

Assignment 3

Task 1	1
Exercise 1	1
Exercise 2	1
Exercise 3 & 4	2
Task 2	6
Class I - PendingRequestScheduler	6
Class II - FacultyHandlerService	7
Class III- FacultyHandler	7
Class IV - ScheduleRequestController	8

Task 1

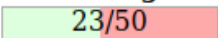
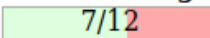
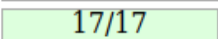
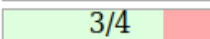
Exercise 1

For this exercise we have decided to use *Pitest* as the mutation testing tool. It generates *html* reports that we can use to access the mutation scores.

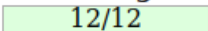
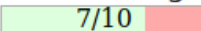
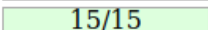
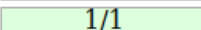
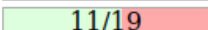
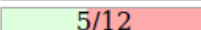
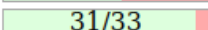
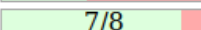
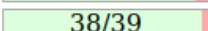
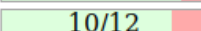
Exercise 2

After running *Pitest* we have found that the following methods have mutation score below 70%:

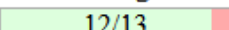
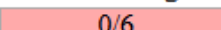
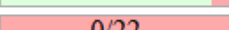
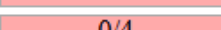
- a. `resource-manager-microservice/src/main/java/nl.tudelft.sem.resource.manager/controllers/NodeClusterController.java`

Name	Line Coverage	Mutation Coverage
NodeClusterController.java	46% 	58% 
ScheduleRequestController.java	100% 	75% 

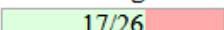
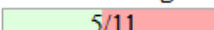
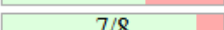
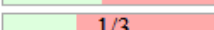
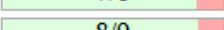
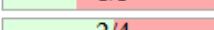
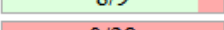
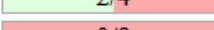
- b. `resource-manager-microservice/src/main/java/nl.tudelft.sem.resource.manager/domain/services/Manager.java`

Name	Line Coverage	Mutation Coverage
DateSchedulingService.java	100% 	70% 
FreepoolManager.java	100% 	100% 
Manager.java	58% 	42% 
ResourceAvailabilityService.java	94% 	88% 
ResourceHandler.java	97% 	83% 

c. *faculty-microservice/src/main/java/sem/faculty/controllers/MainFacultyController.java*

Name	Line Coverage	Mutation Coverage
MainFacultyController.java	92% 	0% 
ScheduleRequestController.java	0% 	0% 

d. *faculty-microservice/src/main/java/sem/faculty/domain/scheduler/AcceptRequestsScheduler.java*

Name	Line Coverage	Mutation Coverage
AcceptRequestsScheduler.java	65% 	45% 
DenyRequestsScheduler.java	88% 	33% 
PendingRequestsScheduler.java	89% 	50% 
SchedulableRequestsScheduler.java	0% 	0% 

Exercise 3 & 4

- I. From the report generated by *Pitest* regarding mutation score for *NodeClusterController.java*, we can see that this score of 46% is partly due to the non-covered lines, but also due to some surviving mutants.

Mutations

```

1. removed call to java/util/List::forEach → KILLED
1. removed call to sem/commons/Resource::setCpu → KILLED
1. removed call to sem/commons/Resource::setGpu → KILLED
1. removed call to sem/commons/Resource::setMemory → KILLED
1. replaced return value with null for nl/tudelft/sem/resource/manager/controllers/NodeClusterController::getUserViewResourcesForDate → KILLED
1. removed call to java/util/EnumSet::forEach → NO_COVERAGE
1. replaced return value with null for nl/tudelft/sem/resource/manager/controllers/NodeClusterController::lambda$getSysadminViewResourcesForDate$2 → NO_COVERAGE
1. replaced return value with null for nl/tudelft/sem/resource/manager/controllers/NodeClusterController::getSysadminViewResourcesForDate → NO_COVERAGE
1. replaced return value with "" for nl/tudelft/sem/resource/manager/controllers/NodeClusterController::removeClusterNode → NO_COVERAGE
1. removed call to nl/tudelft/sem/resource/manager/domain/services/Manager::removeNodeFromCluster → SURVIVED
1. replaced return value with "" for nl/tudelft/sem/resource/manager/controllers/NodeClusterController::removeClusterNode → NO_COVERAGE
1. replaced return value with "" for nl/tudelft/sem/resource/manager/controllers/NodeClusterController::removeClusterNode → KILLED

