siga las reglas estándar de Google Java Style Guide.
3. Se ejecutó el análisis y se revisaron las advertencias generadas sobre el código fuente.

Resultados de los Análisis

- Métodos largos y con múltiples responsabilidades puede hacer que algunas funciones como registrarProducto() en ProductoController, manejen demasiada lógica por lo que pueden ser divididas en métodos más pequeños.
- Se identificaron fragmentos repetidos en diferentes clases, pero se pueden refactorizar y crear métodos reutilizables.
- Uso inconsistente de nombres de variables, porque algunas variables no siguen un estándar claro, lo que afecta la legibilidad para otros desarrollos o actualizaciones.
- Falta de documentación en algunas clases, hay métodos sin comentarios explicativos.
- Posibles vulnerabilidades, ya que las validaciones en la entrada de datos son insuficientes, pero pueden ser más complejas.

Evaluación de los integrantes

Cada integrante realizó una breve evaluación sobre la calidad del código:

- **Kevin Rodríguez:** El código en general es fácil de seguir pero algunos métodos en ProductoController contienen demasiada lógica en un solo lugar, lo que hace que sean difíciles de modificar sin afectar otras partes del sistema, desde mi punto de vista es mejor dividir estos métodos en funciones más pequeñas y seguir el principio

de responsabilidad única.

- **Frank Pardo:** La estructura del código facilita la comprensión sin embargo, observo que algunas variables no tienen nombres suficientemente descriptivos, lo que podría dificultar la lectura para otros desarrolladores ajenos a nuestro equipo... Y también la falta de documentación en ciertos métodos hace que sea necesario deducir su propósito.
- **Jorge Torres:** El código es comprensible en su mayoría pero hay fragmentos repetidos en distintas clases, lo que nos afecta la mantenibilidad del sistema. Podríamos aplicar el principio DRY (Don't Repeat Yourself) ayudaría a reducir la duplicidad y mejorar la reutilización de código

Conclusión

Luego de verificar nuestras respuestas, decimos que el análisis con las herramientas de SonarQube y Checkstyle nos permitió identificar ciertos problemas en la estructura y calidad del código del proyecto StockEase. Si bien el código es funcional y el software funciona con lo que buscamos por ahora, sí es cierto que hay áreas de mejora en cosas como la legibilidad, modularidad y reutilización del código. Como próximos pasos trabajaremos en la refactorización de los métodos que identificamos largos, el uso consistente de nombres de variables y la reducción de código duplicado para mejorar la mantenibilidad del sistema y quizá optimizarlo.