UNIVERSITÉ ABDELHAMIDHRI CONSTANTINE 2
FACULTÉ DES NOUVELLES TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION NTIC
LE DÉPARTEMENT TECHNOLOGIES DES LOGICIELS ET DES SYSTÈMES D'INFORMATION TLSI

TP MEL
rapport 2

SPECIALTY: SOFTWARE ENGINEERING

Maintenance of DAC Project (overView)

Supervised by:
- Dr. Manel DJENOUHAT

Presented by:
- Rouissa Rabah G2
- Belmokhi Dibadj G2
- Brahmia Mohamed Haythem Abderrahmen G2

DATE: 17/03/2025

Contents

List of Tables

List of Figures

1 Introduction

1.1 Project Context

SORTVIEW Desktop Application project focuses on creating a high-performance, user-friendly tool that enables users to classify multiple images concurrently. The application will allow users to load and display images, initiate classification tasks. Leveraging multi-threading, it will support simultaneous processing to optimize performance and efficiency. Robust error-handling mechanisms will ensure the application functions smoothly. The project emphasizes simplicity, clarity, and usability in its interface design, ensuring a seamless user experience.

1.2 Maintenance Objectives

The primary objectives for maintaining the SORTVIEW Desktop Application are:

1.2.1 Code Understandability

Ensure that the code is easy to understand for future developers. This will be achieved by fixing variable names, using clear and consistent naming conventions, and removing any ambiguities that may lead to misunderstandings.

1.2.2 Eliminating Code Duplication

Address any instances of duplicated code to reduce redundancy and make the code more maintainable. This will ensure that changes are easier to implement and less error-prone.

1.2.3 Optimizing Code

Remove unnecessary or overly complicated lines of code to improve efficiency. Refactor complex logic into simpler, more modular components to improve readability and maintainability.

1.3 Tools and Methodology

For efficient maintenance, the following tools and methodologies will be used:

1.3.1 SonarCloud

The primary focus will be on using SonarCloud for continuous static code analysis to identify and fix code smells. This will help in eliminating unnecessary complexities, improving code readability, and ensuring that the code remains easy to maintain for future developers.

1.3.2 GitHub

GitHub will be used for version control, enabling smooth collaboration, tracking changes, and managing code history. It ensures that all modifications are well-documented and easy to access by future developers.

By prioritizing the resolution of code smells and minimizing unnecessary complexity, the SORTVIEW Desktop Application will remain clean, understandable, and easier for future developers to maintain while preserving its core functionality.

