

Assignment 3

Group 27a: Ata Ağırkaya, Alex Bolfă, Lauren de Hoop, Ioan Hedeia, Paris Loizides, Rok Štular

Task 1

UserEntity Class

The UserEntity class represents a pivotal role in our user management system because it is the core logic of the user functionality that we have to deal with throughout our app. It preserves stability in our data management system in the UserEntityRepository. The core change that generated 33% mutant score improvement was the addition of the null checkers that deal with the role, the netId, the UUID, and other fields being null. We introduced a **negation conditional mutant** to check if any field was not null. We pretended as if the field was null when in reality it was and this is how the mutant was killed.

no usages ⓘ ioan

@Test

```
void TestUserDtoNull() {  
    UserEntity ue = new UserEntity();  
    assertNotNull(ue.getDto());  
}
```

no usages ⓘ ioan

@Test

```
void TestUserEntityNotNull() {  
    UserEntity ue = new UserEntity(netId: "", Role.CANDIDATE, address: "", description: "", firstName: "", lastName:  
    assertNotNull(ue.getNetId());  
}
```

nl.tudelft.sem.user.microservice.database.entities

| Number of Classes | Line Coverage | Mutation Coverage | Test Strength |
|-------------------|---------------------------------|-------------------------------|--------------------------------|
| 1 | 85% <div><div>11/13</div></div> | 33% <div><div>1/3</div></div> | 100% <div><div>1/1</div></div> |

Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|---------------------------------|---------------------------------|-------------------------------|--------------------------------|
| UserEntity.java | 85% <div><div>11/13</div></div> | 33% <div><div>1/3</div></div> | 100% <div><div>1/1</div></div> |

nl.tudelft.sem.user.microservice.database.entities

| Number of Classes | Line Coverage | Mutation Coverage | Test Strength |
|-------------------|---------------------------------------------|------------------------------------------|------------------------------------------|
| 1 | 100% <div><div></div><div>13/13</div></div> | 67% <div><div></div><div>2/3</div></div> | 67% <div><div></div><div>2/3</div></div> |

Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|---------------------------------|---------------------------------------------|------------------------------------------|------------------------------------------|
| UserEntity.java | 100% <div><div></div><div>13/13</div></div> | 67% <div><div></div><div>2/3</div></div> | 67% <div><div></div><div>2/3</div></div> |

Commit:

https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM27a/-/merge_requests/94/diffs?commit_id=8e0c3e2e5e2a6c1e567f394956dece6c9d0a6641

UserController Class:

| Number of Classes | Line Coverage | Mutation Coverage | Test Strength |
|-------------------|-------------------------------------------|------------------------------------------|-----------------------------------------|
| 1 | 17% <div><div></div><div>7/41</div></div> | 0% <div><div></div><div>0/15</div></div> | 0% <div><div></div><div>0/0</div></div> |

Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|-------------------------------------|-------------------------------------------|------------------------------------------|-----------------------------------------|
| UserController.java | 17% <div><div></div><div>7/41</div></div> | 0% <div><div></div><div>0/15</div></div> | 0% <div><div></div><div>0/0</div></div> |

nl.tudelft.sem.user.microservice.controllers.internal

| Number of Classes | Line Coverage | Mutation Coverage | Test Strength |
|-------------------|--------------------------------------------|-------------------------------------------|-----------------------------------------|
| 1 | 47% <div><div></div><div>19/41</div></div> | 21% <div><div></div><div>4/15</div></div> | 0% <div><div></div><div>0/0</div></div> |

Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|-------------------------------------|--------------------------------------------|-------------------------------------------|-------------------------------------------|
| UserController.java | 47% <div><div></div><div>19/41</div></div> | 21% <div><div></div><div>4/15</div></div> | 100% <div><div></div><div>4/4</div></div> |

| | | |
|----------------------------------------------------------------------------------------------------------------|-----|------------------------------------------------------------------------------------------|
| <div> <div></div> <div>Show 20 lines</div> <div>Show all unchanged lines</div> <div>Show 20 lines</div> </div> | | |
| 217 | 246 | @@ -217,4 +246,29 @@ class UserServiceTest { |
| 218 | 247 | UserEntity ue = new UserEntity("", Role.CANDIDATE, "", "", "", "", "", ""); |
| 219 | 248 | assertNotNull(ue.getNetId()); |
| | 249 | } |
| | 250 | + @Test |
| | 251 | + void getNetIdIsNull() throws Exception { |
| | 252 | + assertNotNull(this.mockMvc.perform(get("/internal/user/{id}", (Object) null))); |
| | 253 | + this.mockMvc.perform(get("/internal/user/{id}", (Object) null)) |
| | 254 | + .andExpect(status().is(404)); |
| | 255 | + } |
| | 256 | + } |
| | 257 | + @Test |
| | 258 | + void getNonExistingUser() throws Exception { |
| | 259 | + this.mockMvc.perform(get("/internal/user/{id}", getUuid(1))) |
| | 260 | + .andExpect(status().is(404)); |
| | 261 | + } |
| | 262 | + } |
| | 263 | + @Test |
| | 264 | + void userNotSaved() throws Exception { |
| | 265 | + UserEntity userEntity = new UserEntity("newId", Role.CANDIDATE, "", "", |
| | 266 | + "", "", "", ""); |
| | 267 | + UUID uuid = userEntity.getUuid(); |
| | 268 | + //when(userEntityRepository.findById(uuid)).thenReturn(Optional.of(userEntity)); |
| | 269 | + this.mockMvc.perform(get("/getRoleByUUID/{id}", "newId")).andExpect(status().is(401)); |
| | 270 | + } |
| | 271 | + } |
| | 272 | + } |
| | 273 | + } |
| 220 | 274 | } |