```

With this information, we added some methods to cover the lines that appeared uncovered, and added some tests to kill the covered but surviving mutants.

In this class, there was a method that proved rather difficult to test. Therefore, we have extracted some of its functionality into *Manager.java* (we tested that functionality as well) and simply called *Manager.java* here in

NodeClusterController.java. This method was *SysadminViewResources*. After testing the new version of this method, and making some of the other tests kill some surviving mutants (mainly due to lack of assertions on the output of the method), we have brought our mutation score to 90%

Breakdown by Class

Name	Line Coverage	Mutation Coverage
NodeClusterController.java	90% 36/40	100% 11/11
ScheduleRequestController.java	100% 17/17	75% 3/4

Here you can see the mutant list:

Mutations

```

48 1. removed call to java/util/List::forEach → KILLED
54 1. removed call to sem/commons/Resource::setCpu → KILLED
55 1. removed call to sem/commons/Resource::setGpu → KILLED
56 1. removed call to sem/commons/Resource::setMemory → KILLED
63 1. replaced return value with null for nl/tudelft/sem/resource/manager/controllers
   /NodeClusterController::getUserViewResourcesForDate → KILLED
93 1. replaced return value with null for nl/tudelft/sem/resource/manager/controllers
   /NodeClusterController::lambda$getSysadminViewResourcesForDate$1 → KILLED
100 1. replaced return value with null for nl/tudelft/sem/resource/manager/controllers
    /NodeClusterController::getSysadminViewResourcesForDate → KILLED
118 1. replaced return value with "" for nl/tudelft/sem/resource/manager/controllers/NodeClusterC
    → KILLED
140 1. removed call to nl/tudelft/sem/resource/manager/domain/services/Manager::removeNodeFromClus
141 1. replaced return value with "" for nl/tudelft/sem/resource/manager/controllers
    /NodeClusterController::removeClusterNode → KILLED
143 1. replaced return value with "" for nl/tudelft/sem/resource/manager/controllers
    /NodeClusterController::removeClusterNode → KILLED

```

The respective commit to these changes can be found [here](#).

- II. From the report generated by *Pitest* regarding mutation score for *Manager.java*, we can see that this score of 58% is partly due to the non-covered lines, but also due to some surviving mutants.

Mutations

```

37 1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
   /Manager::getDateForRequest → NO_COVERAGE
48 1. removed call to nl/tudelft/sem/resource/manager/domain/services
   /ResourceHandler::releaseResourcesOnDays → NO_COVERAGE
66 1. removed call to nl/tudelft/sem/resource/manager/domain/services
   /ResourceHandler::reserveResourcesOnDay → NO_COVERAGE
75 1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
   /Manager::seeFreeResourcesOnDate → NO_COVERAGE
86 1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
   /Manager::seeFreeResourcesTomorrow → NO_COVERAGE
94 1. replaced return value with Collections.emptyList for nl/tudelft/sem/resource/manager/doma
   /services/Manager::seeClusterNodeInformation → KILLED
104 1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
    /Manager::seeReservedResourcesOnDate → NO_COVERAGE
113 1. negated conditional → KILLED
114 1. replaced return value with "" for nl/tudelft/sem/resource/manager/domain/services
    /Manager::addNodeToCluster → SURVIVED
117 1. replaced return value with "" for nl/tudelft/sem/resource/manager/domain/services
    /Manager::addNodeToCluster → KILLED
126 1. negated conditional → KILLED
129 1. removed call to nl/tudelft/sem/resource/manager/domain/node/NodeRepository::removeByToken
    KILLED

