

**SWT301\_LAB2\_ REPORT**

**Static analysis Report - PVS tool**

**GROUP 2**

**Đào Quang Nguyên – HE176610**

– Hanoi,January 2024 –

# I. Project Report

1. Introduction

1.1 Introduction your Project

In this project, we're learning about the SonarLint. Our goal is to use this tool to find mistakes in a SWP391 Project and project that I consulted online . By doing this, we aim to improve our skills in spotting and fixing errors in software development.

1.2 Introduction to the Testing Tool:

Tool name: SonarLint

Description: SonarLint is a local, on-the-fly static code analysis tool integrated seamlessly with IntelliJ IDEA. It focuses on enhancing code quality by identifying and addressing code smells, bugs, and security vulnerabilities during the development process. SonarLint provides immediate feedback and suggestions within the IDE, allowing developers to proactively improve code quality.

Purpose of the Testing Tool: The primary purpose of the SonarLint is to automate and streamline the testing process within the IDEA IntelliJ environment. It aims to reduce manual effort, increase efficiency, and improve the reliability of the testing process.

Testing objectives:

Code Quality Assurance:

Identify and flag code quality issues such as code smells, potential bugs, and security vulnerabilities.

Enforce coding standards and best practices to maintain a consistent and high-quality codebase.

Early Bug Detection:

Detect and report potential bugs and errors in the code before they manifest as runtime issues, helping developers catch and fix issues early in the development lifecycle.

Security Vulnerability Detection:

Identify security vulnerabilities in the codebase, such as potential injection points, insecure configurations, or other security-related issues.

Code Maintainability:

Assess and provide insights into the maintainability of the code by identifying complex code structures, duplications, and other factors that may hinder future maintenance.

2. Test Strategy:

Test approach: test strategy involves the use of static testing techniques. The SonarLint analyzes the source code of my SWP project to detect potential errors, helping us identify and rectify issues at an early stage of development.

Test environment setup: The test environment setup for the SonarLint requires specific hardware and software configurations. The hardware requirements include a system with sufficient processing power and memory to handle the testing process. The software requirements include the IDEA IntelliJ environment and the SonarLint plugin. Additionally, we need to prepare test data that closely mimics real-world data to ensure the validity of our tests. Any dependencies, such as databases or other services, need to be set up and configured correctly in the test environment. This setup helps us create a controlled environment where we can accurately assess the functionality and performance of the software.

**NOTE (Error level):** SonarLint, a static code analysis tool, classifies warnings based on the certainty of a bug being found. Specifically, PVS-Studio uses three levels:

**High (Red):** The most suspicious places, where the highest probability of errors exists.

**Medium(Yellow):** Warnings at this level are also very important and should be reviewed, but they are not as certain as level 1.

**Low (Blues**: Places where inaccuracies, most likely insignificant, are present in the code.

**Error founded**

1.

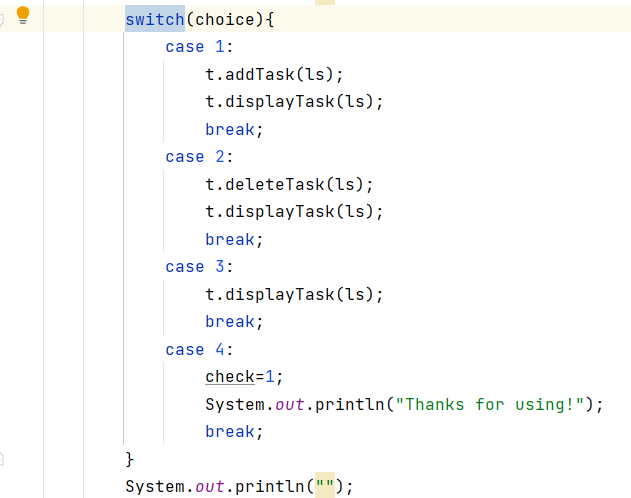
- Type of bug: Java:S131

-Level: High

- Bug image:



-Bug image before fixed:



- Bug image after fixed:



2.

- Type of bug: Java:S1659

- Level:Low

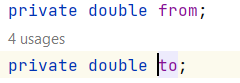
- Bug image:



-Bug image before fixed:



- Bug image after fixed:



3.

- Type of bug: Java:S1659

- Level:Low

- Bug image:



-Bug image before fixed:



- Bug image after fixed:



4.

- Type of bug: Java:S112

- Level:Medium

- Bug image:



-Bug image before fixed:



- Bug image after fixed:



5.

- Type of bug: Java:S1488

- Level:Low

- Bug image:



-Bug image before fixed:



- Bug image after fixed:



6.

- Type of bug: Java:S1319

- Level:Low

- Bug image:



-Bug image before fixed:



- Bug image after fixed:



7.

-Type of bug : Java:1659

-Level:Low

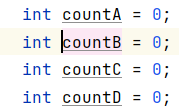
-Bug image:



-Bug image before fixed:



- Bug image after fixed:



8.