The crucial point was to check any null cases when the endpoint was called. Thus, the null mutants were killed successfully.

Commit:

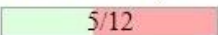
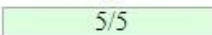
https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM27a/-/merge_requests/93/diffs?commit_id=6e633ef653dfdc795e612b63e65c50b4e0b9cd0a

RequestService Class:

(Commit: 2df86ed2f0d24eb4e62830d445d347c821453c61)

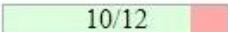
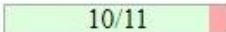
In the case of the RequestService class, the initial mutation coverage rate was 42%. This suggests that the test suite was not effectively detecting a significant portion of the code, and therefore, the code may contain bugs that are not being caught.

Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|-------------------------------------|---------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| RequestService.java | 24%  | 42%  | 100%  |

After adding additional tests, the mutation coverage rate for the RequestService class increased to 83%. This suggests that the new tests were able to detect a much larger portion of the code that was affected by mutations, thus indicating that the test suite is now more effective at catching bugs in the code.

Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|-------------------------------------|---------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| RequestService.java | 80%  | 83%  | 91%  |

ContractPublicController Class:

(Commit: 48c7c3eafa2b67abc2ee794cfcc741efe8d02ca5)

The addition of the extra test cases for the ContractPublicController class has led to an increase in the mutation coverage of the class, as measured by the Pitest tool. This suggests that the new tests are effectively exercising a wider range of code paths within the class, and are therefore providing a stronger level of confidence in the class's overall correctness and robustness.

| Name | Line Coverage | Mutation Coverage | Test Strength |
|-----------------------------------------------|-----------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| ContractPublicController.java | 21%  | 0%  | 0%  |

The addition of new test cases that covered additional code paths and scenarios helped increase the mutation coverage of the class from 0% to 38%.

Breakdown by Class

| Name | Line Coverage | Mutation Coverage | Test Strength |
|-----------------------------------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| ContractPublicController.java | 59%  | 38%  | 100%  |

Task 2

Contract Class:

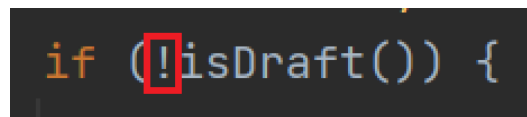
(commit: [33a951380d6038679cdcf2fedf4f1a7822cefe97](#))

The `Contract` class represents a critical class for the Contract microservice because it lies at the heart of its logic. It is the very data that we store inside of our contract database and it influences the behavioural core of our program.

An essential part of this class is the `modifyDraft()` method which should allow HR employees and candidates to make modifications to a draft contract during the negotiation phase. These modifications should not be possible on an already existing contract (not draft). This is guaranteed by if-statement (*line 110*) which checks whether the contract on which we apply the changes is a draft or not. If the latter is true, the method throws an exception.

We introduced a **negating conditional mutant** (Fig. 1) on this specific if-condition statement to check whether the test suite checks for this behaviour constraint (i.e. changing already existing contracts should not be allowed). Since the mutant *survived*, we added two additional tests:

- One that checks that the draft contract has changed after the method call.
- One that checks whether the method throws an exception when we try to modify an already existing (not draft) contract.



```
if (!isDraft()) {
```

Fig. 1

After implementing these two tests we can see that the mutant is **killed**. This was a vulnerability in our system that might have led to bigger issues as the scope of the project grew bigger. Having fixed this now we made sure to decrease the testing-related technical debt of our project.

Commit: [33a951380d6038679cdcf2fedf4f1a7822cefe97](#)

JobPositionController Class:

The JobPositionController is a very important class for our System. It is the class responsible for all the Rest APIs of JobPosition entity. The JobPosition is important since it is the only way of keeping track of an employee's job in the system through the Contract. Furthermore, each JobPosition is correlated to a specific salary scale that the employee is entitled to according to his job requirements. Therefore we can conclude that this class is necessary for creating a system in accordance with the system requirements.

