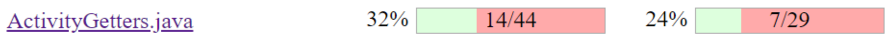
Task 1: Automated Mutation Testing

1. `ActivityGetters (Activity microservice)`

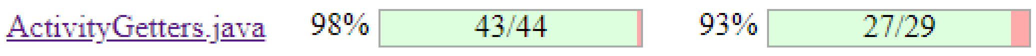
Previous coverage:



ActivityGetters.java	32%	14/44	24%	7/29

The ActivityGetters class handles the main logic for finding suitable activities for a user's preferences. Since the refactoring, the tests for this class were not properly updated, hence the line coverage (as seen on the left of the picture) and the mutation coverage (second box) were both low. More tests were added, to check all the expected interactions this class should have with the repository, since part of the method's functionalities were translated into SQL. It was sufficient to add 4 tests: one that ensures that all the owner's activities are retrieved (regardless of preferences), one that ensures that activities retrieved match the provided date availabilities (with regards to testing edge cases as well), one test that ensures that if a competition is retrieved additional checks are done on the organization and on the gender, and one that also checks for the certificate level to match when we are looking for the COX role. Designing the tests such that all edge cases are covered ensured not only an increase in mutation coverage (over 60% increase), but also in branch coverage.

Current coverage:

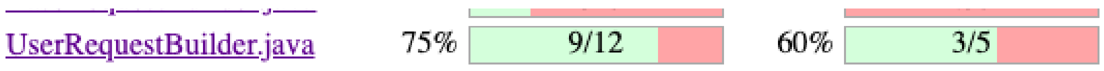


ActivityGetters.java	98%	43/44	93%	27/29

Commit:
The commit where you can find the changes made:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM30a/-/commit/156b2e1ae8ad7b871c0d2a6cf99950839e1ff67b?merge_request_iid=55

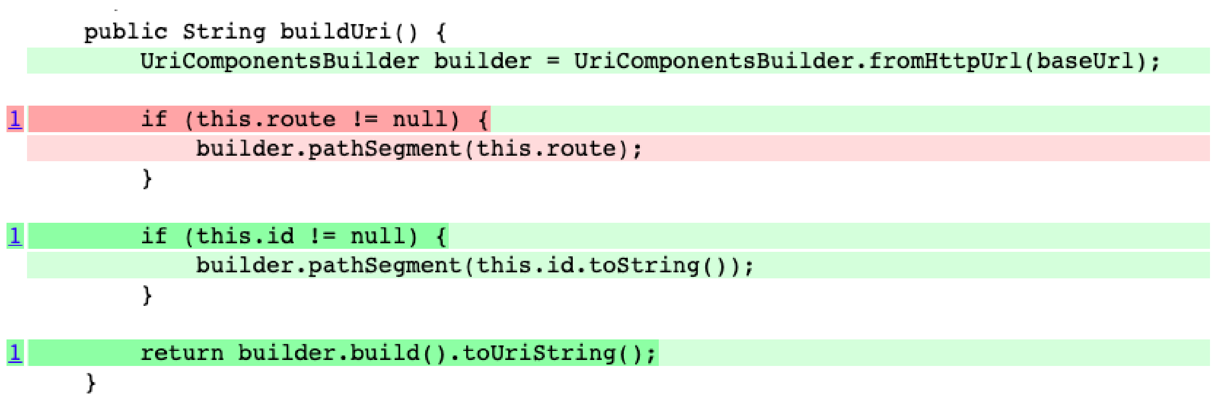
2. `UserRequestBuilder (Planning microservice)`

Previous coverage:



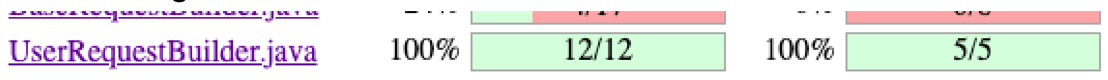
UserRequestBuilder.java	75%	9/12	60%	3/5

For the UserRequestBuilder class inside the Planning Microservice, Pitest revealed the fact that the current tests were only checking the id of the request, and not the route as well.


```
    public String buildUri() {
        UriComponentsBuilder builder = UriComponentsBuilder.fromHttpUrl(baseUrl);

1       if (this.route != null) {
            builder.pathSegment(this.route);
        }

1       if (this.id != null) {
            builder.pathSegment(this.id.toString());
        }

1       return builder.build().toUriString();
    }
```

Therefore, two tests were sufficient for killing all of the mutants: one that checks that the UserRequestBuilder correctly sets the route, and one which checks that setting both of these fields results in a correct Uri.