2 Analysis of Current State

2.1 SonarQube Metrics Overview

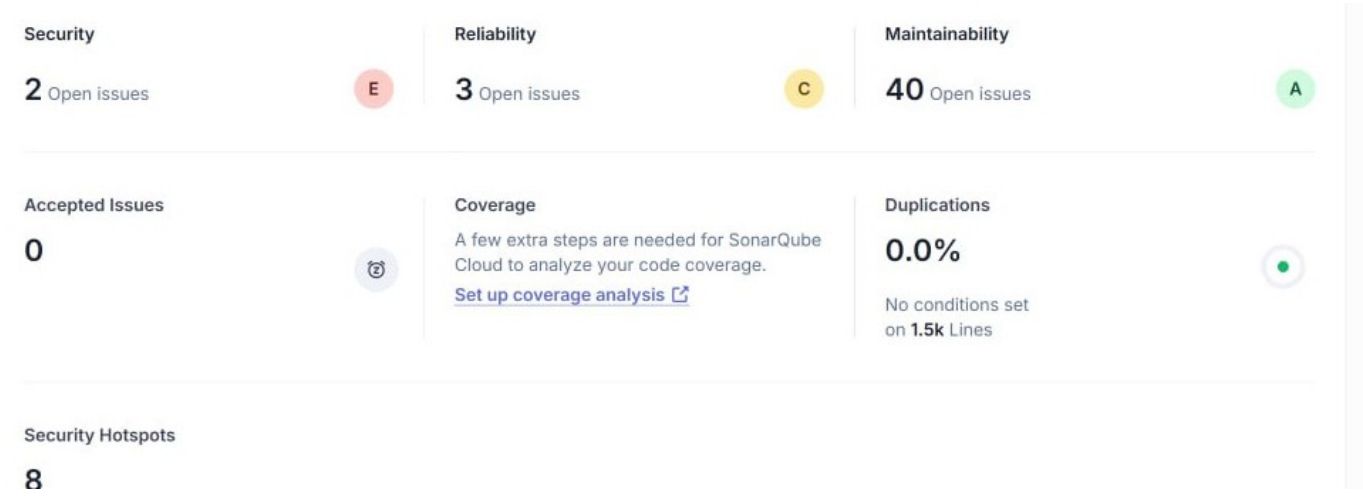

Figure 1: SonarQube Metrics Overviewe

3 Identified Issues

3.1 Security Issues

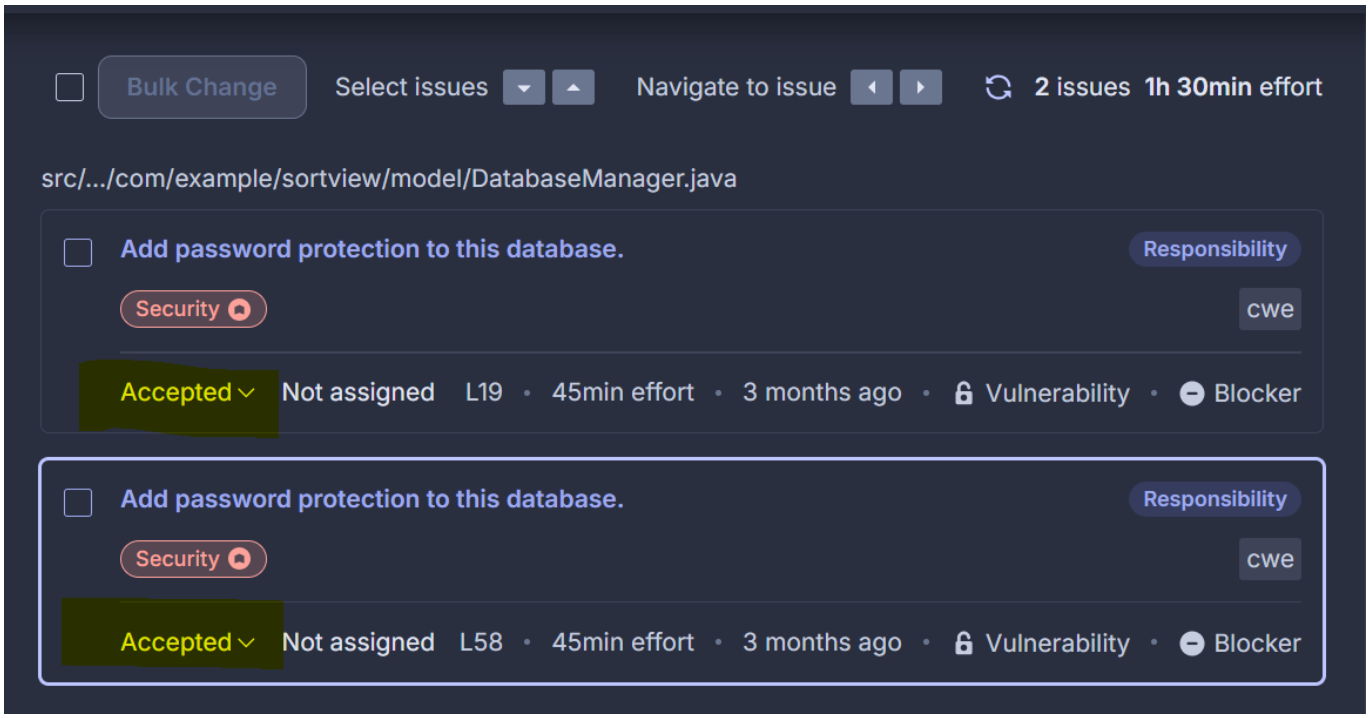

Figure 2: Security Issues

3.2 Reliability Issues

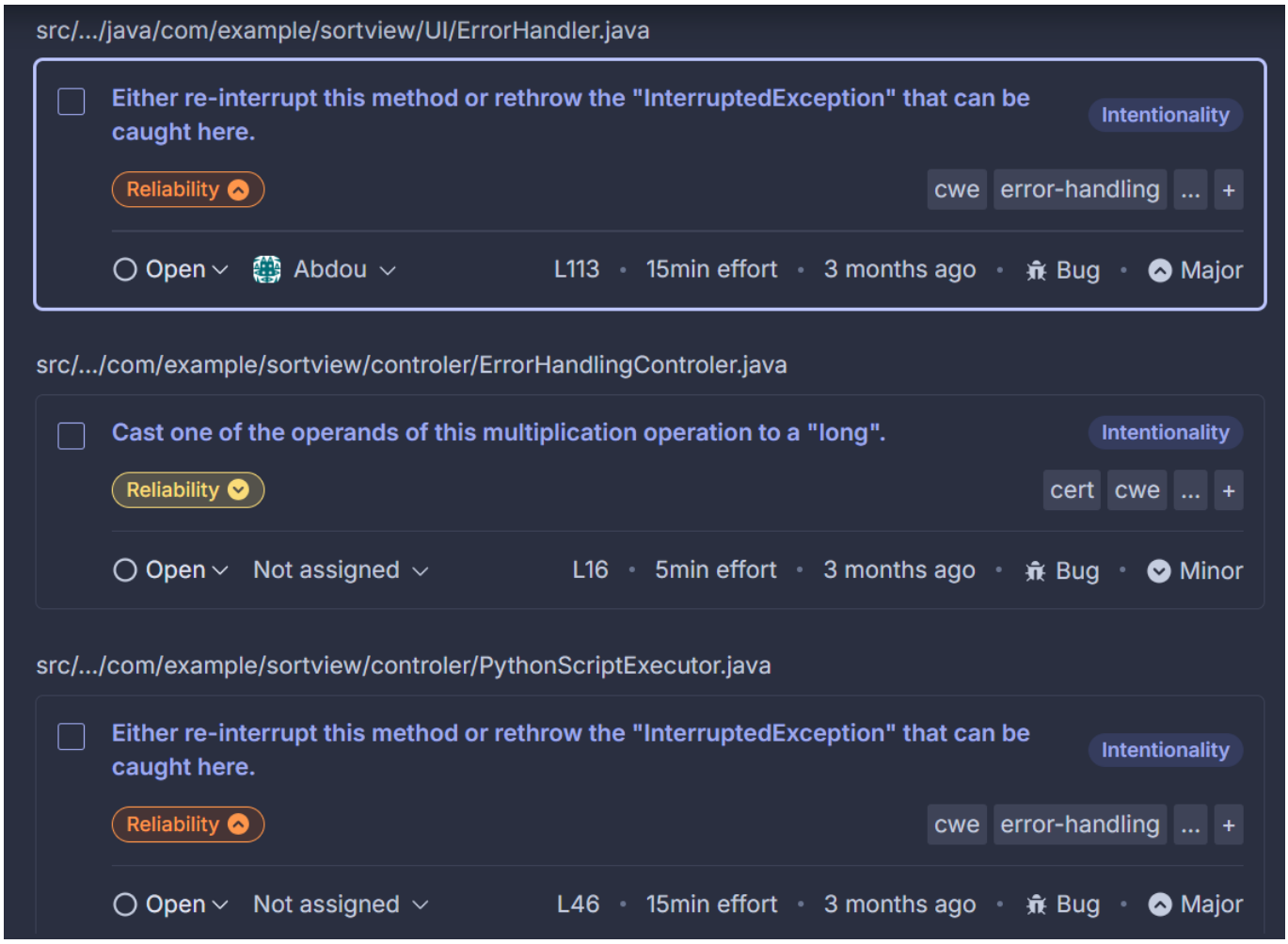

Figure 3: Reliability Issues Overview

Figure 4: Reliability Issues - First Issue

```
public boolean askForCon() {
    Platform.runLater(() -> start(new Stage()));

    try {
        latch.await();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    return this.canContinue;
}
```

```
108+ public boolean askForCon() throws InterruptedException {
109      Platform.runLater(() -> start(new Stage()));
110      latch.await();
111      return this.canContinue;
112  }
```

Figure 5: Reliability Issues - First Issue Solution

```
for (int i = 0; i < files.size(); i++) {
    long MAX_FILE_SIZE = 5 * 1024 * 1024; // 5 MB
```

Cast one of the operands of this multiplication operation to a "long".

```
    if (files.get(i).length() <= MAX_FILE_SIZE) {
```

Figure 6: Reliability Issues - Second Issue

```
for (int i = 0; i < files.size(); i++) {
    long MAX_FILE_SIZE = 5 * 1024 * 1024; // 5 MB
    if (files.get(i).length() <= MAX_FILE_SIZE) {