```

With this information, we added some methods to cover the lines that appeared uncovered, and added a test to kill the only covered but surviving mutant. After such changes, these were our scores:

Breakdown by Class

Name	Line Coverage	Mutation Coverage
DateSchedulingService.java	100% 12/12	70% 7/10
FreepoolManager.java	100% 15/15	100% 1/1
Manager.java	94% 29/31	93% 13/14
ResourceAvailabilityService.java	94% 31/33	88% 7/8
ResourceHandler.java	97% 38/39	83% 10/12

It is worth noting that we have also changed the code of the target class in order to make testing for *point a* easier. Here are the mutants that survived. We have decided that killing these remaining mutants would require changing the test suite in a way that wouldn't increase the bug finding capability on the system under the test here.

Mutations

```

1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
/Manager::getDateForRequest → KILLED
1. removed call to nl/tudelft/sem/resource/manager/domain/services/ResourceHandler::releaseResource
KILLED
1. removed call to nl/tudelft/sem/resource/manager/domain/services/ResourceHandler::reserveResource
KILLED
1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
/Manager::seeFreeResourcesOnDate → KILLED
1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
/Manager::seeFreeResourcesTomorrow → KILLED
1. replaced return value with Collections.emptyList for nl/tudelft/sem/resource/manager/domain/se
/Manager::seeClusterNodeInformation → KILLED
1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
/Manager::seeReservedResourcesOnDate → SURVIVED
1. negated conditional → KILLED
1. replaced return value with "" for nl/tudelft/sem/resource/manager/domain/services
/Manager::addNodeToCluster → KILLED
1. replaced return value with "" for nl/tudelft/sem/resource/manager/domain/services
/Manager::addNodeToCluster → KILLED
1. negated conditional → KILLED
1. removed call to nl/tudelft/sem/resource/manager/domain/node/NodeRepository::removeByToken → KIL
1. removed call to java/util/EnumSet::forEach → KILLED
1. replaced return value with null for nl/tudelft/sem/resource/manager/domain/services
/Manager::getAvailableResourcesForAllFacultiesOnDate → KILLED

```

You can find the respective commit [here](#).

- III. From the report generated by *Pitest* regarding mutation score for *MainFacultyController.java*, we can see that this score of 0% is partly due to the non-covered lines, but mostly due to some surviving mutants. To note here that, even if the class had 92% line coverage, the tests that were already in place are not effective in killing these mutants.

Mutations

```

43 1. replaced return value with null for sem/faculty/controllers/MainFacultyController::listener → NO_COVERAGE
59 1. removed call to java/io/PrintStream::println → SURVIVED
60 1. replaced return value with null for sem/faculty/controllers/MainFacultyController::getPendingRequests → SURVIVED
76 1. removed call to java/io/PrintStream::println → SURVIVED
78 1. replaced return value with null for sem/faculty/controllers/MainFacultyController::acceptRequests → SURVIVED
96 1. replaced return value with null for sem/faculty/controllers/MainFacultyController::getScheduleForDate → SURVIVED

```

With this information, we added more focused tests to kill all the mutants, as well as a test to cover the previously uncovered lines. After these changes, all the mutants were killed, and we have achieved a mutation coverage of 100% as can be seen below:

Mutations	
43	1. replaced return value with null for sem/faculty/controllers/MainFacultyController::listener → KILLED
59	1. removed call to java/io/PrintStream::println → KILLED
60	1. replaced return value with null for sem/faculty/controllers/MainFacultyController::getPendingRequests → KILLED
76	1. removed call to java/io/PrintStream::println → KILLED
78	1. replaced return value with null for sem/faculty/controllers/MainFacultyController::acceptRequests → KILLED
96	1. replaced return value with null for sem/faculty/controllers/MainFacultyController::getScheduleForDate → KILLED

Breakdown by Class

Name	Line Coverage	Mutation Coverage
MainFacultyController.java	100% 13/13	100% 6/6
ScheduleRequestController.java	0% 0/22	0% 0/4

You can find the respective commit [here](#).