Current coverage:

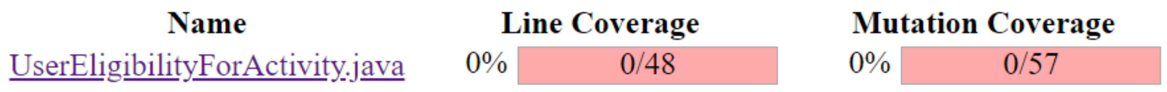


UserRequestBuilder.java	100%	12/12	100%	5/5

Commit:
These tests can be seen in the following commit:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM30a/-/commit/d75d44fe498a7f1a92385b1a806f928d85b8044d

3. `UserEligibilityForActivity (Planning microservice)`

Previous coverage:



Name	Line Coverage		Mutation Coverage	
UserEligibilityForActivity.java	0%	0/48	0%	0/57

The refactoring of the Planning Microservice introduced a number of improvements, one of which was the UserEligibilityForActivity class. This class plays a critical role in the overall functionality of the application and it is essential that it is thoroughly tested. Unfortunately, during the refactoring process, tests for this class were omitted, which led to a lack of mutation coverage. In order to improve the coverage and ensure the class is functioning correctly, a significant amount of testing was conducted.

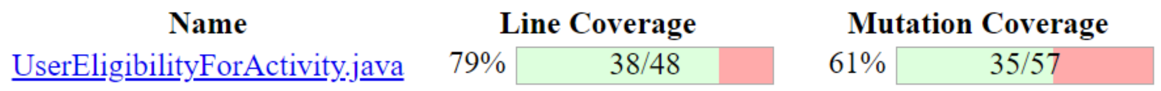
The goal of the testing was to increase the mutation coverage by at least 60%. To achieve this, 20 tests were added to the UserEligibilityForActivityTest class (nl.tudelft.sem.template.planning.domain.service). These tests focused on examining the helper methods within the UserEligibilityForActivity class, such as checkGenderOrganization, checkRole, checkTime, checkCertificate, checkLevel and checkSpotAvailability.

For example, to check the correctness of the checkGenderOrganization method, four different scenarios were investigated. These scenarios included: a) a correct gender and organization, b) a correct gender but the wrong organization, c) a wrong gender but the correct organization, and d) a wrong gender and organization. Through these tests, it was discovered that only by introducing these tests, the coverage increased by 7%.

Similarly, to examine the accuracy of the checkRole method, two scenarios were created. These scenarios examined whether the activity required a role that the user wanted to apply for, and whether it did not require it. Using this pattern, tests were also created for the checkTime, checkCertificate, checkLevel and checkSpotAvailability methods.

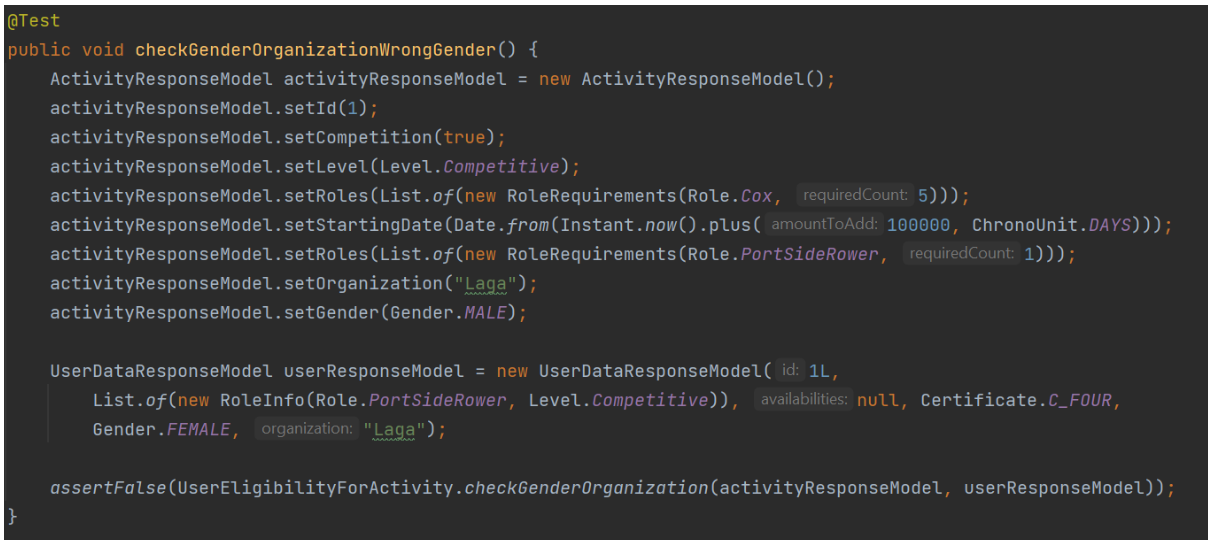
Overall, the testing conducted on the UserEligibilityForActivity class was thorough and comprehensive, with a focus on increasing mutation coverage by at least 60%. The tests were designed to examine various scenarios, including different combinations of correct and incorrect inputs, to ensure that the helper methods within the class were functioning correctly. This was an important step in ensuring the overall reliability and robustness of the Planning Microservice.

Current coverage:

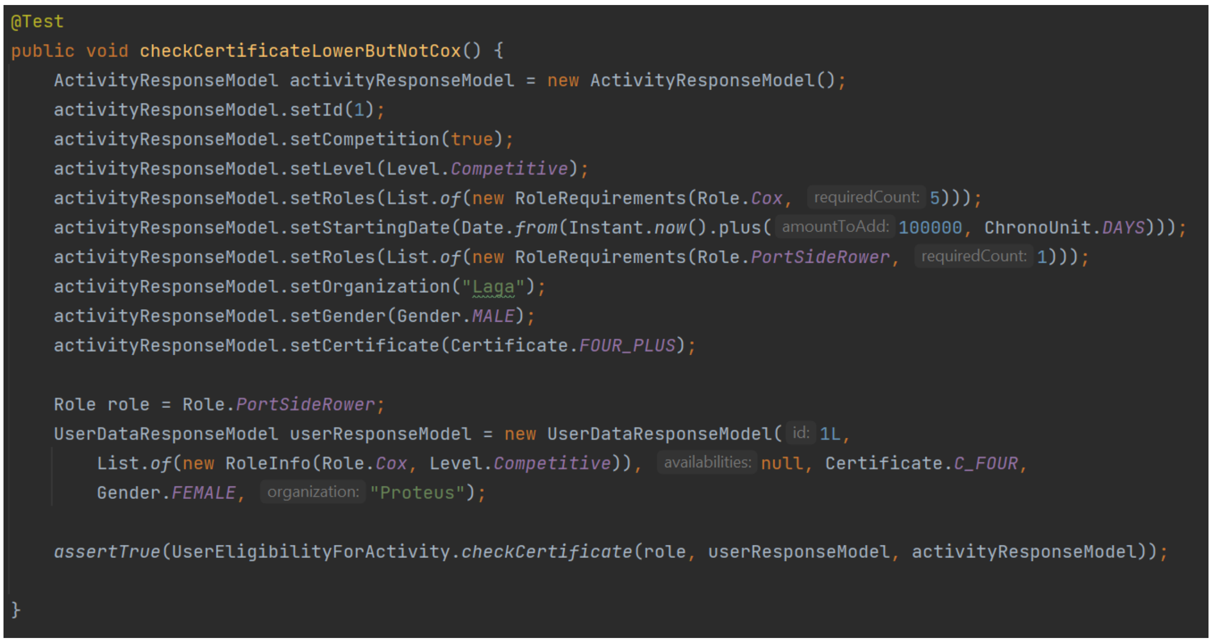


Name	Line Coverage		Mutation Coverage	
UserEligibilityForActivity.java	79%	38/48	61%	35/57

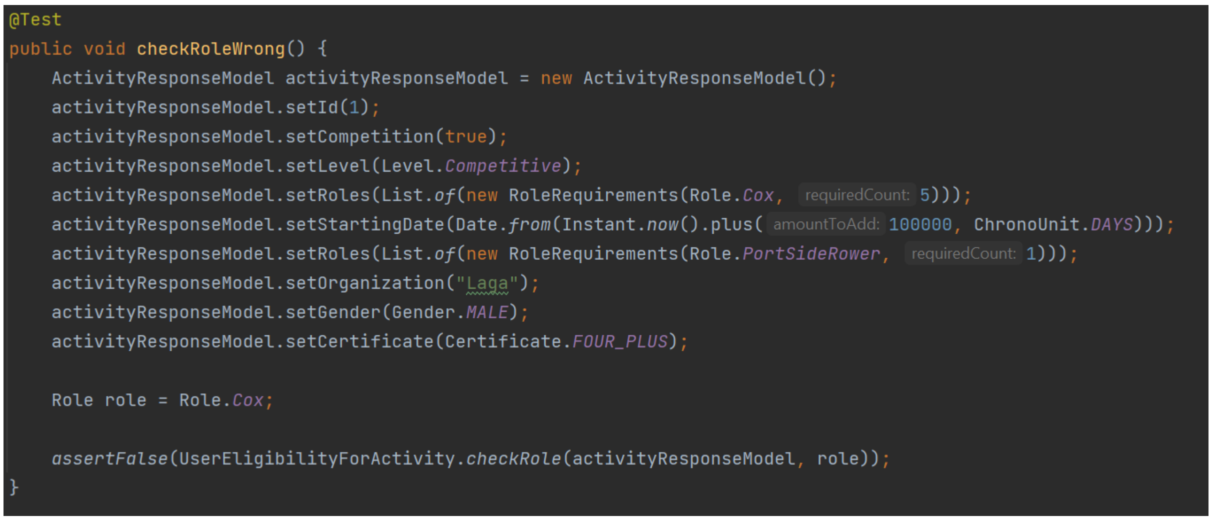
Examples of created tests:



```
@Test
public void checkGenderOrganizationWrongGender() {
    ActivityResponseModel activityResponseModel = new ActivityResponseModel();
    activityResponseModel.setId(1);
    activityResponseModel.setCompetition(true);
    activityResponseModel.setLevel(Level.Competitive);
    activityResponseModel.setRoles(List.of(new RoleRequirements(Role.Cox,  requiredCount: 5)));
    activityResponseModel.setStartingDate(Date.from(Instant.now().plus( amountToAdd: 100000, ChronoUnit.DAYS)));
    activityResponseModel.setRoles(List.of(new RoleRequirements(Role.PortSideRower,  requiredCount: 1)));
    activityResponseModel.setOrganization("Laga");
    activityResponseModel.setGender(Gender.MALE);

    UserDataResponseModel userResponseModel = new UserDataResponseModel( id: 1L,
        List.of(new RoleInfo(Role.PortSideRower, Level.Competitive)),  availabilities: null, Certificate.C_FOUR,
        Gender.FEMALE,  organization: "Laga");

    assertFalse(UserEligibilityForActivity.checkGenderOrganization(activityResponseModel, userResponseModel));
}
```



```java
@Test
public void checkCertificateLowerButNotCox() {
    ActivityResponseModel activityResponseModel = new ActivityResponseModel();
    activityResponseModel.setId(1);
    activityResponseModel.setCompetition(true);
    activityResponseModel.setLevel(Level.Competitive);
    activityResponseModel.setRoles(List.of(new RoleRequirements(Role.Cox,  requiredCount: 5)));
    activityResponseModel.setStartingDate(Date.from(Instant.now().plus( amountToAdd: 100000, ChronoUnit.DAYS)));
    activityResponseModel.setRoles(List.of(new RoleRequirements(Role.PortSideRower,  requiredCount: 1)));
    activityResponseModel.setOrganization("Laga");
    activityResponseModel.setGender(Gender.MALE);
    activityResponseModel.setCertificate(Certificate.FOUR_PLUS);

    Role role = Role.PortSideRower;
    UserDataResponseModel userResponseModel = new UserDataResponseModel( id: 1L,
        List.of(new RoleInfo(Role.Cox, Level.Competitive)),  availabilities: null, Certificate.C_FOUR,
        Gender.FEMALE,  organization: "Proteus");

    assertTrue(UserEligibilityForActivity.checkCertificate(role, userResponseModel, activityResponseModel));

}
```

