Assignment 3 - Mutation Testing

Task 1

We selected pitest to evaluate the mutation coverage of the project. From the report, we selected four classes, two from the HOA microservice and two from the voting microservice, in need of a more extensive mutation coverage:

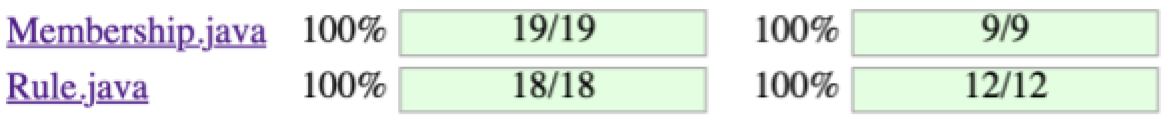
1. Membership
2. Rule
3. VotingInformationController
4. VotingInteractionController

We wrote new test cases to improve these classes' mutation scores.

1. HOA Microservice

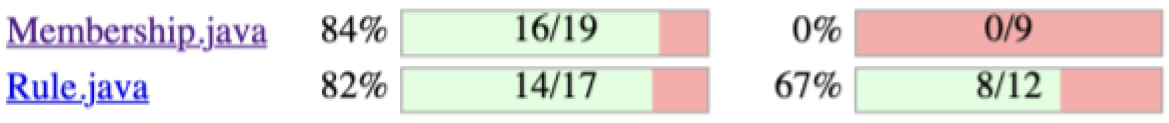
Link to the commit that makes the changes:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM22b/-/commit/82a64ba2c2cb89e59333ed712fe7425e524ffa95

Membership:
Original Score: 0%
New Score: 100%
Rule:
Original Score 67%
New Score: 100%



Membership.java	100%	19/19	100%	9/9
Rule.java	100%	18/18	100%	12/12

Fig 1.1.1 The Mutation Coverage after updating the test suites



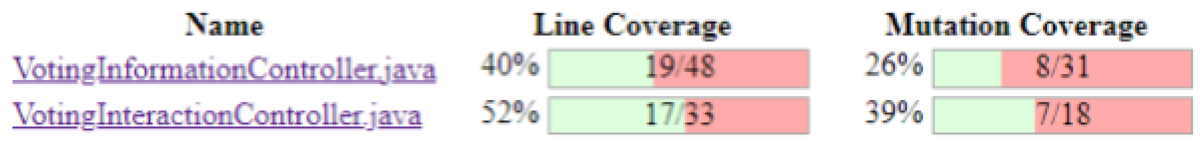
Membership.java	84%	16/19	0%	0/9
Rule.java	82%	14/17	67%	8/12

Fig 1.1.2 The Mutation Coverage before updating the test suites

2. Voting Microservice

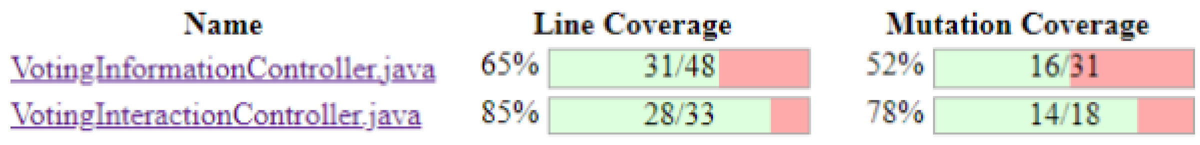
VotingInformationController:
Original score: 26%
New score: 52%

VotingInteractionController:
Original score: 39%
New score: 78%



Name	Line Coverage		Mutation Coverage	
VotingInformationController.java	40%	19/48	26%	8/31
VotingInteractionController.java	52%	17/33	39%	7/18

Fig.1.2.1 Original Report for the classes in the Voting Microservice



Name	Line Coverage		Mutation Coverage	
VotingInformationController.java	65%	31/48	52%	16/31
VotingInteractionController.java	85%	28/33	78%	14/18

Fig.1.2.2 New Report for the classes in the Voting Microservice

Both classes were lacking some core tests for certain methods, such as checking the behaviour of null inputs. The pitest report helped single out those problems and improve the overall quality of the test suite.