A core method for the class is the deleteJobPosition() method that is necessary for removing a job position from the database since the job position names and responsibilities can change overtime therefore modifying them is necessary. Even though the method was tested already, we managed to introduce a mutant by deleting the functionality of deleting the job

position(fig.2), using its Id, from the database (line 77). The motivation behind the choice of mutant was that we knew that the specific line was always called in the method however we could have missed the part where we should actually test if everything went as planned during the method call. After the deletion we ran the test suite for JobPositionController and the tests passed, therefore the mutant had survived. As a result we added an additional test that verifies the entire method is tested correctly and successfully kills all mutants.

Commit: [37e00254befc543a0edea0626904e0c94c8e9edb](#)

```
@DeleteMapping("/{id}")
ResponseBody<JobPositionDto> deleteJobPosition(@PathVariable UUID id) {
    JobPositionDto jobPositionDto = jobPositionRepository.findById(id)
        .orElseThrow(JobPositionNotFoundException::new).getDto();
    jobPositionRepository.deleteById(id);
    return ResponseEntity.ok(jobPositionDto);
}
```

SalaryScaleController Class:

As for all of the controllers of the contract microservice, the SalaryScaleController class can be considered as critical to the functioning of our application. For a contract to be legally valid, one of the elements a contract must contain is a salary scale.

These salary scales are all saved within the contract database and accessed mainly through the salary scale controller. If any of the functions within this controller do not operate as they should the salary scales will not be properly saved, or accessible to HR. For this task, the addSalaryScale method will be tested. It is crucial for the HR employees to be able to add salary scales, so they can actually assign them.

For the mutant, I have chosen the following:

```
@PostMapping("")
ResponseBody<SalaryScaleDto> addSalaryScale(@RequestBody SalaryScaleDto salaryScaleDto) {
    SalaryScale salaryScale = salaryScaleRepository.save(new SalaryScale(salaryScaleDto));
    return ResponseEntity.created(URI.create(INTERNAL_PATH + "/salary-scale/" + salaryScale.getId()))
        .body(salaryScale.getDto());
}
```

```
@PostMapping("")
ResponseBody<SalaryScaleDto> addSalaryScale(@RequestBody SalaryScaleDto salaryScaleDto) {
    SalaryScale salaryScale = new SalaryScale(salaryScaleDto);
    return ResponseEntity.created(URI.create(INTERNAL_PATH + "/salary-scale/" + salaryScale.getId()))
        .body(salaryScale.getDto());
}
```

For the POST mapping of this controller, the tests never actually verify if the new salary scale is saved in the correct repository.

To kill this mutant I have added a test to verify if the repository save is called within the function. The added test will verify if any saves have been made to the SalaryScale repository. This mutant has passed through the original test suite, but not with the updated test.

Commit: [0949edbe0d15ce9407c149ee66833e1ed25cfa5](#)

ContractService Class:

ContractService is a Spring Bean which provides functionality for managing contracts. In the context of an HR management application, it provides core functionality for the application, and is needed to properly track salaries, taxes, etc. As such, it is mission-critical for the application, as any bugs in the class could cause major disruptions, legal problems, if, for example, contract constraints are not properly enforced, or financial loss, if events aren't thoroughly tracked. One method residing within the *ContractService* class is the *terminateContract* method. As the name suggests, the method can be used to (unilaterally) terminate a contract of a specific employee - when they are fired, for example.

When this happens, it is crucial that the system keeps track of the termination date, as missing this could expose the business to a lot of legal trouble, as it is not clear *when* the termination happened. This problem can snowball into possible disputes about owed pay (e.g. it is not clear how much money the employee is owed), taxes (the business does not know how much taxes it owes to the governmental agencies) and so on.

Current test suite does not verify that this date has been properly set. The mutant, which survives the tests looks like this (note how line 68 has been commented out and as such the system doesn't store the termination date):

```
56  /**
57   * Mark a contract as terminated.
58   *
59   * @param contractId the id of the contract to terminate.
60   */
61  8 usages  rstular *
62  @Transactional
63  public void terminateContract(UUID contractId) {
64      Contract contract = contractRepository.findById(contractId).orElseThrow(ContractNotFoundException::new);
65      if (contract.getContractInfo().getStatus() != ContractStatus.ACTIVE) {
66          throw new ActionNotAllowedException("Only active contracts can be terminated.");
67      }
68      contract.getContractInfo().setStatus(ContractStatus.TERMINATED);
69      // contract.getContractTerms().setTerminationDate(LocalDate.now());
70      contractRepository.save(contract);
71  }
```

In order to “kill” this mutant, the test suite was extended to include the verification of the termination date, including a check to prevent anyone from backdating the termination date (as this is most likely very illegal).

The test was added in **commit:** [329de2b8ef5b17b0fd67f4560fda2d830f314c97](https://github.com/rstular/contract-service/commit/329de2b8ef5b17b0fd67f4560fda2d830f314c97).