Department di Informatica

Università degli Studi di Salerno

Dependability analysis on library management system

Jan 20th, 2024

Project Report By

ismail ali darez

Mustafa hafed abuzaraiba

Oussama boukhachem

Supervised by:

Prof: Dario Di nucci

Table of Contents

Chapter 1: Introduction

This report is about the different dependability analysis which ran on library management system project.

1.1 Library Management System project

The Library Management System encompasses features to add, update, view, and delete books, authors, categories, and publishers. Librarians can efficiently manage book details, author information, category classifications, and publisher data through a user-friendly interface. This system streamlines library operations, ensuring accurate and organized record-keeping.

1.2 Report Structure

This report details the methodologies employed to assess the reliability, dependability, and security of the project. The introductory chapter outlines the project, defines the report's scope, and discusses external software and tools utilized in system development, software quality measurement, and test case generation.

The central chapter elaborates on the analyses, procedures, results, and subsequent changes, encompassing system building, Sonar Cloud for quality analysis, JaCoCo for code coverage, Pitest Mutation testing, JMH for performance testing, and regression test case generation.

The third chapter delves into code blocks and additional plugins used in the project, while the fourth and fifth chapters present conclusions and references, respectively.

1.3 External tools and software

1.3.1 Tools and software

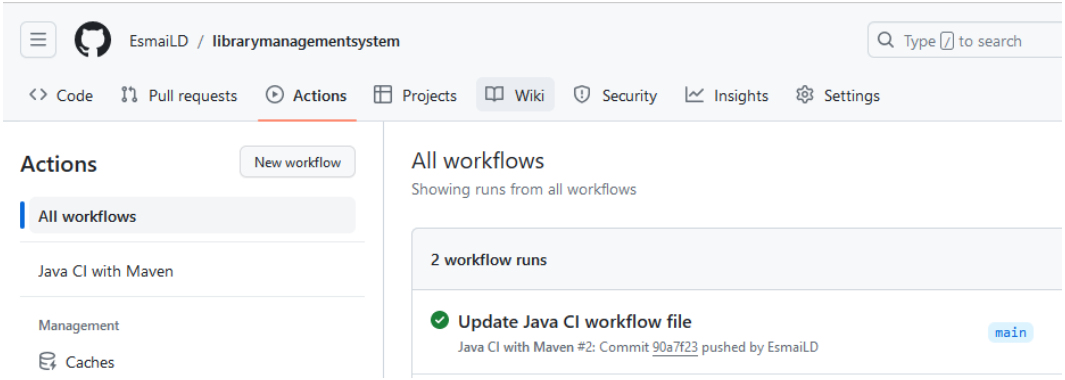
For coding intelli J IDEA, github desktop version controlling, maven for building

1.3.2 Building the system

For building the system I used Maven latest version. I also used JDK 17 in some analysis and JDK 8 in some others. I used JDK for compiling the Java sources, running unit tests, or sign JARs.
Maven was used for generating jars, or run test cases in some scenarios.

CI / CD process

Github actions workflows can make the build CI (continuous integration) process. As given:



CI workflow file is given in section 3.4 Github CI workflow file

As the project is a dynamic web application so CD (continuous deployment) was not possible with github pages. Source: https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages

For CD part, it requires a separate purchasing a hosting provider such as vps, dedicated, etc applicable. Like vps was used here https://blog.tericcabrel.com/springboot-github-actions-ci-cd/ build image was packed with docker and deployed to vps server.

Chapter 2: Software Dependability Analysis

2.1 Building the Project

This project was started by forking the project in my GitHub (https://github.com/EsmaiLD/librarymanagementsystem/tree/main/jmh-reports)

Step 1: Download or clone the source code from GitHub to the local machine.

Step 2: Install JDK 17 - https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html

Step 3: Install IntelliJ IDEA or Eclipse or Apache NetBeans IDE

Step 4: Install Apache Maven - https://maven.apache.org/install.html

Step 5: mvn clean install

Step 6: mvn spring-boot:run

Step 7: From the browser call the endpoint http://localhost:9080

Step 8: Admin Login User Id: admin@admin.in & Password: longp@$$w0rdn0w

2.2 SonarCloud Analysis

I used SonarCloud analysis for checking code quality and security. It utilizes static code analysisand detects bugs, vulnerabilities, and code smells in my projects.