Task 2

For this task, the core domain we chose is the HOA, because it's the most crucial domain for the functionality of our application. We selected 4 classes to create manual mutation, namely:

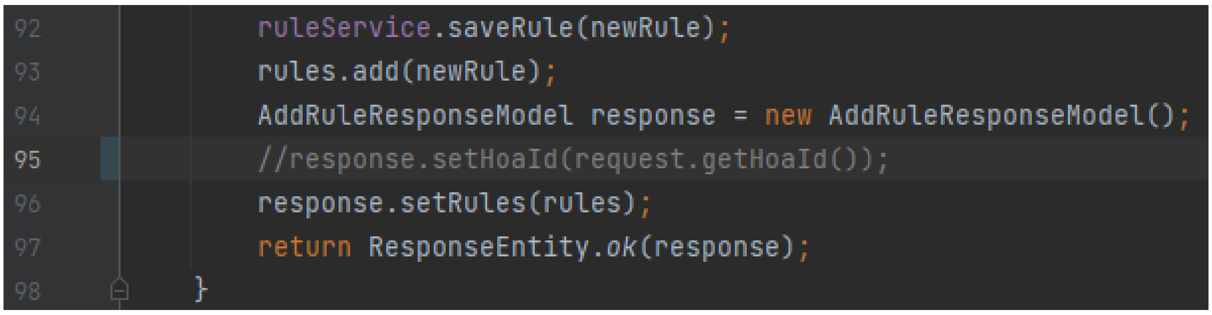
1. RuleController
2. HoaService
3. HoaController
4. Membership Management Service

1. RuleController

Link to the commit, that adds the changes to the tests:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM22b/-/commit/d529788b7d2ed5c0c7aec8f20b36f83c09ed648f

We decided to plant a mutation in the endpoint for adding a rule to an Hoa. We chose this class, because it is critical for our application - without it, we would not have satisfied one of the requirements, given from the client.

The mutant I injected was commenting out the line that sets one of the parameters of the response, as shown below:



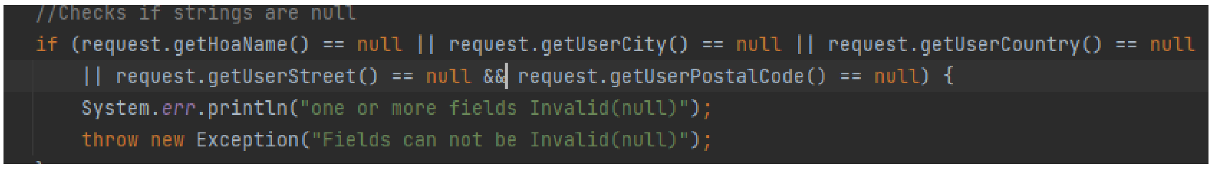
```java
        ruleService.saveRule(newRule);
        rules.add(newRule);
        AddRuleResponseModel response = new AddRuleResponseModel();
        //response.setHoaId(request.getHoaId());
        response.setRules(rules);
        return ResponseEntity.ok(response);
    }
```

At first the test for this method passed. So, in order to kill the mutant, we added another assert in the test, that ensures, that the parameter is set properly

2. HoaService

Link to the commit, that adds the changes to the tests:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM22b/-/commit/10e7b893ed109077cfaa829d464bcbbb9dedd034

Providing valid information is extremely critical for the proper functionality of our application. Because of that, we decided to create a mutant in the HoaService class, in a method that checks whether the provided parameters for creating an Hoa are valid. We changed an or-operator (| |) to an and-operator (&&), as shown in the picture below, and the test was still passing.



