CS567-THE PROJECT

Introduction:

In brief, the **ProductCrudApp** is a neat and lightweight application for creating, reading, updating and deleting the product data. Built with Java and Maven, it enables customers to create, read, update, and delete product information at ease. The use of the application guarantees a lightweight approach to the management needs of small-scale products. It is easy to navigate permitting its integration into the normal running of the business.

Tools, Dependencies and Jar's Used in ProductCrudApp:

1)Java JRE 23 & Java JDK 23

Java JRE and JDK 23 has being used to run and develop the ProductCrudApp. JDK 23 is used for the development of this application and JRE 23 ensures that it can be run on any machine having a compatible JRE installed. This makes the applications to be compatible with the other and also to perform well.

2)Apache Maven

Apache Maven is employed to handle dependency and build process in this project. It makes the work of accumulating, testing and delivering the application more manageable as well as brings more order into the project, making it more manageable and scalable.

3)Chocolatey (For Cloc Installation)

Chocolatey is package manager for Windows which makes easier for users to install packages or software applications. It is used in this project to install Cloc a codebase size measurement tool which is effective in tracking and analyzing a project.

4)Cloc

To facilitate measuring and analysing the LOC in the project, use Cloc termed as Count Lines of Code. This way, there is an understanding of how big a code set is and how difficult it is to manage because it outlines the overall size of the project.

5)JUnit Jupiter 5

For unit testing the project, JUnit Jupiter 5 is employed. It let you write tests to check that the application behaves right and keeps the code in order while evolving it.

6)PIT (Pitest) Plugin

In this project, mutation testing is performed using the PIT plugin. It contributes to the evaluation of the tests by adding slight alterations into the codes, providing certain mutations to the codes and making sure that the tests can identify these mutations while making certain that the test suite is indeed robust.

7)Randoop

Randoop is used for auto generation of tests in this project. It therefore creates unit tests based on how the code works and the different possibilities in an endeavor of checking the code for possible defects, within the shortest time possible and during development.

Project Structure Overview:

The **ProductCrudApp** project follows a clean structure where different components are responsible for specific functionalities such as handling products, their operations (CRUD), and tests. Below is an outline of the main classes and methods.

1. Main Application Class

- **Purpose**: This class serves as the entry point for the application. It initializes the application and may be used to configure dependencies or launch the application.
- **Main Methods**:
  - `public static void main(String[] args)`: Entry point for the application.

2. Product Class

- **Purpose**: Represents the product entity with attributes like `id`, `name`, `description`, `price`, and `quantity`. It defines the data structure for a product.
- **Main Methods**:
  - Getter and Setter methods for each attribute (e.g., `getId()`, `getName()`, `setName()`, etc.).
  - `toString()`: Returns a string representation of the product.

3. ProductRepository Class

- **Purpose**: Handles the operations related to the storage and retrieval of products. It manages a collection of products in memory (a map or list, depending on the design).
- **Main Methods**:
  - `public List<Product> findByKeyword(String keyword)`: Finds products based on a keyword in their name or description.
  - `public void save(Product product)`: Saves a new product or updates an existing one.
  - `public void delete(Long id)`: Deletes a product by its ID.
  - `public List<Product> findAll()`: Returns all products stored in the repository.

4. Order Class

- **Purpose**: Represents an order placed by a user, containing details like order ID, product ID, quantity, and order status.

- **Main Methods**:
    - Getter and Setter methods (e.g., `getId()`, `getQuantity()`, `setStatus()`).
    - `toString()`: Returns a string representation of the order.

5. OrderStatus Enum

- **Purpose**: Defines the possible statuses of an order (e.g., `PENDING`, `COMPLETED`, `CANCELLED`).
- **Main Methods**:
    - Getter for status names.

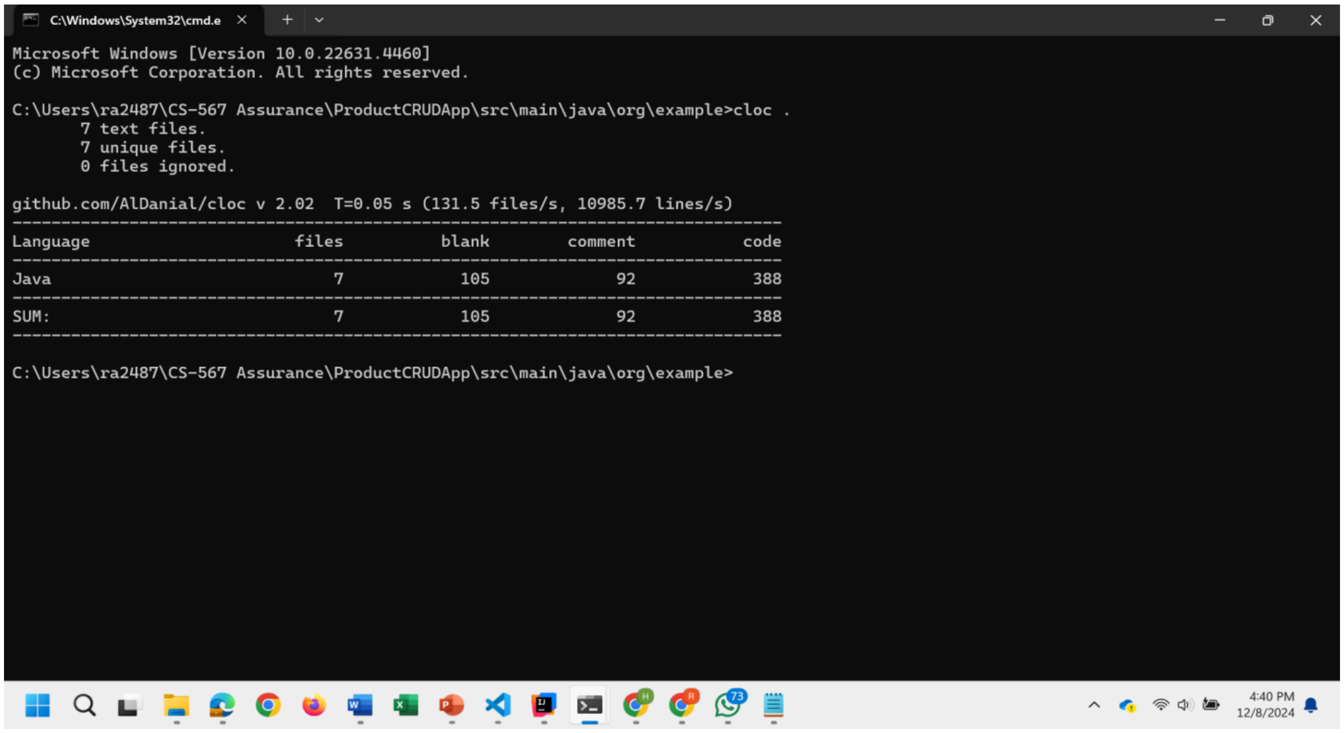
6. InventoryAlert Class

- **Purpose**: Represents alerts when inventory levels are low. This is useful for stock management.
- **Main Methods**:
    - Getter and Setter methods (e.g., `getProductId()`, `getCurrentStock()`, etc.).
    - `toString()`: Returns a string representation of the inventory alert.

Code Measuring Using Cloc Tool:

Command used:

In working directory --- **cloc .**



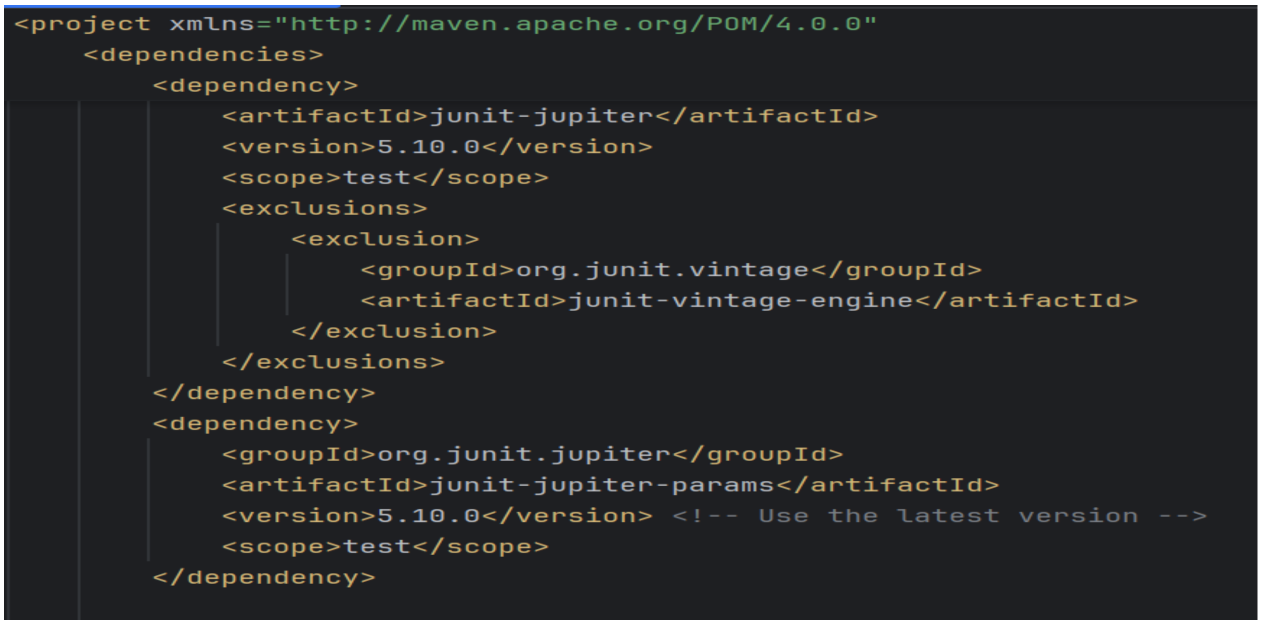
```
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\src\main\java\org\example>cloc .
       7 text files.
       7 unique files.
       0 files ignored.

github.com/AlDanial/cloc v 2.02  T=0.05 s (131.5 files/s, 10985.7 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                             7            105             92            388
-------------------------------------------------------------------------------
SUM:                             7            105             92            388
-------------------------------------------------------------------------------

C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\src\main\java\org\example>
```

```
C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\src\main\java\org\example>cloc .
       7 text files.
       7 unique files.
       0 files ignored.

github.com/AlDanial/cloc v 2.02  T=0.05 s (131.5 files/s, 10985.7 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment       code
-------------------------------------------------------------------------------
Java                            7            105             92          388
-------------------------------------------------------------------------------
SUM:                            7            105             92          388
```

Writing JUnit Tests and Need for Dependencies:

JUnit Dependencies

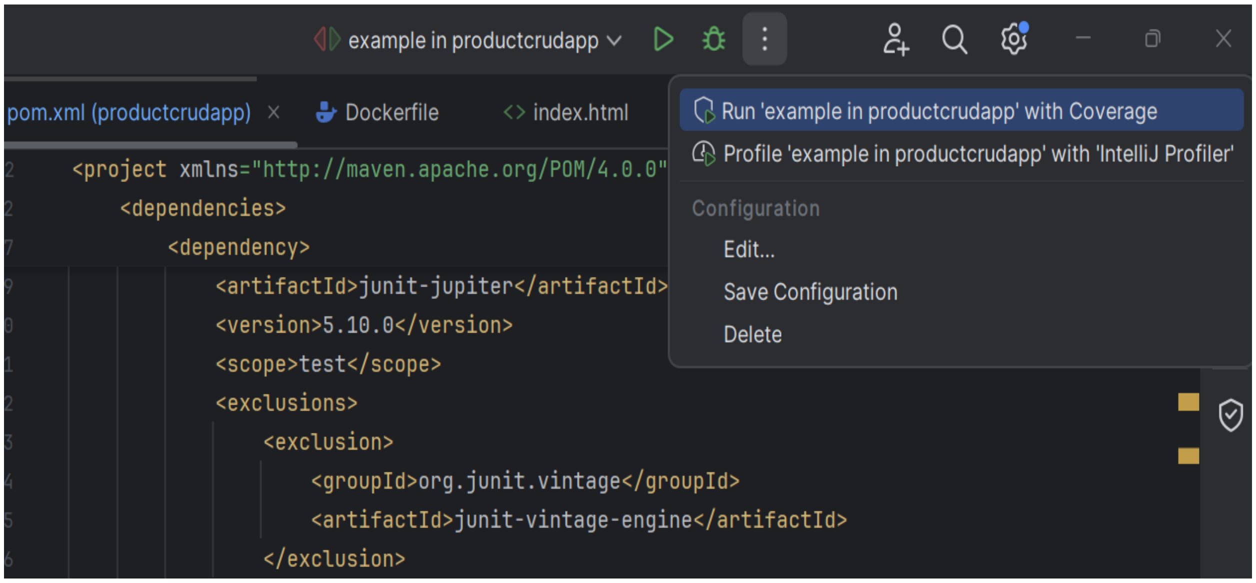
To effectively write and run JUnit tests with coverage for your **ProductCrudApp** project, the following dependencies are necessary. These dependencies help in running the tests, asserting results, and measuring test coverage.



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <dependencies>
        <dependency>
            <artifactId>junit-jupiter</artifactId>
            <version>5.10.0</version>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <version>5.10.0</version> <!-- Use the latest version -->
            <scope>test</scope>
        </dependency>
```

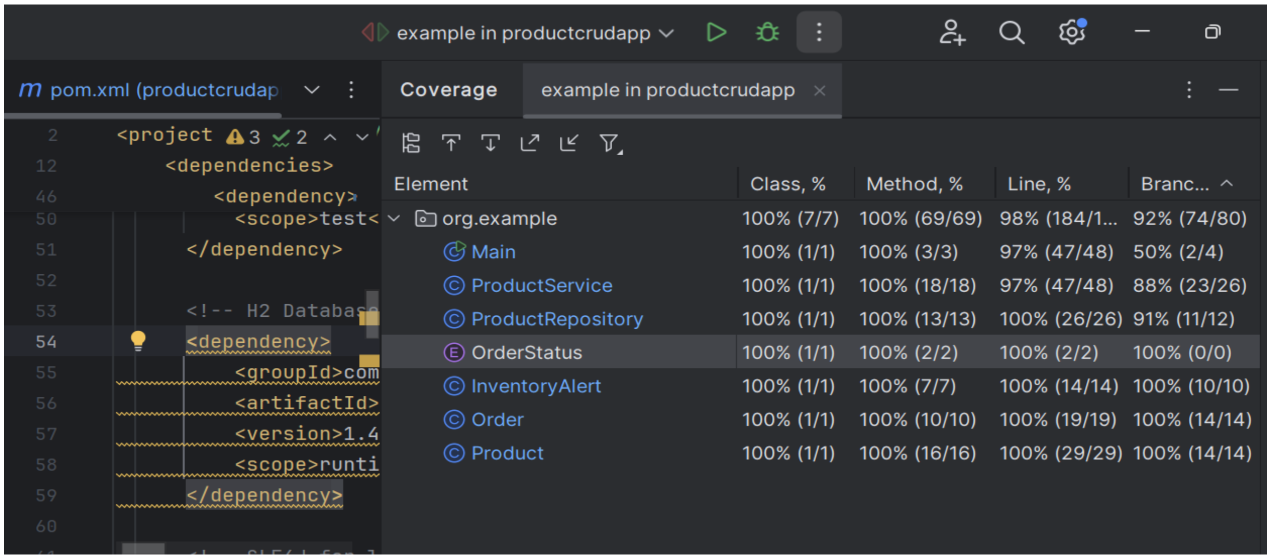
Commands to Run:

After writing the JUnit tests with the proper project structure and adding the required dependencies in the pom.xml file, run the tests with code coverage and make sure to run the command **mvn clean install** to download all the dependencies and plugins


example in productcrudapp
pom.xml (productcrudapp)
Dockerfile
index.html
Run 'example in productcrudapp' with Coverage
Profile 'example in productcrudapp' with 'IntelliJ Profiler'
Configuration
Edit...
Save Configuration
Delete

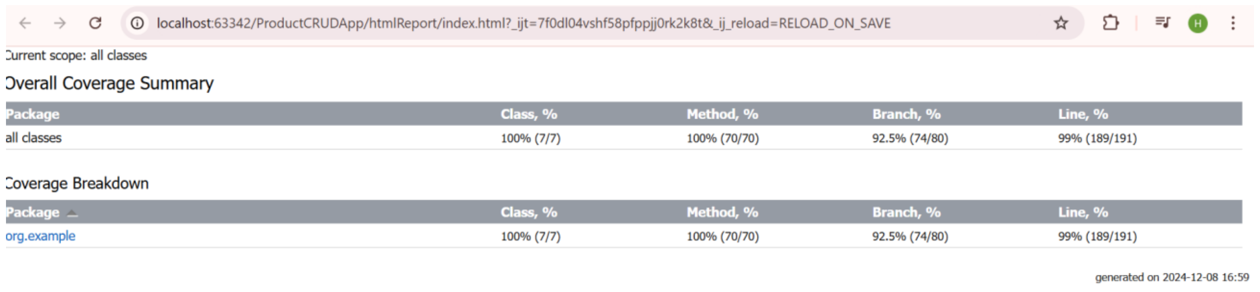
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <dependencies>
        <dependency>
            <artifactId>junit-jupiter</artifactId>
            <version>5.10.0</version>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
```

Code Coverage Report with Junits:



```
<project
    <dependencies>
        <dependency>
          <scope>test<
        </dependency>

        <!-- H2 Databas
        <dependency>
            <groupId>com
            <artifactId>
            <version>1.4
            <scope>runti
        </dependency>
```

Element	Class, %	Method, %	Line, %	Branc...
org.example	100% (7/7)	100% (69/69)	98% (184/1...	92% (74/80)
Main	100% (1/1)	100% (3/3)	97% (47/48)	50% (2/4)
ProductService	100% (1/1)	100% (18/18)	97% (47/48)	88% (23/26)
ProductRepository	100% (1/1)	100% (13/13)	100% (26/26)	91% (11/12)
OrderStatus	100% (1/1)	100% (2/2)	100% (2/2)	100% (0/0)
InventoryAlert	100% (1/1)	100% (7/7)	100% (14/14)	100% (10/10)
Order	100% (1/1)	100% (10/10)	100% (19/19)	100% (14/14)
Product	100% (1/1)	100% (16/16)	100% (29/29)	100% (14/14)



localhost:63342/ProductCRUDApp/htmlReport/index.html?_ijt=7f0dl04vshf58pfppjj0rk2k8t&_ij_reload=RELOAD_ON_SAVE

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	100% (7/7)	100% (70/70)	92.5% (74/80)	99% (189/191)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
org.example	100% (7/7)	100% (70/70)	92.5% (74/80)	99% (189/191)

generated on 2024-12-08 16:59

Mutation Testing Using PIT:

Importance of Mutation Testing

Mutation testing is a vital technique for assessing the effectiveness of your unit tests. It involves intentionally introducing small changes (mutations) into your code to simulate potential faults or errors. The goal is to determine whether your tests can detect these changes. If the tests fail after a mutation, they are considered effective in catching potential issues. However, if the tests pass, it indicates a potential weakness in the test suite, as the mutated code still passes the tests.

The key benefits of mutation testing include:

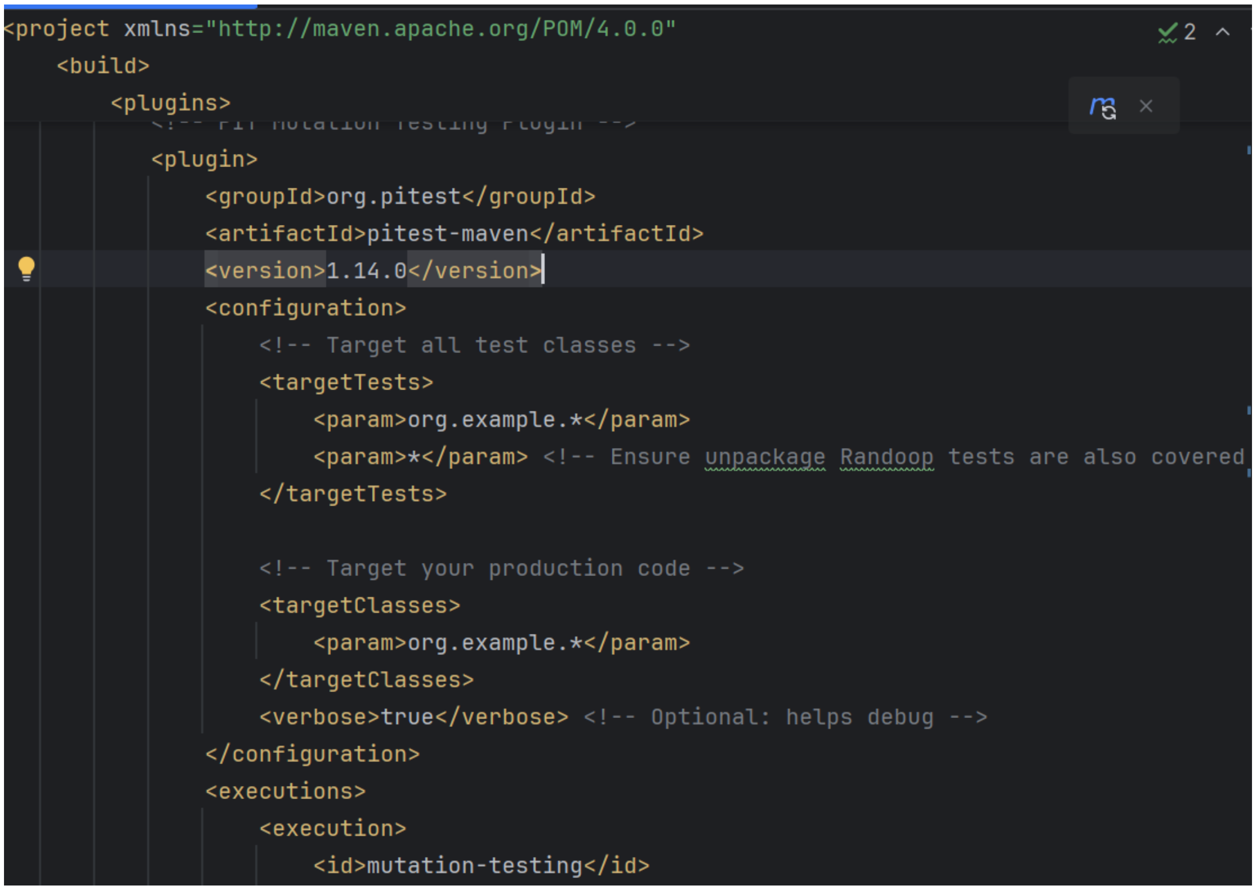
1. **Test Quality Assurance**: It ensures your tests are robust and capable of detecting issues that might arise during code changes.
2. **Uncovering Hidden Bugs**: Even if unit tests pass initially, mutation testing can reveal potential flaws that aren't covered by existing test cases.
3. **Improved Test Suite Effectiveness**: By identifying weak or redundant tests, mutation testing helps improve the overall quality of your testing framework.

How Mutation Testing Is Used in This Project

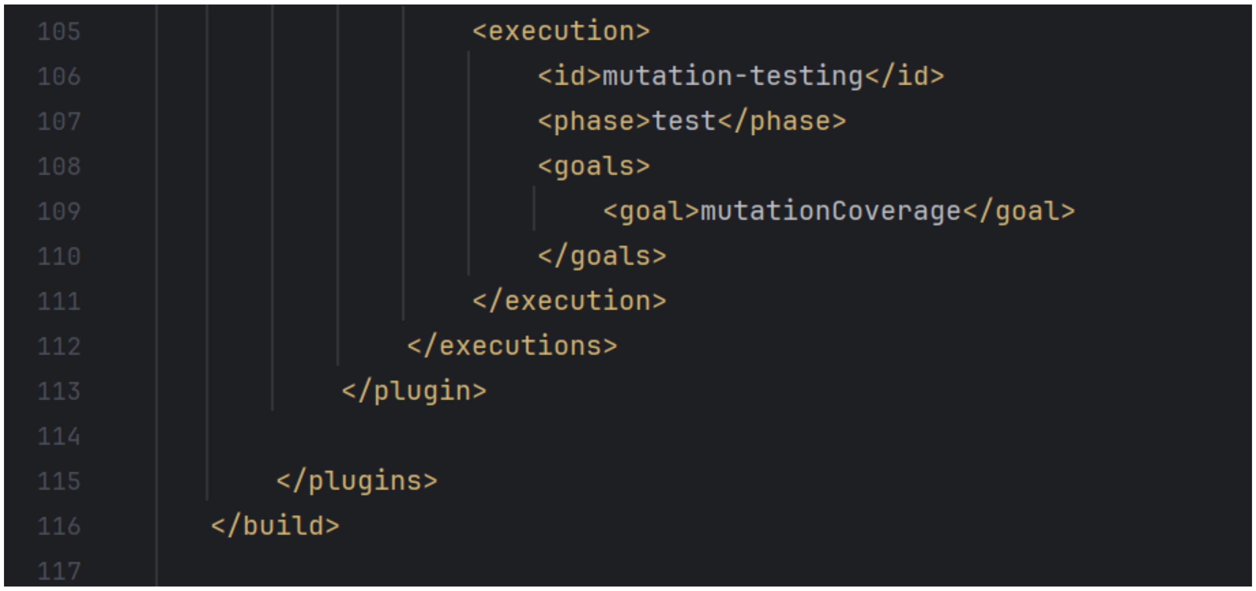
In this project, mutation testing is performed using the **Pitest** plugin. Here's how it's integrated and used:

1. **Setup in pom.xml**: The Pitest plugin is added as a dependency in the pom.xml file of the project. It helps automate the mutation testing process and integrates with Maven.
2. **Execution**: Once the tests are written and configured, the mutation tests are executed using the Maven command mvn pitest:mutationCoverage. Pitest modifies parts of the code (mutations) and runs the existing tests to see if they fail (detect the mutation).
3. **Analyzing Results**: The mutation testing results provide a coverage report, indicating how many mutations were detected by the tests. If a mutation is not detected, it suggests that the corresponding test case did not cover that area of code effectively, prompting an update or addition to the test suite.

Plugins for PIT to perform mutation test:



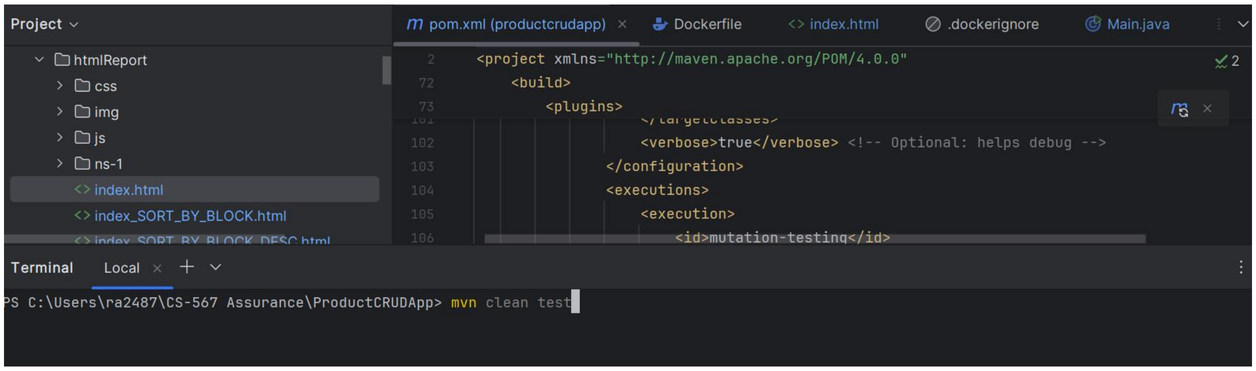
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <build>
        <plugins>
            <!-- PIT mutation Testing Plugin -->
            <plugin>
                <groupId>org.pitest</groupId>
                <artifactId>pitest-maven</artifactId>
                <version>1.14.0</version>
                <configuration>
                    <!-- Target all test classes -->
                    <targetTests>
                        <param>org.example.*</param>
                        <param>*</param> <!-- Ensure unpackage Randoop tests are also covered
                    </targetTests>

                    <!-- Target your production code -->
                    <targetClasses>
                        <param>org.example.*</param>
                    </targetClasses>
                    <verbose>true</verbose> <!-- Optional: helps debug -->
                </configuration>
                <executions>
                    <execution>
                        <id>mutation-testing</id>
```



```xml
                    <execution>
                        <id>mutation-testing</id>
                        <phase>test</phase>
                        <goals>
                            <goal>mutationCoverage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>

        </plugins>
    </build>

```

Command to Run the Mutation test:

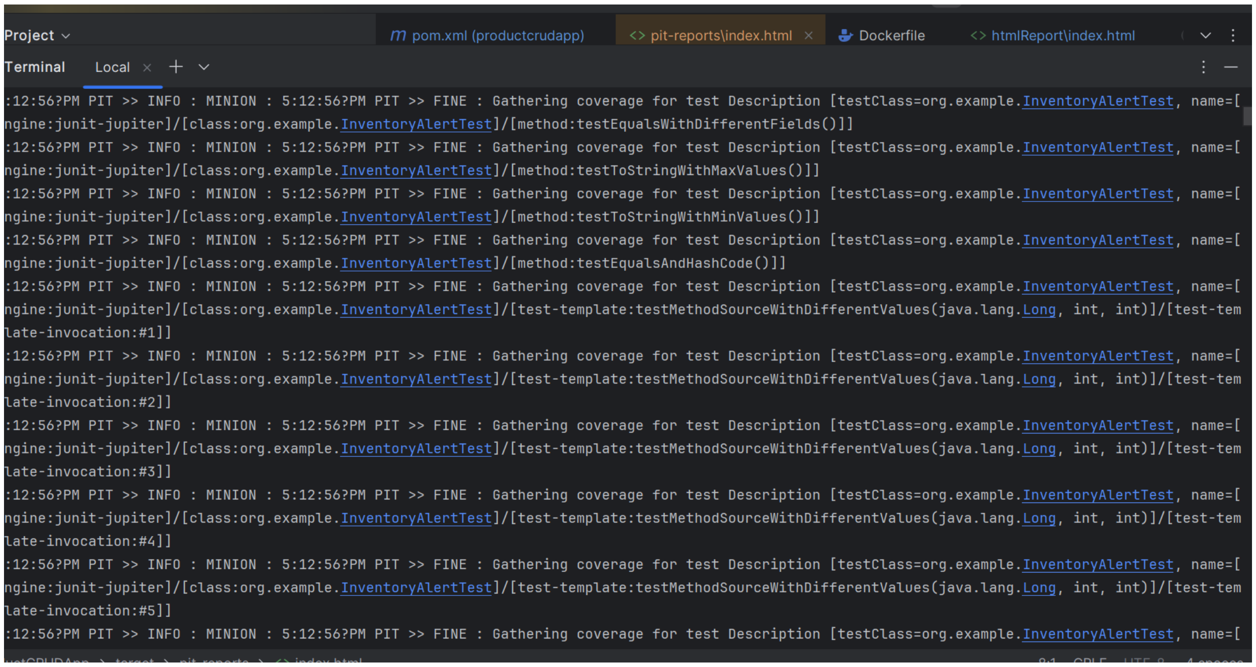
---mvn clean test



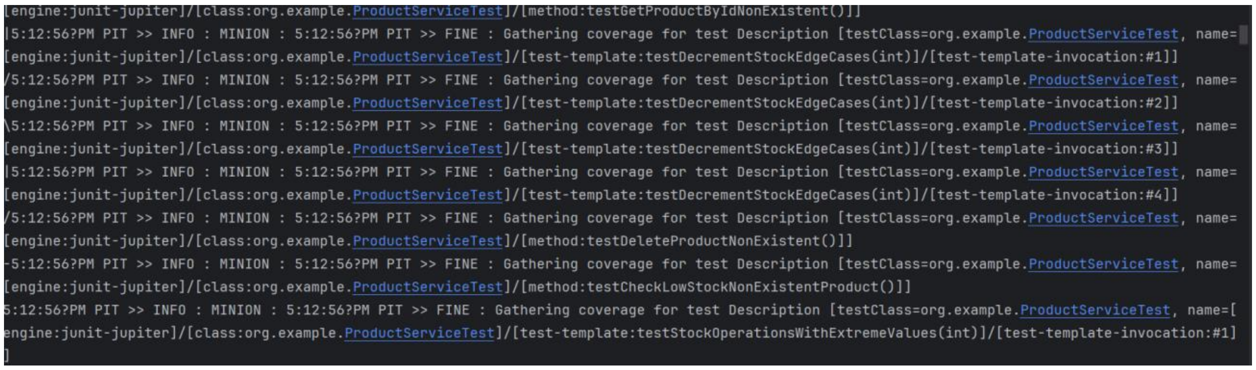
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <build>
        <plugins>
                </targetClasses>
                <verbose>true</verbose> <!-- Optional: helps debug -->
            </configuration>
            <executions>
                <execution>
                    <id>mutation-testing</id>
```

```
PS C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp> mvn clean test
```

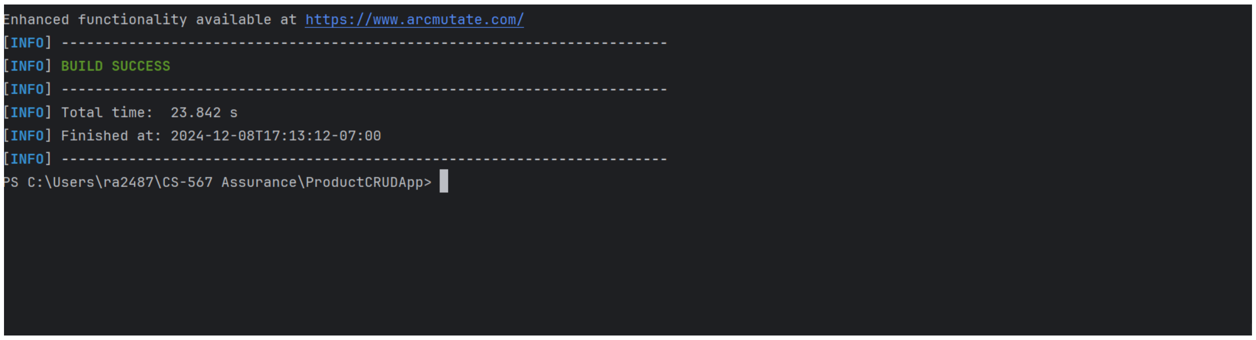
Running Screenshots of Pit Tests:



```
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[method:testEqualsWithDifferentFields()]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[method:testToStringWithMaxValues()]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[method:testToStringWithMinValues()]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[method:testEqualsAndHashCode()]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[test-template:testMethodSourceWithDifferentValues(java.lang.Long, int, int)]/[test-tem
late-invocation:#1]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[test-template:testMethodSourceWithDifferentValues(java.lang.Long, int, int)]/[test-tem
late-invocation:#2]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[test-template:testMethodSourceWithDifferentValues(java.lang.Long, int, int)]/[test-tem
late-invocation:#3]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[test-template:testMethodSourceWithDifferentValues(java.lang.Long, int, int)]/[test-tem
late-invocation:#4]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
ngine:junit-jupiter]/[class:org.example.InventoryAlertTest]/[test-template:testMethodSourceWithDifferentValues(java.lang.Long, int, int)]/[test-tem
late-invocation:#5]]
:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.InventoryAlertTest, name=[
```

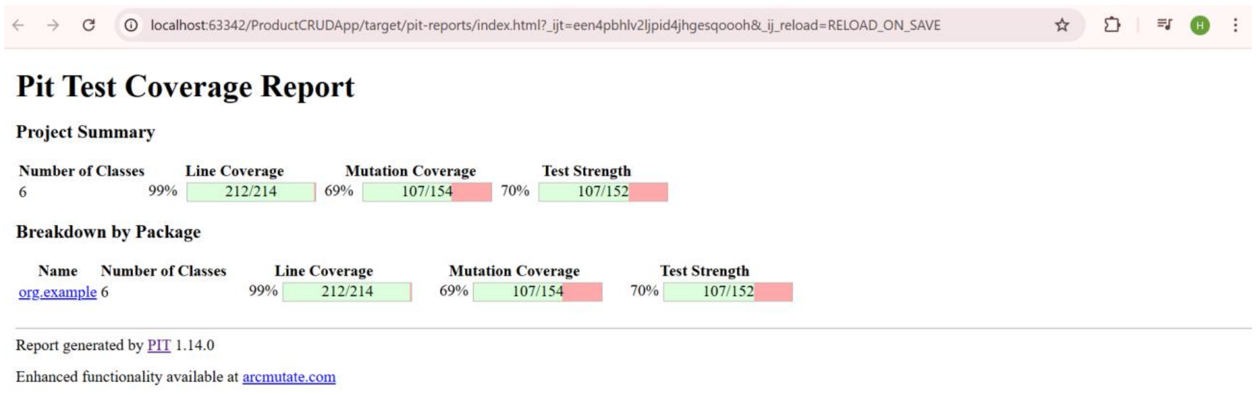


```
[engine:junit-jupiter]/[class:org.example.ProductServiceTest]/[method:testGetProductByIdNonExistent()]]
|5:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.ProductServiceTest, name=
[engine:junit-jupiter]/[class:org.example.ProductServiceTest]/[test-template:testDecrementStockEdgeCases(int)]/[test-template-invocation:#1]]
/5:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.ProductServiceTest, name=
[engine:junit-jupiter]/[class:org.example.ProductServiceTest]/[test-template:testDecrementStockEdgeCases(int)]/[test-template-invocation:#2]]
\5:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.ProductServiceTest, name=
[engine:junit-jupiter]/[class:org.example.ProductServiceTest]/[test-template:testDecrementStockEdgeCases(int)]/[test-template-invocation:#3]]
|5:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.ProductServiceTest, name=
[engine:junit-jupiter]/[class:org.example.ProductServiceTest]/[test-template:testDecrementStockEdgeCases(int)]/[test-template-invocation:#4]]
/5:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.ProductServiceTest, name=
[engine:junit-jupiter]/[class:org.example.ProductServiceTest]/[method:testDeleteProductNonExistent()]]
-5:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.ProductServiceTest, name=
[engine:junit-jupiter]/[class:org.example.ProductServiceTest]/[method:testCheckLowStockNonExistentProduct()]]
5:12:56?PM PIT >> INFO : MINION : 5:12:56?PM PIT >> FINE : Gathering coverage for test Description [testClass=org.example.ProductServiceTest, name=[
engine:junit-jupiter]/[class:org.example.ProductServiceTest]/[test-template:testStockOperationsWithExtremeValues(int)]/[test-template-invocation:#1]
]
```



```
Enhanced functionality available at https://www.arcmutate.com/
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  23.842 s
[INFO] Finished at: 2024-12-08T17:13:12-07:00
[INFO] ------------------------------------------------------------------------
PS C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp>
```

PIT Report before adding Regression tests:



localhost:63342/ProductCRUDApp/target/pit-reports/index.html?_ijt=een4pbhlv2ljpid4jhgesqoooh&_ij_reload=RELOAD_ON_SAVE

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
6	99%	212/214	69%	107/154	70%	107/152

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.example	6	99%	212/214	69%	107/154	70%	107/152

Report generated by PIT 1.14.0

Enhanced functionality available at arcmutate.com

Regression Testing Using Randoop:

What is Randoop?

Randoop is a Java tool packaged as a JAR file used for generating unit tests automatically. It creates random sequences of method calls to test the behavior of your Java classes. This tool helps to identify edge cases and potential bugs that may not be covered by traditional unit testing.

Usage of Randoop in the Project

In the ProductCrudApp project, Randoop is beneficial for:

1. Automatic Test Generation: Randoop generates unit tests for core classes (like `ProductService` and `ProductRepository`) by invoking methods randomly, which helps to uncover untested scenarios.

2. Improved Test Coverage: It automatically fills gaps in the testing coverage by generating tests for methods that might be overlooked manually.

3. Regression Testing: Regular test executions using Randoop ensure that code changes don't introduce new issues, helping maintain project stability.

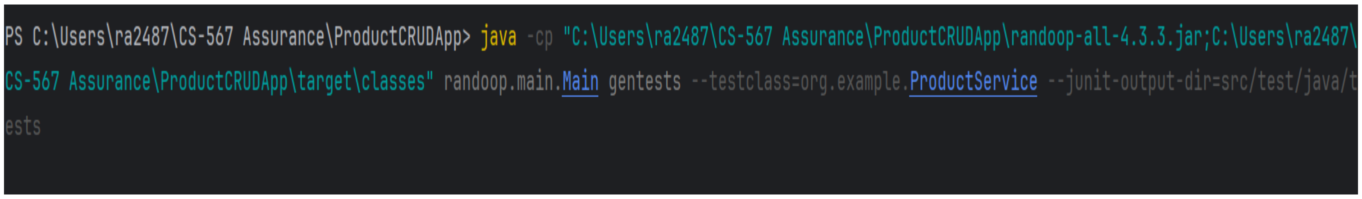
How Randoop Helps

- Jar File: Randoop is a JAR file that can be executed from the command line or integrated into build processes.

- Enhances Test Coverage: By running Randoop tests, the project benefits from greater test coverage and improved stability.

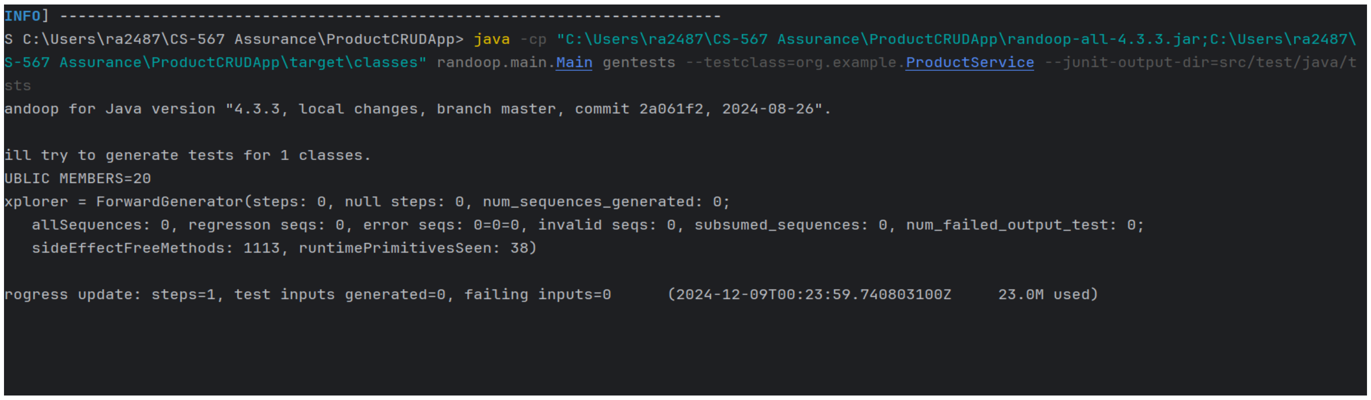
- Automated Bug Detection: It identifies bugs or edge cases that may not be anticipated by manual tests.

Command to generate Regression Tests Using Randoop:

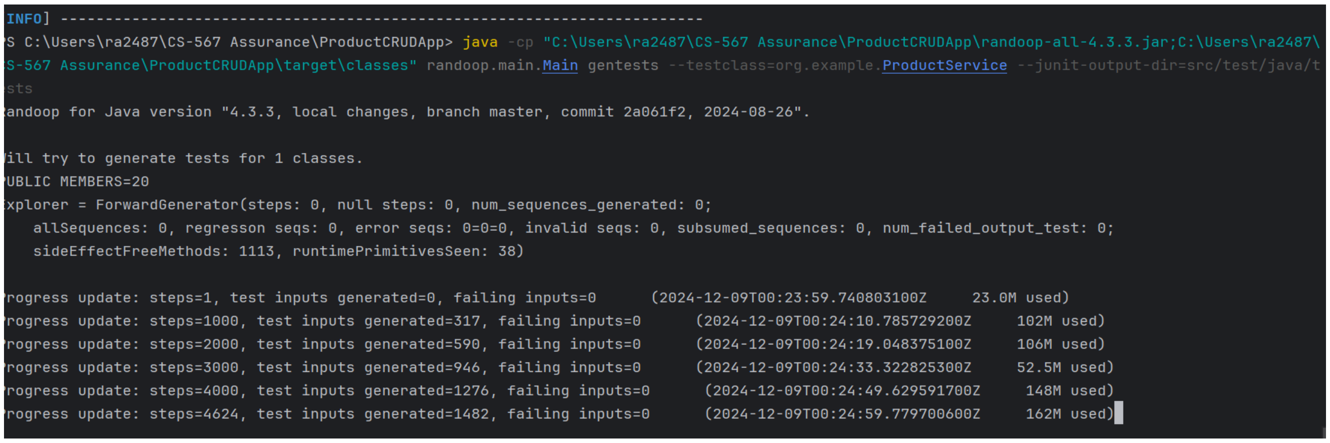


```
PS C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp> java -cp "C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\randoop-all-4.3.3.jar;C:\Users\ra2487\
CS-567 Assurance\ProductCRUDApp\target\classes" randoop.main.Main gentests --testclass=org.example.ProductService --junit-output-dir=src/test/java/t
ests
```

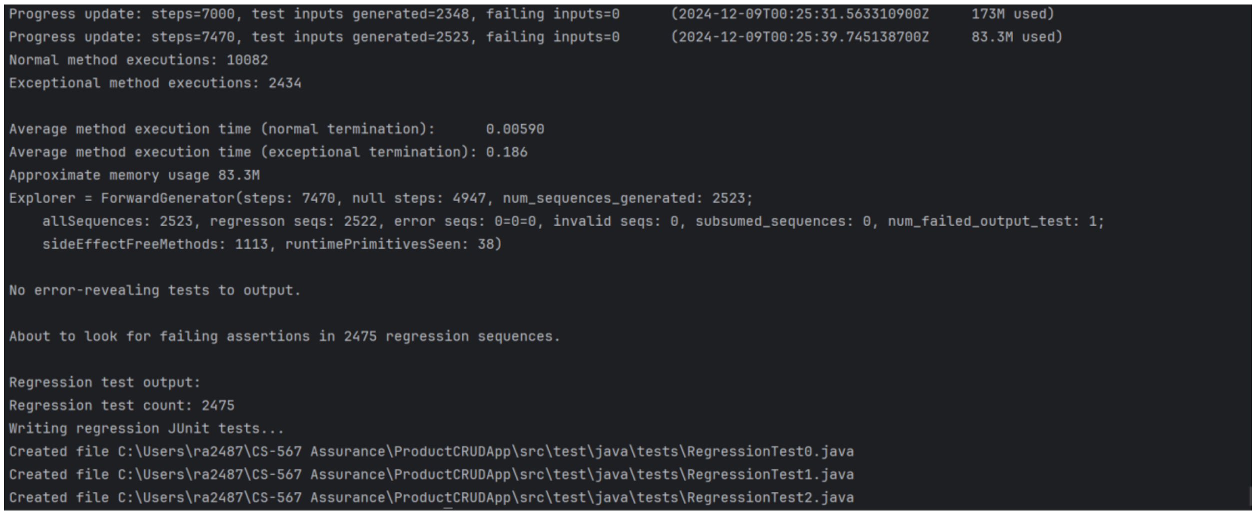
Running Screenshots of Randoop Tests:



```
INFO] ------------------------------------------------------------------------
S C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp> java -cp "C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\randoop-all-4.3.3.jar;C:\Users\ra2487\
S-567 Assurance\ProductCRUDApp\target\classes" randoop.main.Main gentests --testclass=org.example.ProductService --junit-output-dir=src/test/java/t
sts
andoop for Java version "4.3.3, local changes, branch master, commit 2a061f2, 2024-08-26".

ill try to generate tests for 1 classes.
UBLIC MEMBERS=20
xplorer = ForwardGenerator(steps: 0, null steps: 0, num_sequences_generated: 0;
   allSequences: 0, regresson seqs: 0, error seqs: 0=0=0, invalid seqs: 0, subsumed_sequences: 0, num_failed_output_test: 0;
   sideEffectFreeMethods: 1113, runtimePrimitivesSeen: 38)

rogress update: steps=1, test inputs generated=0, failing inputs=0      (2024-12-09T00:23:59.740803100Z     23.0M used)
```



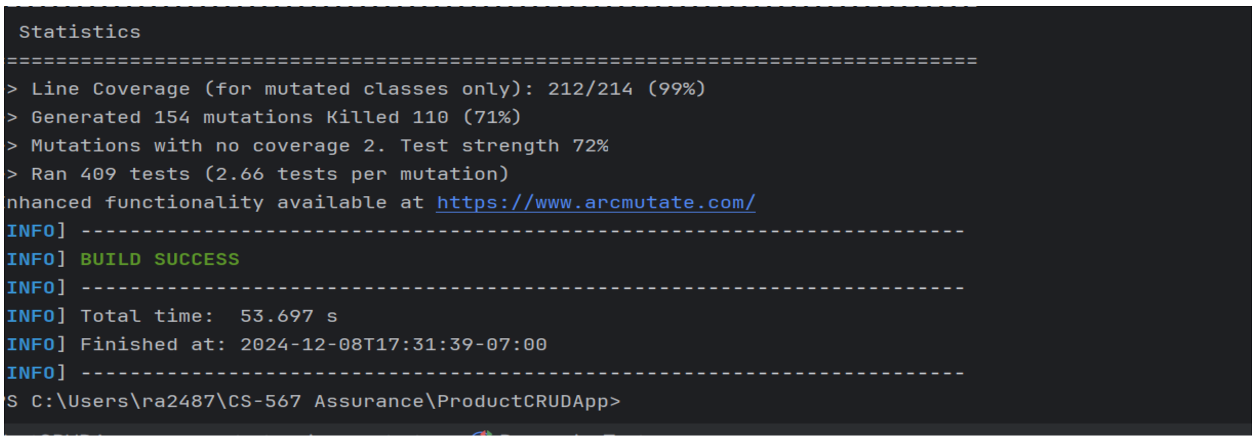
```
INFO] ------------------------------------------------------------------------
S C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp> java -cp "C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\randoop-all-4.3.3.jar;C:\Users\ra2487\
S-567 Assurance\ProductCRUDApp\target\classes" randoop.main.Main gentests --testclass=org.example.ProductService --junit-output-dir=src/test/java/t
sts
andoop for Java version "4.3.3, local changes, branch master, commit 2a061f2, 2024-08-26".

ill try to generate tests for 1 classes.
UBLIC MEMBERS=20
xplorer = ForwardGenerator(steps: 0, null steps: 0, num_sequences_generated: 0;
   allSequences: 0, regresson seqs: 0, error seqs: 0=0=0, invalid seqs: 0, subsumed_sequences: 0, num_failed_output_test: 0;
   sideEffectFreeMethods: 1113, runtimePrimitivesSeen: 38)

rogress update: steps=1, test inputs generated=0, failing inputs=0      (2024-12-09T00:23:59.740803100Z     23.0M used)
rogress update: steps=1000, test inputs generated=317, failing inputs=0      (2024-12-09T00:24:10.785729200Z     102M used)
rogress update: steps=2000, test inputs generated=590, failing inputs=0      (2024-12-09T00:24:19.048375100Z     106M used)
rogress update: steps=3000, test inputs generated=946, failing inputs=0      (2024-12-09T00:24:33.322825300Z     52.5M used)
rogress update: steps=4000, test inputs generated=1276, failing inputs=0      (2024-12-09T00:24:49.629591700Z     148M used)
rogress update: steps=4624, test inputs generated=1482, failing inputs=0      (2024-12-09T00:24:59.779700600Z     162M used)
```



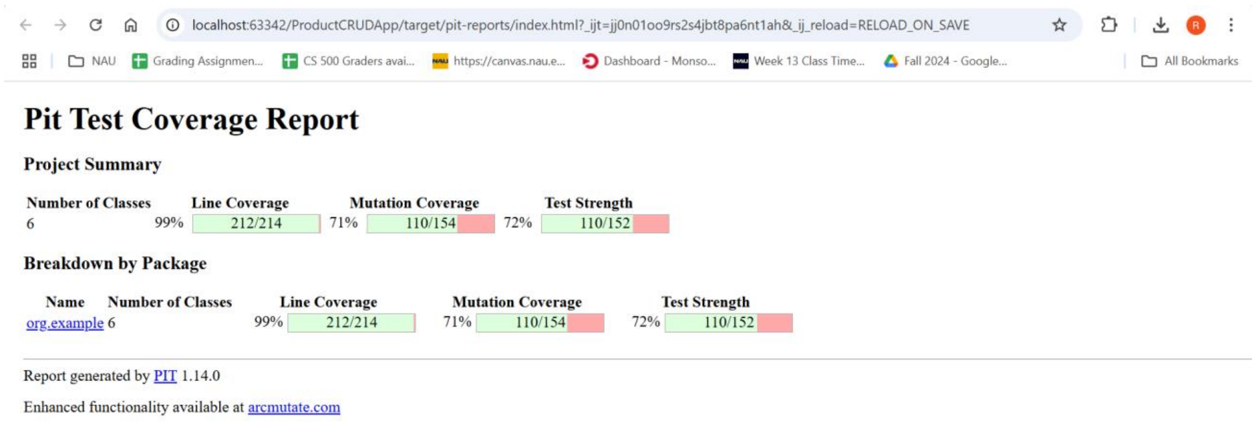
```
Progress update: steps=7000, test inputs generated=2348, failing inputs=0      (2024-12-09T00:25:31.563310900Z     173M used)
Progress update: steps=7470, test inputs generated=2523, failing inputs=0      (2024-12-09T00:25:39.745138700Z     83.3M used)
Normal method executions: 10082
Exceptional method executions: 2434

Average method execution time (normal termination):      0.00590
Average method execution time (exceptional termination): 0.186
Approximate memory usage 83.3M
Explorer = ForwardGenerator(steps: 7470, null steps: 4947, num_sequences_generated: 2523;
    allSequences: 2523, regresson seqs: 2522, error seqs: 0=0=0, invalid seqs: 0, subsumed_sequences: 0, num_failed_output_test: 1;
    sideEffectFreeMethods: 1113, runtimePrimitivesSeen: 38)

No error-revealing tests to output.

About to look for failing assertions in 2475 regression sequences.

Regression test output:
Regression test count: 2475
Writing regression JUnit tests...
Created file C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\src\test\java\tests\RegressionTest0.java
Created file C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\src\test\java\tests\RegressionTest1.java
Created file C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp\src\test\java\tests\RegressionTest2.java
```

So totally for this project ProductCrudApp it generated 14 Regression tests and also we can run again mutation test to check the score of mutation coverage and test strength.

PIT Report after adding Regression tests:



```
 Statistics
================================================================================
> Line Coverage (for mutated classes only): 212/214 (99%)
> Generated 154 mutations Killed 110 (71%)
> Mutations with no coverage 2. Test strength 72%
> Ran 409 tests (2.66 tests per mutation)
nhanced functionality available at https://www.arcmutate.com/
INFO] ------------------------------------------------------------------------
INFO] BUILD SUCCESS
INFO] ------------------------------------------------------------------------
INFO] Total time:  53.697 s
INFO] Finished at: 2024-12-08T17:31:39-07:00
INFO] ------------------------------------------------------------------------
S C:\Users\ra2487\CS-567 Assurance\ProductCRUDApp>
```



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
6	99%	212/214	71%	110/154	72%	110/152

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
org.example	6	99%	212/214	71%	110/154	72%	110/152

Report generated by PIT 1.14.0

Enhanced functionality available at arcmutate.com

Key Observations:

The testing process for the ProductCrudApp has shown substantial improvements after implementing automated regression tests. Initially, the mutation coverage was at 69% and test strength was at 70%. However, after incorporating regression tests, mutation coverage increased to 71% and test strength rose to 72%, demonstrating the positive impact of these tests in strengthening the codebase.

Randoop's automatic test generation and mutation testing have also contributed to enhancing the reliability and stability of the application. The tests now cover a broader range of edge cases, reducing the likelihood of bugs with future changes. These improvements validate the importance of continuous testing, with automated tools helping ensure consistent quality and performance throughout the project lifecycle.

In summary, regression testing has been pivotal in improving the overall quality metrics of the application, confirming that automated testing is essential for maintaining robust, error-free software.

Git Hub Url:

https://github.com/ARaghavender/assurance_project