In my initial check with sonar cloud, I obtained following results:

As per the severity levels, 11 high severity, 29 Medium and 5 low were fixed. some were false positive (such as including variable naming that wasn't true because of java conventions.)

2.3 Findings and Refactoring

a. Findings

High severity issues

1. Use static access with "com.opencsv.ICSVWriter" for "NO_QUOTE_CHARACTER".
2. Use static access with "com.opencsv.ICSVWriter" for "DEFAULT_SEPARATOR".
3. Inject this field value directly into "authenticationProvider", the only method that uses it.
4. Define a constant instead of duplicating this literal "author" 5 times.
5. Define a constant instead of duplicating this literal "redirect:/authors" 3 times.
6. Define a constant instead of duplicating this literal "redirect:/books" 3 times.
7. Define a constant instead of duplicating this literal "category" 5 times.
8. Define a constant instead of duplicating this literal "redirect:/categories" 3 times.
9. Define a constant instead of duplicating this literal "publisher" 5 times.

10. Define a constant instead of duplicating this literal "redirect:/publishers" 3 times.
11. Compromised security password

More details:

https://sonarcloud.io/project/issues?impactSeverities=HIGH&resolutions=FIXED&id=EsmaiLD_librarymanagementsystem

Medium severity issues

1. Remove this field injection and use constructor injection instead. x4 times
2. Replace this usage of 'Stream.collect(Collectors.toList())' with 'Stream.toList()' x7 times
3. Add "lang" and/or "xml:lang" attributes to this "<html>" element x16 times
4. Call "item.isPresent()" or "!item.isEmpty()" before accessing the value.
5. Add a private constructor to hide the implicit public one.

More details:

https://sonarcloud.io/project/issues?impactSeverities=MEDIUM&resolutions=FIXED&id=EsmaiLD_librarymanagementsystem

Low severity issues

1. Immediately return this expression instead of assigning it to the temporary variable "bookPage".
2. Immediately return this expression instead of assigning it to the temporary variable "bookVo".
3. Immediately return this expression instead of assigning it to the temporary variable "authorVo".
4. Immediately return this expression instead of assigning it to the temporary variable "categoryVo".
5. Immediately return this expression instead of assigning it to the temporary variable "publisherVo".

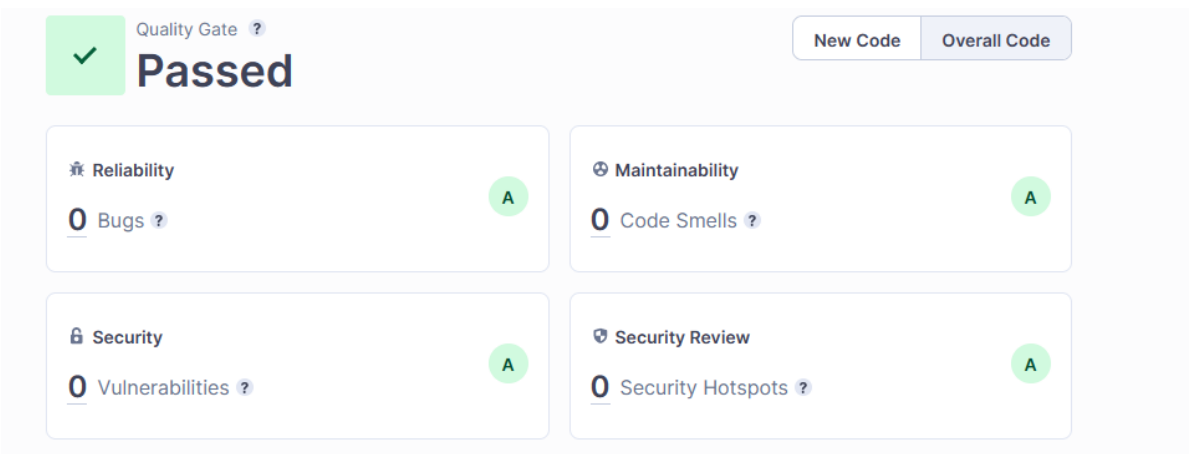
More details:

https://sonarcloud.io/project/issues?impactSeverities=LOW&resolutions=FIXED&id=EsmaiLD_librarymanagementsystem

b. Refactoring

As of the above mentioned issues, as required some were fixed and some were refactored respectively. Pull requests were used to merge other branch changes into main branch.