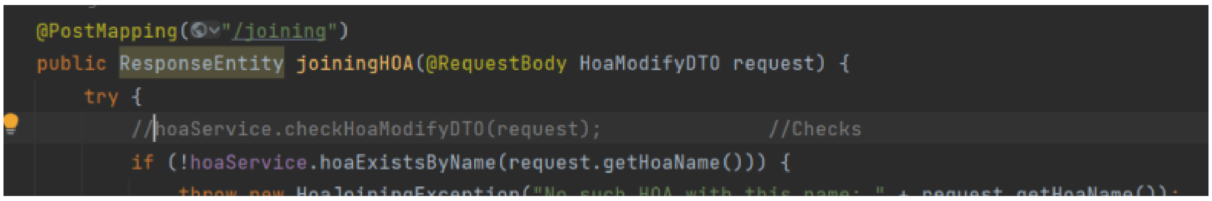
```java
//Checks if strings are null
if (request.getHoaName() == null || request.getUserCity() == null || request.getUserCountry() == null
    || request.getUserStreet() == null && request.getUserPostalCode() == null) {
    System.err.println("one or more fields Invalid(null)");
    throw new Exception("Fields can not be Invalid(null)");
```

In order to kill the mutant, we added more tests that check for each of the parameters (null check and empty check). If we now run the tests and one of the parameters is invalid, there will be at least 1 test that is going to fail.

3. HoaController

Link to commit:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM22b/-/commit/86942e52fe07d5988e7a74d4d9c48293793369fb

It is critical for our application to check if the parameters provided are valid. For that reason it is important that a mutant concerning parameter checks does not survive, this is why when we saw that a mutant linked to removing the checks from one of our core methods was surviving we quickly fixed this oversight. If a developer were to make the mistake of removing this check and forgetting about it we would have no way of knowing and invalid parameters for the addresses of HOA members would have been able to get through our system. It is crucial that we prevent this from happening.



```
@PostMapping("/joining")
public ResponseEntity joiningHOA(@RequestBody HoaModifyDTO request) {
    try {
        //hoaService.checkHoaModifyDTO(request);              //Checks
        if (!hoaService.hoaExistsByName(request.getHoaName())) {
```

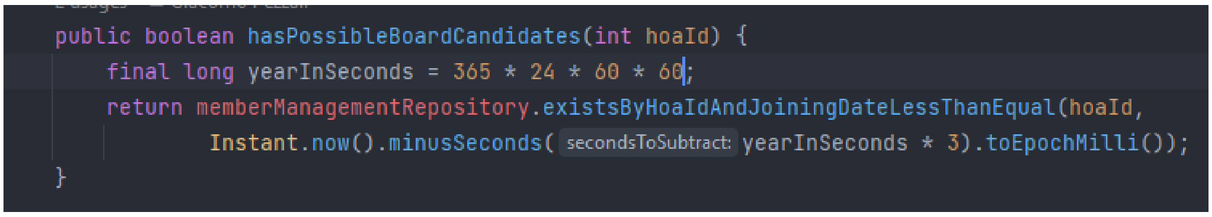
To kill this mutant we added a test which validates that checks are indeed ran for this method, thus putting an end to it. The test successfully failed if checks were not being done as needed.

4. MembershipManagementService

Link to commit:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM22b/-/commit/37007f6bb3584b88b87abab6681d5cdc7f848630

We identified this class to be critical since it contains important eligibility checks and information retrievals of the memberships. Most of the functionalities related to memberships such as adding/modifying members or finding information board information must go through this service at some point.

One of the important checks is to find whether a HOA has a possible board candidate (members who have joined the HOA for at least 3 years). In the method hasPossibleBoardCandidates, we simply multiply a few numbers to convert 3 years into milliseconds. A mutant is introduced here by changing 60 to 59 in the calculation.



```
public boolean hasPossibleBoardCandidates(int hoaId) {
    final long yearInSeconds = 365 * 24 * 60 * 60;
    return memberManagementRepository.existsByHoaIdAndJoiningDateLessThanEqual(hoaId,
            Instant.now().minusSeconds( secondsToSubtract: yearInSeconds * 3).toEpochMilli());
}
```

To kill the mutant, we added a test that checks whether a member who has joined a HOA for 3 years minus 1 day can still be considered a possible candidate.