```

```
15      for (int i = 0; i < files.size(); i++) {
16+         long MAX_FILE_SIZE = 5L * 1024 * 1024; // 5 MB
17          if (files.get(i).length() <= MAX_FILE_SIZE) {
```

Figure 7: Reliability Issues - Second Issue Solution

```
        return lastLine;
    } catch (IOException | InterruptedException e) {
```

Either re-interrupt this method or rethrow the "InterruptedException" that can be caught here.

```
        throw new RuntimeException("Failed to execute Python script", e);
```

Figure 8: Reliability Issues - Third Issue

```
    return lastLine;
- } catch (IOException | InterruptedException e) {
    throw new RuntimeException("Failed to execute Python scr
}
```

```
45      return lastLine;
46+ } catch (IOException e) {
47      throw new RuntimeException(message:"Failed to execute Python
48+ } catch (InterruptedException e) {
49+     Thread.currentThread().interrupt();
50+     throw new RuntimeException(message:"Thread was interrupted w
51  }
52+
```

Figure 9: Reliability Issues - Third Issue Solution

3.3 Maintainability Issues

Figure 10: Maintainability Issues Overview

```
package com.example.sortview.UI;
```

Rename this package name to match the regular expression '^[a-z_]+(\.[a-z_][a-z0-9_]*)*$'.

Figure 11: Maintainability Issues - First Issue

```
package com.example.sortview.UI;
```

```
1+ package com.example.sortview.ui;
```

Figure 12: Maintainability Issues - First Issue Solution

```
if (files != null && files.size() > 0) {
```

Use isEmpty() to check whether the collection is empty or not.

Figure 13: Maintainability Issues - Second Issue

```
if (files != null && files.size() > 0) {
```

```
47+ if (files != null && !files.isEmpty()) {
```

Figure 14: Maintainability Issues - Second Issue Solution

```
ArrayList<String> ClassesNamesOld = null;
```

Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'.

Figure 15: Maintainability Issues - Third Issue

```
ArrayList<String> ClassesNamesOld = null;
try (Connection conn = dbManager.connectToDatabase()) {
    ClassesNamesOld = dbManager.getClassesNamesOld(conn);
```

```
56+ ArrayList<String> classesNamesOld = null;
57  try (Connection conn = dbManager.connectToDatabase()) {
58+     classesNamesOld = dbManager.getClassesNamesOld(conn);
```

Figure 16: Maintainability Issues - Third Issue Solution

```
public ArrayList<String> getClassesNamesNew(Connection conn) {
```

The return type of this method should be an interface such as "List" rather than the implementation "ArrayList".