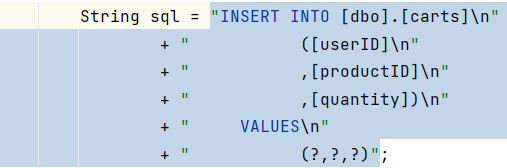
-Type of bug: Java:S6126

-Level:Medium

-Bug image:



-Bug image before fixed:



- Bug image after fixed:

String sql = new StringBuilder().append("INSERT INTO [dbo].[carts]\n").append(" ([userID]\n").append(" ,[productID]\n").append(" ,[quantity])\n").append(" VALUES\n").append(" (?,?,?)").toString();

9.

-Type of bug:Java:S1854

-Level: Medium

-Bug image:



-Bug image before fixed:



- Bug image after fixed:



10.

-Type of bug:Java:S1186

-Level: High

-Bug image:



-Bug image before fixed:

protected void processGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
   
}

- Bug image after fixed:

protected void processGet() throws ServletException, IOException {  
 processGet(null, null);  
}

11.

-Type of bug:Java:S2293

-Level: Low

-Bug image:



-Bug image before fixed:

Vector<Cart> vector = new Vector<Cart>();

- Bug image after fixed:

Vector<Cart> vector = new Vector<>();

12.

-Type of bug: Java:S1117

-Level: Medium

-Bug image:



-Bug image before fixed:

ResultSet rs = null;

- Bug image after fixed:



13.

-Type of bug: Java:S1481

-Level: Low

-Bug image:



-Bug image before fixed:

User user = (User) request.getSession().getAttribute("user");

- Bug image after fixed:

User user;  
user = (User) request.getSession().getAttribute("user");

14.

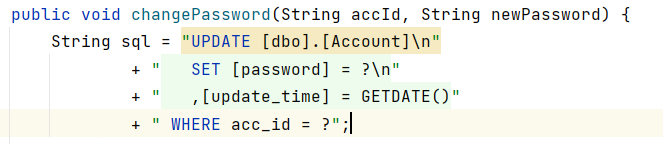
-Type of bug: Java:S1192

-Level: High

-Bug image:



-Bug image before fixed:



- Bug image after fixed:

String sql = new StringBuilder().append("UPDATE [dbo].[Account]\n").append(" SET [password] = ?\n").append(" ,[update\_time] = GETDATE() WHERE acc\_id = ?").toString();

15.

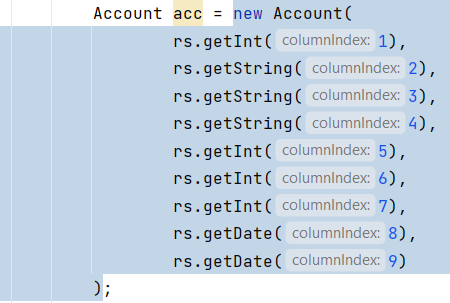
-Type of bug: Java:S1488

-Level:Low

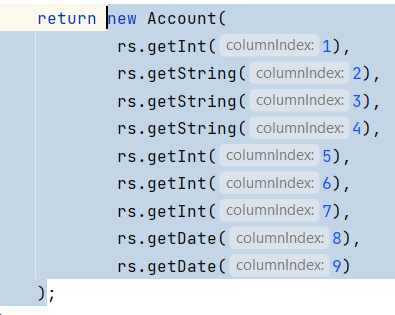
-Bug image:



--Bug image before fixed:



- Bug image after fixed:



16.

-Type of bug: Java:S1854

-Level:Medium

-Bug image:



--Bug image before fixed:

List<Product> listProduct = prd.getAllProduct();

- Bug image after fixed:

List<Product> listProduct;  
listProduct = prd.getAllProduct();

17.

-Type of bug: Java:S1149

-Level: Medium

-Bug image:



--Bug image before fixed:

Vector<Product> listProduct = daoProduct.pagingProduct(index, productPerPage);

- Bug image after fixed:

Vector<Product> listProduct;  
 listProduct = daoProduct.pagingProduct(index, productPerPage);

18.

-Type of bug: Java:S1659

-Level: Low

-Bug image:



--Bug image before fixed:

int id, num = 0;

- Bug image after fixed:

int id;  
 int num = 0;

19.

-Type of bug: Java:S1854

-Level:Medium

-Bug image:



--Bug image before fixed:

endPage++;

- Bug image after fixed:

endPage = endPage + 1;

19.

-Type of bug: Java:S1643

-Level:Low

-Bug image:



--Bug image before fixed:

sql += categoryIDs[i] + ", ";

- Bug image after fixed:

sql = sql + (categoryIDs[i] + ", ");

19.

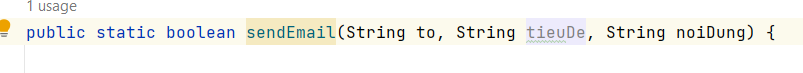
-Type of bug: Java:S1172

-Level:Medium

-Bug image:



--Bug image before fixed:



- Bug image after fixed:



20.

-Type of bug: Java:S1128

-Level:Low

-Bug image:



--Bug image before fixed:



- Bug image after fixed:

