Mutation Testing

Software Testing CS-731

Akanksha Shukla	**MT2022008**
Vikranth Bari	**MT2022131**

Click here to access code

Introduction

We have implemented mutation testing on a web application (Backend only). The endpoints are checked through **Postman** and integration and unit testing is done using **PIT test suite**. Mutation testing is a valuable technique to assess the effectiveness of your test suite by introducing small changes (mutations) to the source code and checking if the tests can detect these changes. In the context of Java and Spring Boot, PIT (Pitest) is a popular mutation testing framework. The project we have created is a *Blood Bank Management System,* which allows user to easily donate and request blood units from there nearby blood banks, this code contains around 300+ lines of code.

Here are the general steps to perform mutation testing using PIT and check endpoints using Postman:

- ✓ Add PIT plugin to your project
- ✓ Configure mutators
- ✓ Run PIT Mutation Testing
- ✓ View the reports
- ✓ Check endpoints using Postman

Add PIT plugin to your project:

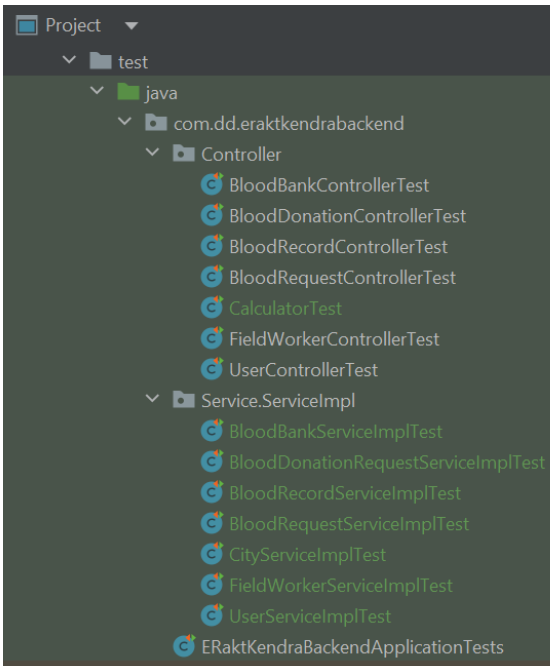
Add the PIT Maven plugin to your ***pom.xml*** file. Ensure that you specify the necessary configuration for your project. By default pitest will mutate all code in your project.

```xml
<plugin>
    <groupId>org.pitest</groupId>
    <artifactId>pitest-maven</artifactId>
    <version>LATEST</version>
</plugin>
```

You can limit which code is mutated and which tests are run using **targetClasses** and **targetTests**. If no *targetClasses* are provided in versions before 1.2.0, pitest assumes that your classes live in a package matching your projects group id. In 1.2.0 and later versions pitest will scan your project to determine which classes are present.

```xml
<plugin>
    <groupId>org.pitest</groupId>
    <artifactId>pitest-maven</artifactId>
    <version>LATEST</version>
    <configuration>
        <targetClasses>
            <param>com.your.package.root.want.to.mutate*</param>
        </targetClasses>
        <targetTests>
            <param>com.your.package.root*</param>
        </targetTests>
    </configuration>
</plugin>
```

Write Testcases for your project



```
Project
  test
    java
      com.dd.eraktkendrabackend
        Controller
          BloodBankControllerTest
          BloodDonationControllerTest
          BloodRecordControllerTest
          BloodRequestControllerTest
          CalculatorTest
          FieldWorkerControllerTest
          UserControllerTest
        Service.ServiceImpl
          BloodBankServiceImplTest
          BloodDonationRequestServiceImplTest
          BloodRecordServiceImplTest
          BloodRequestServiceImplTest
          CityServiceImplTest
          FieldWorkerServiceImplTest
          UserServiceImplTest
        ERaktKendraBackendApplicationTests
```

You have to write testcases for your classes which has some functionalities implemented, here I have created test cases for all the Controller classes and Service classes.

Configure Mutators:

PIT provides various mutators that you can configure. For example, we can configure PIT to use specific mutators or exclude certain ones. Here is the screenshot of our code of mutators.

