SonarQube Integration with Address Book Application

Submitted By: Aryan Chaturvedi
Project: addressbook-app
Tool Used: SonarQube (Local Server)
IDE: Eclipse
Build Tool: Maven

Objective of the Assignment

The objective of this task was:

- To integrate **SonarQube** into a Java project.
- To create and integrate an **Address Book module**.
- To perform static code analysis.
- To generate code coverage using JaCoCo.
- To execute full Sonar analysis using Maven.
- To document the step-by-step implementation process.

Project Overview – Address Book Module

The developed project is a **Java-based Address Book Application**.

Modules Implemented

1. Model Layer

- `ContactPerson`
  - Fields: id, name, phone, email
  - Getters & Setters
  - Constructor
  - toString()

2. DAO Layer

- `ContactDAO`
    - Add Contact
    - Delete Contact
    - Update Contact
    - Get All Contacts

3. Service Layer

- `ContactService`
    - Business logic
    - Duplicate handling
    - Validations
    - Exception handling

4. Main Class

- `AddressBookMain`
    - Console-based interaction
    - Demonstrates execution

5. Test Layer

- `ContactServiceTest`
    - Written using JUnit 5
    - Covers:
        - Add Contact
        - Delete Contact
        - Update Contact
        - Exception scenarios

Tools & Technologies Used

- Java 21 (LTS)
- Eclipse IDE
- Maven
- JUnit 5
- JaCoCo Plugin
- SonarQube (Localhost:9000)

Step-by-Step Implementation Procedure

STEP 1: Install & Setup SonarQube

1. Download SonarQube (Community Edition).
2. Extract the folder.
3. Navigate to:

```
sonarqube/bin/windows-x86-64
```

4. Run:

```
StartSonar.bat
```

5. Open browser:

```
http://localhost:9000
```

6. Default login:

- Username: admin
- Password: admin

7. Generate new authentication token.

STEP 2: Create Maven Project in Eclipse

1. Open Eclipse.
2. Create → Maven Project.
3. GroupId: `com.addressbook`
4. ArtifactId: `addressbook-app`
5. Packaging: jar

STEP 3: Configure pom.xml

Java Configuration

```
<maven.compiler.release>21</maven.compiler.release>
```

(Java 21 used because JaCoCo supports up to 21 properly)

Add JUnit Dependency

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.10.2</version>
    <scope>test</scope>
</dependency>
```

Add Surefire Plugin

Required to execute JUnit 5 tests.

Add JaCoCo Plugin

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.11</version>
</plugin>
```

Functions:

- `prepare-agent`
- `report`

SonarQube Properties

```
<sonar.projectKey>addressbook-app</sonar.projectKey>
<sonar.host.url>http://localhost:9000</sonar.host.url>
```

STEP 4: Write Unit Test Cases

Created test class:

```
ContactServiceTest
```

Covered:

- Valid input
- Invalid input
- Edge cases
- Exception scenarios

STEP 5: Execute Maven Command

Final command used:

```
mvn clean verify sonar:sonar \
-Dsonar.projectKey=addressbook-app \
-Dsonar.host.url=http://localhost:9000 \
-Dsonar.token=YOUR_NEW_TOKEN
```

Explanation:

Command	Purpose
clean	Removes old compiled files
verify	Runs tests + generates coverage
sonar:sonar	Sends report to SonarQube
sonar.token	Authenticates project

STEP 6: Project Execution Screenshot

Screenshot 1: Eclipse Project Execution

(Attach screenshot showing AddressBookMain running successfully in Console.)

STEP 7: SonarQube Analysis Report Screenshot

Screenshot 2: Sonar Dashboard (Attached Above)

From the screenshot:

- Quality Gate: PASSED
- Security Issues: 0
- Reliability: A

- Maintainability: A
- Coverage: 47.1%
- Duplications: 0%
- Security Hotspots: 0

This confirms:

- No critical vulnerabilities
- Good code structure
- Moderate test coverage
- Clean duplication handling

Coverage Report (JaCoCo)

Generated at:

```
target/site/jacoco/index.html
```

JaCoCo provided:

- Line coverage
- Branch coverage
- Method coverage
- Class coverage

Coverage achieved: **47.1%**

(Improvement possible by adding more test cases.)