Figure 17: Maintainability Issues - Fourth Issue

```
public ArrayList<String> getClassesNamesNew(Connection conn)
    String getClasses = "SELECT DISTINCT className FROM "+DB

    ArrayList<String> classList = new ArrayList<>();
```

```
134+ public List<String> getClassesNamesNew(Connection conn) {
135      String getClasses = "SELECT DISTINCT className FROM "+D
136
137+     List<String> classList = new ArrayList<>();
```

Figure 18: Maintainability Issues - Fourth Issue Solution

```
}
if (classesNamesOld.size() < classifications.size()) {
    if (!classesNamesOld.contains(s)) {
```

Merge this if statement with the enclosing one.

```
        createChild(s);
```

Figure 19: Maintainability Issues - Fifth Issue

```
    logger.error("An error occurred: ", e);
}
if (classesNamesOld.size() < classifications.size()) {
    if (!classesNamesOld.contains(s)) {
        createChild(s);
        try (Connection conn = dbManager.connectToDatabase()
```

```
62      logger.error(msg:"An error occurred: ", e);
63  }
64+ if (classesNamesOld.size() < classifications.size() && !classesN
65      createChild(s);
66      try (Connection conn = dbManager.connectToDatabase()) {
```

Figure 20: Maintainability Issues - Fifth Issue Solution

```
} catch (IOException e) {
    System.out.println("Error reading the image: " + e.getMessage());
```

Replace this use of System.out by a logger.

Figure 21: Maintainability Issues - Sixth Issue

```
atch (IOException e) {
System.out.println("Error reading the image: " + e.getMessage())
```

```
63  atch (IOException e) {
64+ logger.error("Error reading the image: {}", e.getMessage(), e);
65
```