In my final check with sonar cloud, I obtained following results:



2.4 Skipped Refactoring:

1. Maintainibility = 5 were skipped (replacing transactions with service layer code - refactor task) and rest were changing variable conventions (false positives) I had coding breaking changes that was not build despite suggested refactoring. Other solutions were not accepted by sonar cloud bot. So this was skipped.
2. Security = 13 , Replacing Persistent entities at controllers with request DTOs is a best practice in context of security. So that data is validated/handled at DTO layer then parsed to entity classes for DB operations. However by this change we were getting into entire UAC testing of the project which was not feasible given the time constraints.

2.5 Code coverage

Code coverage measures how much of a software's source code is tested by a specific set of testcases. I have carried out coverage of this project by two ways:

2.5.1 Via JaCoCo

JaCoCo is a code coverage tool designed for Java applications. It provides detailed information about how much Java code is executed during test runs. I integrated JaCoCo into my build process and generated a comprehensive coverage report. Dependency plugin entry can be found in code blocks section.

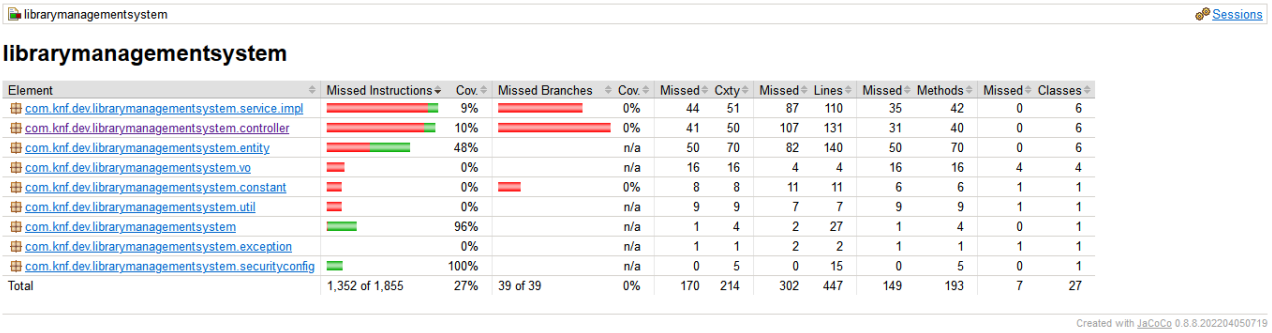
Execution command:

mvn clean test

mvn jacoco:prepare-agent

mvn jacoco:report

Here's a brief report snap:


librarymanagementsystem

Sessions

librarymanagementsystem

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.knf.dev.librarymanagementsystem.service.impl		9%		0%	44	51	87	110	35	42	0	6
com.knf.dev.librarymanagementsystem.controller		10%		0%	41	50	107	131	31	40	0	6
com.knf.dev.librarymanagementsystem.entity		48%		n/a	50	70	82	140	50	70	0	6
com.knf.dev.librarymanagementsystem.vo		0%		n/a	16	16	4	4	16	16	4	4
com.knf.dev.librarymanagementsystem.constant		0%		0%	8	8	11	11	6	6	1	1
com.knf.dev.librarymanagementsystem.util		0%		n/a	9	9	7	7	9	9	1	1
com.knf.dev.librarymanagementsystem		96%		n/a	1	4	2	27	1	4	0	1
com.knf.dev.librarymanagementsystem.exception		0%		n/a	1	1	2	2	1	1	1	1
com.knf.dev.librarymanagementsystem.securityconfig		100%		n/a	0	5	0	15	0	5	0	1
Total	1,352 of 1,855	27%	39 of 39	0%	170	214	302	447	149	193	7	27