- IV. From the report generated by *Pitest* regarding mutation score for *AcceptRequestScheduler.java*, we can see that this score of 45% is partly due to the non-covered lines, but also due to one surviving mutant.

Mutations	
36	1. negated conditional → KILLED
37	1. removed call to sem/faculty/domain/Request::setStatus → NO_COVERAGE
38	1. negated conditional → NO_COVERAGE
39	1. removed call to sem/faculty/domain/RequestRepository::delete → NO_COVERAGE
44	1. removed call to sem/faculty/domain/Request::setStatus → KILLED
50	1. negated conditional → SURVIVED
51	1. removed call to sem/faculty/domain/RequestRepository::updateRequestStatusAccepted → NO_COVERAGE
56	1. removed call to sem/faculty/domain/Faculty::scheduleForDate → KILLED
74	1. negated conditional → KILLED
75	1. replaced boolean return with false for sem/faculty/domain/scheduler/AcceptRequestsScheduler::reserveResource → KILLED
83	1. replaced boolean return with true for sem/faculty/domain/scheduler/AcceptRequestsScheduler::reserveResource → NO_COVERAGE

To solve this, we have added a few more tests that have covered the rest of the code, as well as killed all the remaining mutants. Also, due to the fact that the *reserveResources* method is private, the functionality and possible mutants have to be checked by testing the previous method. In the end, all the mutants have been killed and the mutation score reached 100% as can be seen below:

Mutations	
36	1. negated conditional → KILLED
37	1. removed call to sem/faculty/domain/Request::setStatus → KILLED
38	1. negated conditional → KILLED
39	1. removed call to sem/faculty/domain/RequestRepository::delete → KILLED
44	1. removed call to sem/faculty/domain/Request::setStatus → KILLED
50	1. negated conditional → KILLED
51	1. removed call to sem/faculty/domain/RequestRepository::updateRequestStatusAccepted → KILLED
56	1. removed call to sem/faculty/domain/Faculty::scheduleForDate → KILLED
74	1. negated conditional → KILLED
75	1. replaced boolean return with false for sem/faculty/domain/scheduler/AcceptRequestsScheduler::reserveResource → KILLED
83	1. replaced boolean return with true for sem/faculty/domain/scheduler/AcceptRequestsScheduler::reserveResource → KILLED

Breakdown by Class

Name	Line Coverage	Mutation Coverage
AcceptRequestsScheduler.java	100% 29/29	100% 11/11
DenyRequestsScheduler.java	88% 7/8	33% 1/3
PendingRequestsScheduler.java	89% 8/9	50% 2/4
SchedulableRequestsScheduler.java	0% 0/28	0% 0/8

You can find the respective commit [here](#).

Task 2

We have chosen to do our manual mutation testing in the faculty domain. This domain performs the most important functionality of the system - adding requests. There are many points where we validate input and save or retrieve data from databases, in which accidental bugs that might have slipped past our attention would cause serious damage to the reliability of our system.

Class I - PendingRequestScheduler

The first class that we will verify is *faculty-microservice/src/main/java/sem/faculty/domain/scheduler/PendingRequestsScheduler.java*. Although this is not a very large class, it is rather important. When requests come in, this class changes the status of the request to *pending* in the database. This makes it show up for the reviewer, so it can be either denied or accepted. If this class were to have a bug, this would completely block most requests from making it to the schedule.

