HOA 23b Assignment 3

1.

For this assignment we have to improve our test suite for our microservices. To do this we have used mutation testing. Mutation testing is where we slightly alter the code, and expect the test suite to fail. This way we can check if the test suite actually tests something meaningful.

For this assignment we have used Pitest as our mutation testing tool. We have identified 4 classes that we can improve the mutation test coverage of.

1. ActivitiesController from Association-microservice
    - 0% → 100% mutation coverage
    - https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM23b/-/merge_requests/76
2. VotingController from Voting-microservice
    - 61% → 96% mutation coverage
    - https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM23b/-/merge_requests/76
3. CouncilController from Association-microservice
    - 42% → 100% mutation coverage
    - https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM23b/-/merge_requests/76
4. AppUser from Authentication-microservice
    - 17% → 75% mutation coverage
    - https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM23b/-/merge_requests/76

2.1

For the manual mutation testing, we chose the AssociationService, MembershipService, HistoryService, ReportService, and Activity classes. Most of the logic regarding association, such as creating, joining, and leaving the association, storing its history, notifications, and reporting resides in the Service classes. Meanwhile, the Activity class contains methods that edit the Activity objects from the Notice Board. These functionalities are essential to the application, therefore all of these classes are critical.

2.2 + 2.3

AssociationService.leaveAssociation(): *(queries and updates membershipRepository)*

- *We consider the method critical because it's important for users to leave associations as they may move to a different city or neighborhood.*
- I chose to create a mutant by changing an OR to an AND. This way, the code would still throw an exception when both of the selected association and membership did not exist, but not if just one of them did not exist. I injected the mutant and it survived (commit).
- One such case where only one of the selected association and membership does not exist, is when the association exists, but the user is not a member. I wrote a test for such an occasion, and one more for a case where the function leaveAssociation() executes successfully. The tests killed the mutant (commit).
- Lastly, I changed the code back to the original (removed the mutant) and checked that all tests passed (commit).

MembershipService.displayNotifications(): *(queries and updates membershipRepository)*

- *We consider the method critical because, without it, the users have no access to the notification system.*
- The mutant in this method was created by changing the setter of the boolean, indicating a notification is read after displaying, from “true” to “false”. The code will keep running as usual and one can still display all notifications. However when someone displays the notifications and dismisses after, there will be no notifications to dismiss. This is due to the fact that all notifications are now marked unread after each display. The mutant was
- injected and survived (commit).
- I wrote a test addressing the previous situation where first the notifications are displayed and afterwards they are dismissed. The test then asserts whether the notifications were actually deleted or not. This killed the mutant (commit).
- Finally, I reverted the mutant injection and confirmed the new test suite still ran as expected (commit).

ReportService.addReport(): *(queries membershipRepository and updates reportRepository)*

- *We consider the method critical because, without it, there is no report system.*
- For ReportService, I created a mutant in the addReport method, by negating the two arguments in checking the validity of the reporter and violator. Now, if neither of the two people exist, it won’t throw an exception. I injected the mutant and it survived (commit).
- Two new integration tests are added to the test suite that check if both people do/don’t exist in the association. These two tests successfully killed the mutant (commit).
- I changed the code back to the original without the mutant, and checked that all tests pass correctly (commit).

HistoryService.addEvent(): *(queries and updates historyRepository)*

- *We consider the method critical because, without it, the history system is pointless as there would be no way to add new entries.*
- I would like to emphasize the importance of testing this method or it having been tested while we were working initially on the project. Because there were not enough tests to cover it, we did not catch a bug when starting an association which resulted in the association’s history not being initialized and saved. That led to quite a few problems which would’ve been easily avoided.
- I added a mutant in the method by making the if condition always true (adding an or statement with the negation of the first condition). After injection, the mutant survived (commit).
- The fix to the mutant lied modifying a previously made test which was supposed to check for said relationship. The problem was that the test was checking if the exception message was not empty but the exception which was sent was from optionalHistory.get() instead. Checking for the exact exception message resulted in the test killing the mutant (commit).
- After reverting the code to its original state I noticed that the code kept failing because of a null-related problem so I added an additional condition for the optionalHistory to not be null. Then, all tests passed. (commit)

Activity.addParticipating(): *(queries and updates activityRepository)*

- *This method is critical because, without it, users aren't able to express participation in activities, making the social aspect of them a bit redundant.*
- This method allows users to mark themselves as participating in an activity. When that happens, they also need to be removed from the list of interested users, if they were a part of it. During development we changed the GoingTo attribute to Participating. Because of this change, some confusion arose and while testing this method, we forgot to fully test if the user was also removed from the other list. This led to the mutant I found which consists of removing the call to removeInterested(). All tests passed after doing so. (commit)
- To deal with the mutant, a test was made to check for the contests of both Interested and Participating. This way, we can be sure that users are removed from their previous list when being added to a new one. There are several asserts to check the contents of the lists after multiple uses of the method. (commit)
- After reverting the changes the test suite passed. While a small thing, without the mutant testing this small oversight might have been left like that in the code so this is certainly proven to be a very useful tool (commit).