Created with JaCoCo 0.8.8.202204050719

Report link: https://github.com/EsmaiLD/librarymanagementsystem/tree/main/jacoco

2.5.2 Mutation testing by PI test

Mutation testing is a method employed to evaluate the effectiveness of a test suite by injecting simulated defects, referred to as mutations, into the code. Subsequently, it verifies whether the tests can successfully identify these mutations. To initiate this testing process, I incorporated the PIT Plugin and JUnit 5 plugin, with the dependency script detailed in the code blocks chapter. Following the additions, I proceeded to build the system and executed the **mvn org.pitest:pitest-maven:mutationCoverage** command. Generated report snap:

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
22	33%	145/441	0%	0/171	0%	0/22

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.knf.dev.librarymanagementsystem	1	93%	25/27	0%	0/13	0%	0/13
com.knf.dev.librarymanagementsystem.constant	1	0%	0/11	0%	0/6	0%	0/0
com.knf.dev.librarymanagementsystem.controller	6	18%	24/131	0%	0/69	0%	0/0
com.knf.dev.librarymanagementsystem.entity	6	41%	58/140	0%	0/26	0%	0/5
com.knf.dev.librarymanagementsystem.securityconfig	1	100%	15/15	0%	0/4	0%	0/4
com.knf.dev.librarymanagementsystem.service.impl	6	21%	23/110	0%	0/45	0%	0/0
com.knf.dev.librarymanagementsystem.util	1	0%	0/7	0%	0/8	0%	0/0

Report generated by PIT 1.13.0

Enhanced functionality available at arcmutate.com

Report link: https://github.com/EsmaiLD/librarymanagementsystem/tree/main/pit-reports

2.6 Performance testing by Java Microbenchmark Harness

The Java Microbenchmark Harness (JMH) is a framework offered by OpenJDK that allows to create and execute microbenchmarks for Java code. Microbenchmarks are designed to assess the performance and behavior of specific code snippets or methods.

I used JMH for measuring performance of my methods. I started this analysis by adding JMH dependency in my pom.xml and the script can be found in code blocks chapter. Public methods from Mapper class were the candidates for testing. So I created a class MapperBenchmark for performing the tests.

I imported the relevant packages and annotations from the `org.openjdk.jmh.annotations` package, which is used for benchmarking purposes. The code sets up a benchmark environment using JMH annotations to measure the performance of the methods in Mapper class.

@Setup init() methods sets up the code required by other

methods i.e

testPublisherModelToVo(),testBookModelToVo(),

testAuthorModelToVo(), testCategoryModelToVo()

Following is the Benchmark Configuration:

1. @BenchmarkMode(Mode.AverageTime): Specifies that the benchmark mode is set to measure the average time taken by the benchmarked method.

2. @OutputTimeUnit(TimeUnit.NANOSECONDS): Sets the time unit for reporting benchmark results to nanoseconds, providing a standardized measurement.

3. @Warmup(iterations = 5, time = 1, timeUnit = TimeUnit.SECONDS): Configures the warm-up phase with 5 iterations, each lasting 1 second, using seconds as the time unit.

4. @Measurement(iterations = 5, time = 1, timeUnit = TimeUnit.SECONDS): Sets the measurement phase with 5 iterations, each lasting 1 second, and using seconds as the time unit.

5. @Fork(1): Specifies that the benchmark should be forked once, indicating how many times the entire set of benchmarks should be executed.

6. @State(Scope.Benchmark): Defines the scope of the state object for benchmark methods, indicating that the state should be shared among all threads participating in the benchmark.