Figure 22: Maintainability Issues - Sixth Issue Solution

3.4 Security Hotspots

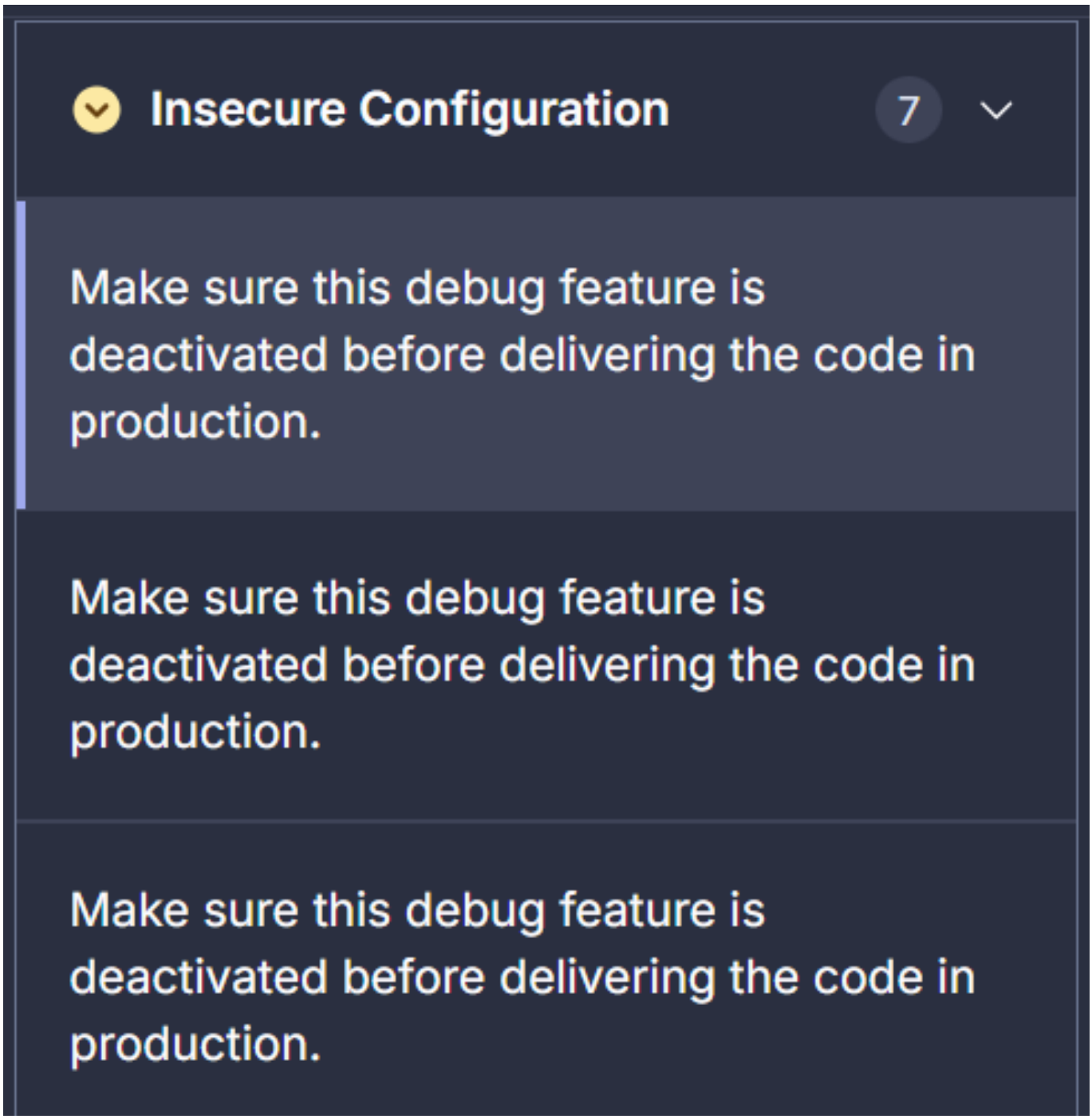

Figure 23: Security Hotspots Overview

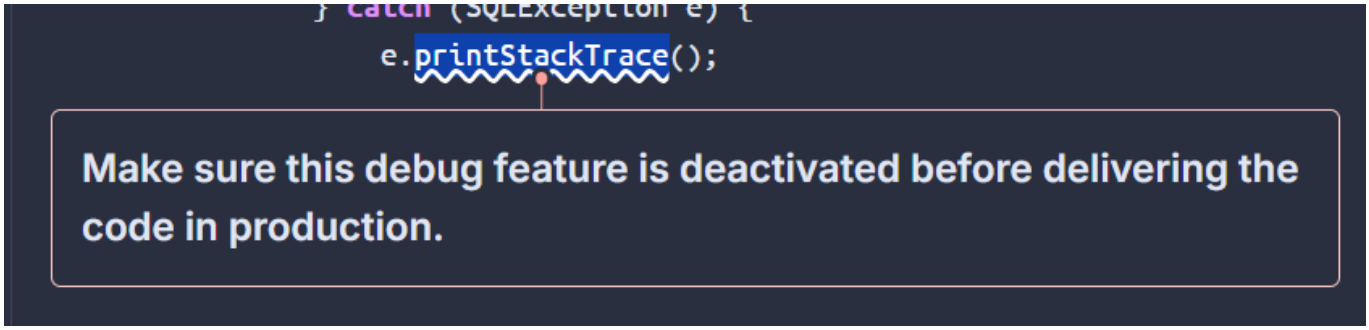

Figure 24: Security Hotspots - First Issue

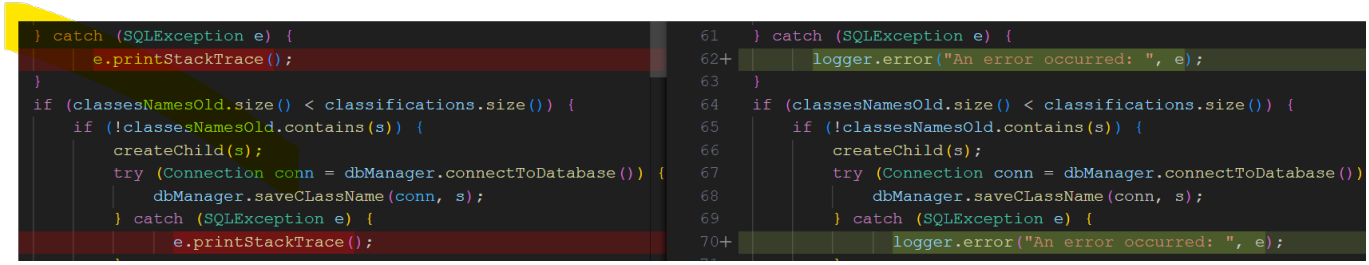

Figure 25: Security Hotspots - First Issue Solution

4 Results and Verification