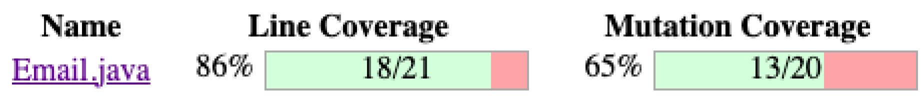


```java
@Test
public void checkRoleWrong() {
    ActivityResponseModel activityResponseModel = new ActivityResponseModel();
    activityResponseModel.setId(1);
    activityResponseModel.setCompetition(true);
    activityResponseModel.setLevel(Level.Competitive);
    activityResponseModel.setRoles(List.of(new RoleRequirements(Role.Cox,  requiredCount: 5)));
    activityResponseModel.setStartingDate(Date.from(Instant.now().plus( amountToAdd: 100000, ChronoUnit.DAYS)));
    activityResponseModel.setRoles(List.of(new RoleRequirements(Role.PortSideRower,  requiredCount: 1)));
    activityResponseModel.setOrganization("Laga");
    activityResponseModel.setGender(Gender.MALE);
    activityResponseModel.setCertificate(Certificate.FOUR_PLUS);

    Role role = Role.Cox;

    assertFalse(UserEligibilityForActivity.checkRole(activityResponseModel, role));
}
```

Commit:
Previously mentioned test cases were added in commit:
Improved mutation coverage in UserEligibilityForActivity by 61% (675fa9ca) · Commits · CSE2115 - Software Engineering Methods / 2022-2023 / SEM-30a · GitLab (tudelft.nl)

4. `Email (Messaging microservice)`

Previous coverage:



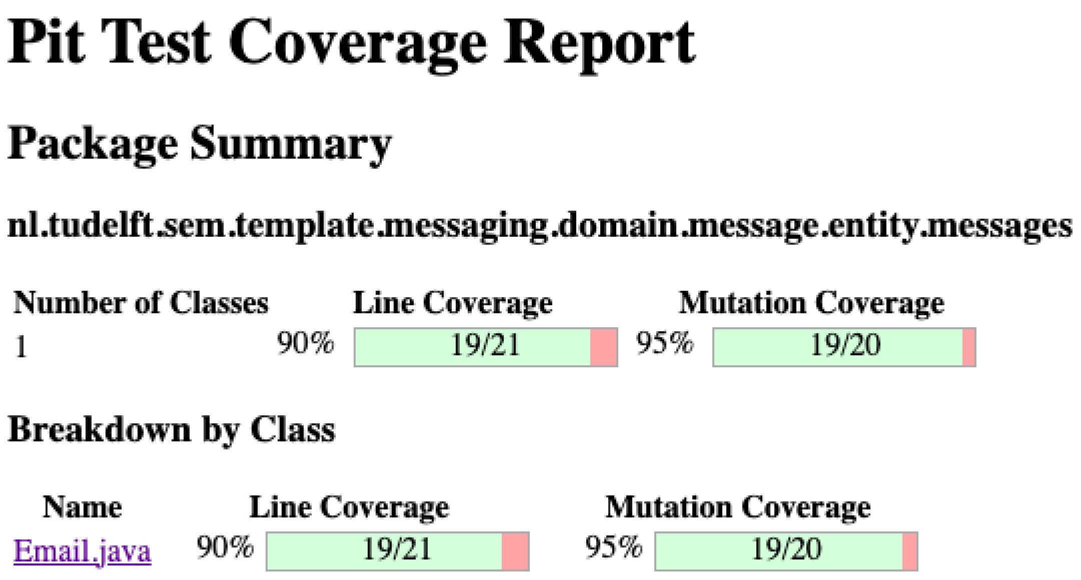
Name	Line Coverage		Mutation Coverage	
Email.java	86%	18/21	65%	13/20

Pitest revealed that the original tests failed to check that:

1. Emails can only be sent to non-null email addresses;

2. When sent, emails have the correct values for the sender, recipient, subject, message fields.

Resulting coverage:



Pit Test Coverage Report

Package Summary

nl.tudelft.sem.template.messaging.domain.message.entity.messages

Number of Classes	Line Coverage		Mutation Coverage	
1	90%	19/21	95%	19/20

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Email.java	90%	19/21	95%	19/20

Commit:
Therefore, two tests were created for these cases. They can be seen in the following commit:
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM30a/-/commit/f93e522452418bee52f0171f95f0430e02a7c288