Here's the brief snippet from the report: Complete report can be found here:
https://github.com/EsmaiLD/librarymanagementsystem/tree/main/jmh-reports

Console output:

```
# Run complete. Total time: 00:00:48

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

NOTE: Current JVM experimentally supports Compiler Blackholes, and they are in use. Please exercise
extra caution when trusting the results, look into the generated code to check the benchmark still
works, and factor in a small probability of new VM bugs. Additionally, while comparisons between
different JVMs are already problematic, the performance difference caused by different Blackhole
modes can be very significant. Please make sure you use the consistent Blackhole mode for comparisons.

Benchmark                                        Mode  Cnt  Score   Error  Units
MapperBenchmark.testAuthorModelToVo              avgt    5  61.155 ± 20.381  ns/op
MapperBenchmark.testBookModelToVo                avgt    5  59.409 ± 10.584  ns/op
MapperBenchmark.testCategoryModelToVo            avgt    5  59.506 ± 16.781  ns/op
MapperBenchmark.testPublisherModelToVo           avgt    5  56.174 ±  8.843  ns/op

Process finished with exit code 0
```

2.7 Regression test cases generation via Randoop

Randoop is another tool used for automated software testing. It generates testsautomatically by analyzing the code under the test. Randoop is mainly in unit testing. Developers used this tool to create test suites to validate the functionality of their code.

Installation: For making use of randoop I firstly cloned randoop latest release on my system and extracted the archive.

%RANDOOP_JAR% is the environment variable i.e E:\Downloads\randoop-4.3.2\randoop-all-4.3.2.jar

Execution command:

```
java -classpath E:\Downloads\randoop-4.3.2;%RANDOOP_JAR% randoop.main.Main gentests --testjar=librarymanagementsystem-0.0.1-SNAPSHOT.jar
```

Regression test java files and logged output can be found here

Report link: https://github.com/EsmaiLD/librarymanagementsystem/tree/main/randoop%20reports

2.8 Security Analysis with FindSecbugs

FindSecBugs is a security analysis tool designed to identify bugs and vulnerabilities. I initiated the analysis by locally cloning it onto my machine. Subsequently, I successfully executed the findsecbugs.bat file. Please note that I utilized the .bat file due to running it on a Windows operating system.

Upon installing the FindSecBugs.jar from GitHub, I ran the .bat file with the path to my project. The outcome revealed no potential security bugs in my project. The attached file provides a visual representation of the results.

Console output:

```
E:\Downloads\findsecbugs-cli-1.12.0>findsecbugs.bat -progress -html -output report.htm -onlyAnalyze
com.knf.dev.librarymanagementsystem.*  librarymanagementsystem-0.0.1-SNAPSHOT.jar

SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#noProviders for further details. Scanning archives (5/5)
2 analysis passes to perform
Pass 1: Analyzing classes (32927 / 32927) - 100% complete
Pass 2: Analyzing classes (0/30798) - 00% complete Done with analysis
```

SpotBugs Report

Project Information

Project:

SpotBugs version: 4.6.0

Code analyzed:

- E:\Downloads\librarymanagementsystem\target\librarymanagementsystem-0.0.1-SNAPSHOT.jar
- -maxHeap
- 2000
- -effort
- max

Metrics

1372169 lines of code analyzed, in 30798 classes, in 1751 packages.

Metric	Total	Density*
High Priority Warnings		0.00
Medium Priority Warnings		0.00
Total Warnings	**0**	**0.00**

(Defects per Thousand lines of non-commenting source statements)*

Report link: https://github.com/EsmaiLD/librarymanagementsystem/tree/main/findsecbugs-reports

Chapter 3: Code blocks

3.1. PIT and junit plugin dependency:

```
<plugin>
  <groupId>org.pitest</groupId>
  <artifactId>pitest-maven</artifactId>
  <version>1.1.10</version>
  <configuration>
    <targetClasses>
      <param>com.knf.dev.*</param>
    </targetClasses>
  </configuration>
</plugin>
<plugin>
  <groupId>org.pitest</groupId>
  <artifactId>pitest-maven</artifactId>
  <version>1.13.0</version>
  <dependencies>
    <dependency>
      <groupId>org.pitest</groupId>
      <artifactId>pitest-junit5-plugin </artifactId>
      <version>1.1.2</version>
    </dependency>
  </dependencies>
</plugin>
```

3.2 JMH dependency

```
<dependency>
  <groupId>org.openjdk.jmh</groupId>
  <artifactId>jmh-core</artifactId>
  <version>1.37</version>
</dependency>
<dependency>
  <groupId>org.openjdk.jmh</groupId>
  <artifactId>jmh-generator-annprocess</artifactId>
  <version>1.37</version>
</dependency>
```