4.1 SonarQube Results After Maintenance

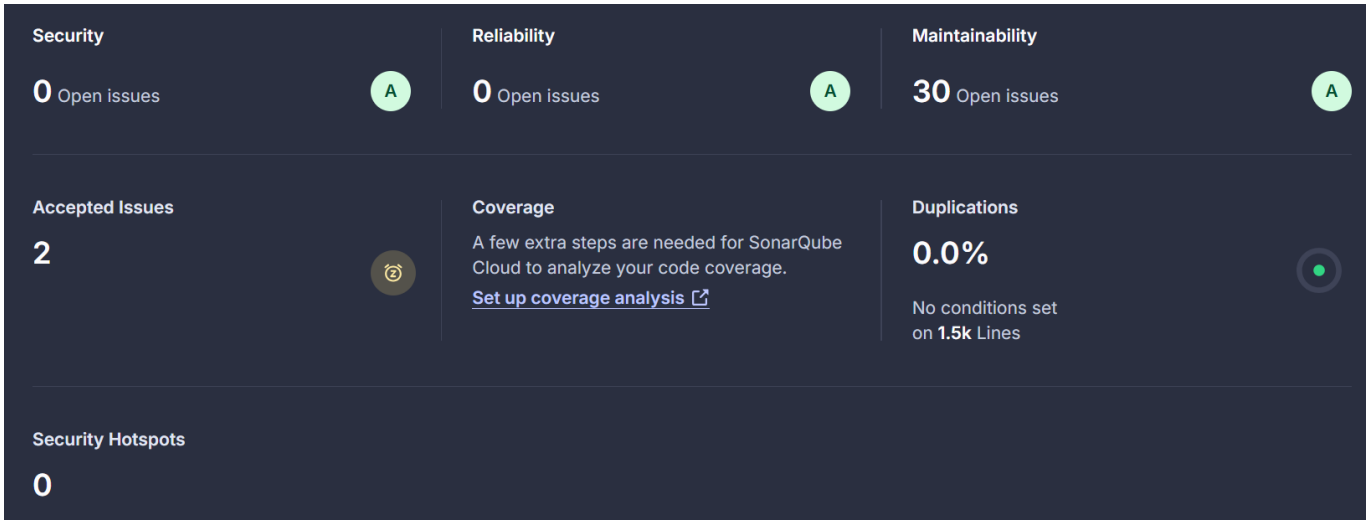

Figure 26: SonarQube Metrics Overview After Maintenance

Metric	Before	After
Security Issues	2	0
Reliability Issues	3	0
Maintainability Issues	40	30
Accepted Issues	0	2
Coverage	Not configured	Not configured
Duplications	0.0%	0.0%
Security Hotspots	8	0

Table 1: SonarQube Analysis Results - Before vs. After