This class has one method that has all the important code. A clear point that has to be tested is the if-statement, because decision points have many mutations that might survive.

```
if (Objects.equals(requestRepository.findByRequestId(requestID), request)) {  
    requestRepository.updateRequestStatusPending(requestID);  
} else {  
    requestRepository.saveAndFlush(request);  
}
```

Currently the test suite passes. Now we negate the if-statement, which is a conditional operator replacement (COR):

```
if (!Objects.equals(requestRepository.findByRequestId(requestID), request)) {  
    requestRepository.updateRequestStatusPending(requestID);  
}
```

After doing this, the test suite still passes, even though it clearly should not. The problem is that the test suite mocks the *requestRepository*, but doesn't test its invocations. So we add a test, which can be found through the link below. Now the test suite doesn't pass and the mutation is killed.

The commit which adds this test can be found [here](#).

Class II - FacultyHandlerService

The next class we chose is *faculty-microservice/src/main/java/sem/faculty/handler/FacultyHandlerService.java*. This class is responsible for the validation of all incoming requests sent to the faculty microservice. It contains a method *requestListener()* which validates incoming requests to schedule a job before passing them further. An important part of this method is a check for whether the preferred date of the request is today or in the past. A bug in this functionality would be system-breaking, as it definitely should not be possible to schedule requests in the past.

Here is the check without mutant:

```
if (!requestDate.isAfter(facultyHandler.timeProvider.getCurrentDate())) {  
    return new StatusDTO("You cannot schedule requests for today or the past!");  
}
```

Now we can introduce a mutant by replacing the conditional operator (COR). The code with mutant, which is not killed by our test suite:

```
if (requestDate.isAfter(facultyHandler.timeProvider.getCurrentDate())) {  
    return new StatusDTO("You cannot schedule requests for today or the past!");  
}
```

The commit which adds a test that catches the bug (along with small changes to production code) can be found [here](#).

Class III- FacultyHandler

Another class we chose is

faculty-microservice/src/main/java/sem/faculty/handler/FacultyHandler.java, which is responsible for handling requests, either by scheduling them, or retrieving them, both crucial parts of the system. It has the method *getPendingRequests()*, which is vital to the microservice, as it is a step in the process of retrieving pending requests from the database. A bug in this method would make it so reviewers cannot access the pending requests, which would stop any new requests from being accepted, and then subsequently scheduled by the system.

Below is the method before the mutant is introduced:

```
118 public List<Request> getPendingRequests(FacultyName facultyName) {  
119     Faculty faculty = faculties.get(facultyName);  
120     List<Request> requests = faculty.getPendingRequests().stream() Stream<Long>  
121         .map(x -> requestRepository.findById(x)) Stream<Request>  
122         .collect(Collectors.toList());  
123     return requests;  
124 }
```

Now we introduce a mutant by changing the return value of the method to be *Collections.emptyList()*. Even though the method is completely non-functional, the mutant survives our test suite. Below is the altered code:

```
118 public List<Request> getPendingRequests(FacultyName facultyName) {  
119     Faculty faculty = faculties.get(facultyName);  
120     List<Request> requests = faculty.getPendingRequests().stream() Stream<Long>  
121         .map(x -> requestRepository.findById(x)) Stream<Request>  
122         .collect(Collectors.toList());  
123     return Collections.emptyList();  
124 }
```

The commit which adds the test for this mutant is found [here](#).

Class IV - ScheduleRequestController

The last class we chose is

faculty-microservice/src/main/java/sem/faculty/controllers/ScheduleRequestController.java.

This class is vital to our application as it uses kafka for communication between the faculty and the resource manager. Any issues here would create errors between the communication of these two microservices. It has the method *sendScheduleRequest()*, which sends a *scheduleDateDTO* to the topic “schedule-date”, and in return receives a response entity with the *LocalDate* for when there are enough resources available for the request to be carried out. If a bug were to exist here, it would severely disrupt the application, however, there are currently no tests for this class, so replacing the return value with null would still make the test suite pass.

Here is the return statement present before the mutation:

```
return ResponseEntity.ok(consumerRecord.value());
```

Here is the return statement after the mutation:

```
return null;
```

To fix this issue, a test was added for the *sendScheduleRequest()* method. *The commit which adds the test for this mutant can be found [here](#).*