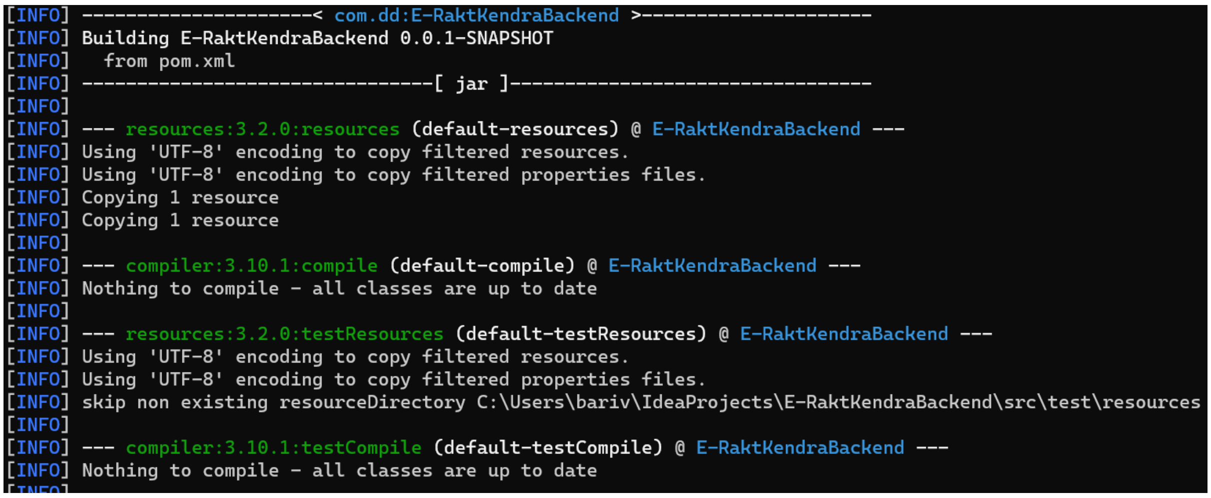
```
<plugin>
    <groupId>org.pitest</groupId>
    <artifactId>pitest-maven</artifactId>
    <version>1.15.3</version>
    <configuration>
        <targetClasses>
            <param>com.dd.eraktkendrabackend.Controller.*</param>

<param>com.dd.eraktkendrabackend.Service.ServiceImpl.*</param>
        </targetClasses>

        <mutators>
            <mutator>CONDITIONALS_BOUNDARY</mutator>
            <mutator>EMPTY_RETURNS</mutator>
            <mutator>FALSE_RETURNS</mutator>
            <mutator>INCREMENTS</mutator>
            <mutator>INVERT_NEGS</mutator>
            <mutator>MATH</mutator>
            <mutator>NEGATE_CONDITIONALS</mutator>
            <mutator>NULL_RETURNS</mutator>
            <mutator>PRIMITIVE_RETURNS</mutator>
            <mutator>TRUE_RETURNS</mutator>
            <mutator>VOID_METHOD_CALLS</mutator>

            <mutator>NON_VOID_METHOD_CALLS</mutator>
            <mutator>EXPERIMENTAL_ARGUMENT_PROPAGATION</mutator>
            <mutator>EXPERIMENTAL_NAKED_RECEIVER</mutator>
        </mutators>
    </configuration>
</plugin>
```

Run PIT runner

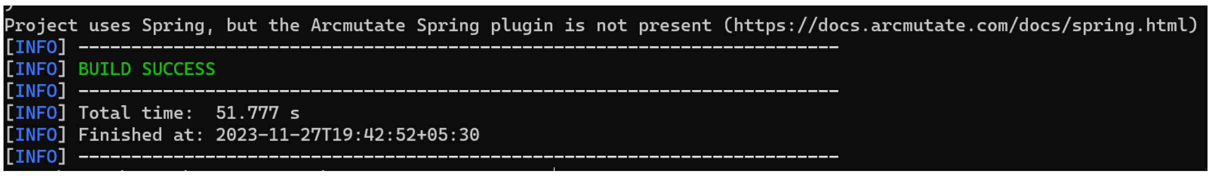
It can be run directly from the command-line:

```
mvn test-compile org.pitest:pitest-maven:mutationCoverage
```

this command will create mutants and run the mutation testing on the codebase, the resultant report will be generated in the target folder as an HTML document.



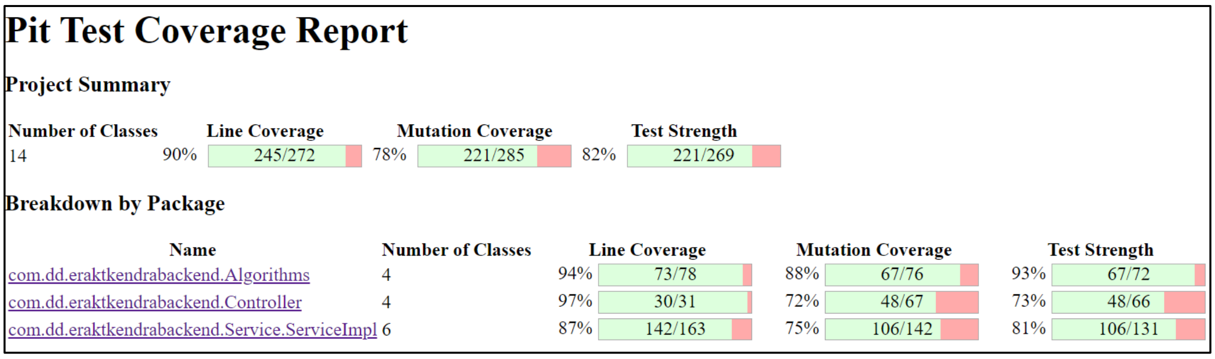
```
[INFO] --------------------< com.dd:E-RaktKendraBackend >--------------------
[INFO] Building E-RaktKendraBackend 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]--------------------------------
[INFO]
[INFO] --- resources:3.2.0:resources (default-resources) @ E-RaktKendraBackend ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 1 resource
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ E-RaktKendraBackend ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.2.0:testResources (default-testResources) @ E-RaktKendraBackend ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory C:\Users\bariv\IdeaProjects\E-RaktKendraBackend\src\test\resources
[INFO]
[INFO] --- compiler:3.10.1:testCompile (default-testCompile) @ E-RaktKendraBackend ---
[INFO] Nothing to compile - all classes are up to date
```



```
Project uses Spring, but the Arcmutate Spring plugin is not present (https://docs.arcmutate.com/docs/spring.html)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  51.777 s
[INFO] Finished at: 2023-11-27T19:42:52+05:30
[INFO] ------------------------------------------------------------------------
```

View the generated reports

After the execution is complete, we can find the HTML report in the *target/pit-reports directory*. Open the *index.html* file in a browser to view the detailed mutation testing report.



Pit Test Coverage Report

Project Summary

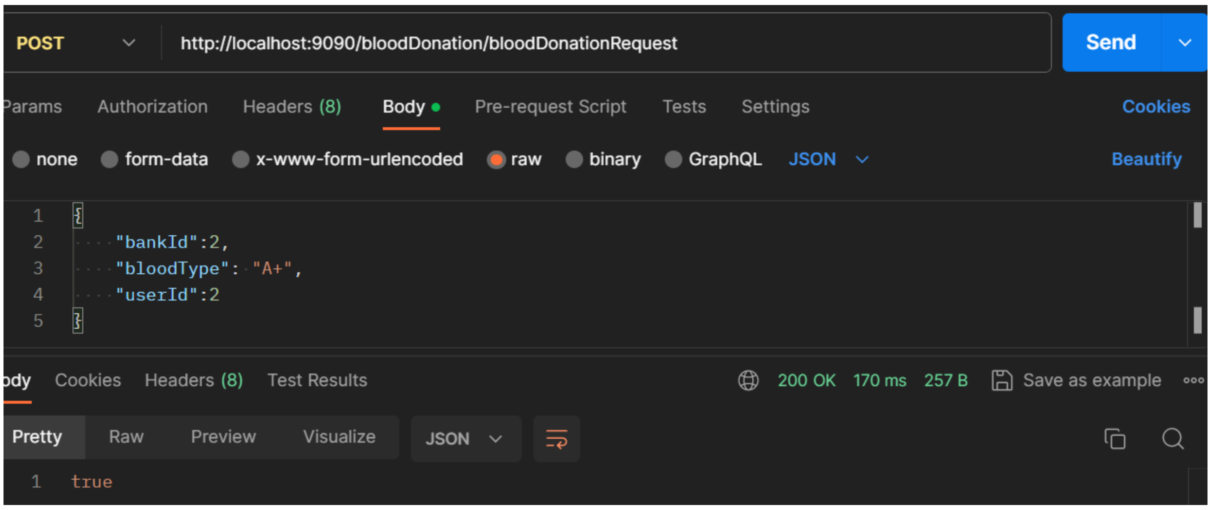
Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
14	90%	245/272	78%	221/285	82%	221/269

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
com.dd.eraktkendrabackend.Algorithms	4	94%	73/78	88%	67/76	93%	67/72
com.dd.eraktkendrabackend.Controller	4	97%	30/31	72%	48/67	73%	48/66
com.dd.eraktkendrabackend.Service.ServiceImpl	6	87%	142/163	75%	106/142	81%	106/131

Checking End-Points using Postman:

To check all the endpoints of our project, we have used postman, postman allows us to test all kinds of endpoints (POST, GET, DELETE, etc.). For example, this is an endpoint to register a blood donation request:



POST http://localhost:9090/bloodDonation/bloodDonationRequest

```
{
    "bankId":2,
    "bloodType": "A+",
    "userId":2
}
```

200 OK 170 ms 257 B

```
true
```

Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify
Body Cookies Headers (8) Test Results
Save as example
Pretty Raw Preview Visualize JSON

Here, we are using post mapping because this is a post request, along with it we are providing data to be registered in database in json format. The **200ok** status shows that the request is executed successfully. In the same way other APIs are also tested.