3.3 JMH benchmark

```
/**<!--Licensed to the Apache Software Foundation (ASF) under one or morecontributor license
 agreements.  See the NOTICE file distributed with
 this work for additional information regarding copyright ownership. The ASF
 licenses this file to You under the Apache License, Version 2.0 (the "License");
 you may not use this file except in compliance withthe License. You may obtain
 a copy of the License at http://www.apache.org/licenses/LICENSE-2.0
 Unless required by applicable law or agreed to in writing, softwaredistributed
 under the License is distributed on an "AS IS" BASIS,
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.See
 the License for the specific language governing permissions and limitations under the
 License.--> */
```

```java
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@Warmup(iterations = 5, time = 1, timeUnit = TimeUnit.SECONDS)
@Measurement(iterations = 5, time = 1, timeUnit = TimeUnit.SECONDS)
@Fork(1)
@State(Scope.Benchmark)
public class MapperBenchmark {
    private List<Publisher> publishers;

    private List<Author> authors;

    private List<Book> books;

    private List<Category> categories;

    @Setup
    public void init() {
        this.publishers = new ArrayList<>();
        this.authors = new ArrayList<>();
        this.categories = new ArrayList<>();
        this.books = new ArrayList<>();

        Publisher publisher = new Publisher();
        publisher.setId(1L);
        publisher.setName("name");

        Author author = new Author();
        author.setId(1L);
        author.setName("John doe");
        author.setDescription("an author");

        Category category = new Category();
        category.setId(1L);
        category.setName("random category");

        Book book = new Book();
        book.setId(1L);
        book.setIsbn("dummy isbn");
        book.setName("coding book");
        book.setSerialName("serialnumber");


        publishers.add(publisher);
        authors.add(author);
        categories.add(category);
        books.add(book);

    }

    @Benchmark
    public List<PublisherRecord> testPublisherModelToVo() {
        return Mapper.publisherModelToVo(publishers);
    }

    @Benchmark
    public List<BookRecord> testBookModelToVo(){
        return Mapper.bookModelToVo(books);
    }

    @Benchmark
    public List<AuthorRecord> testAuthorModelToVo(){
        return Mapper.authorModelToVo(authors);
    }

    @Benchmark
    public List<CategoryRecord> testCategoryModelToVo(){
        return Mapper.categoryModelToVo(categories);
    }
    public static void main(String[] args) throws Exception {
```

```
        org.openjdk.jmh.Main.main(args);
    }
}
```

3.4 Github CI workflow file

```
# This workflow will build a Java project with Maven, and cache/restore any dependencies to improve the workflow execution time
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-maven

# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

name: Java CI with Maven

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'
        cache: maven
    - name: Build with Maven
      run: mvn clean install

    # Optional: Uploads the full dependency graph to GitHub to improve the quality of Dependabot alerts this repository can receive
    - name: Update dependency graph
      uses: advanced-security/maven-dependency-submission-action@571e99aab1055c2e71a1e2309b9691de18d6b7d6
```

3.5 Jacoco plugin dependency

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.8</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>report</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>report</goal>
```

```
        </goals>
      </execution>
    </executions>
</plugin>
```

Chapter 4: Conclusion

This report is created to fulfill the semester project requirement of the Software Dependability course, showcasing the practical knowledge gained throughout the course. It provides a comprehensive overview of the hands-on experience acquired, specifically focusing on the software dependability analyses performed on the library management system project.

Chapter 5: References

1. Project repository https://github.com/EsmaiLD/librarymanagementsystem
2. Baeldung - Java Mutation Testing with Pitest: https://www.baeldung.com/java-mutation-testing-with-pitest
3. Baeldung - Java Microbenchmark Harness (JMH) - Introduction and Integration. https://www.baeldung.com/java-microbenchmark-harness
4. Randoop: Automated Software Testing for Java. https://randoop.github.io/randoop/
5. FindBugs Security Auditing https://find-sec-bugs.github.io/
6. Baeldung – Intro to jacoco - https://www.baeldung.com/jacoco
7. https://blog.tericcabrel.com/springboot-github-actions-ci-cd/
8. https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages