

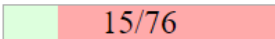
Assignment 3 Testing

Task 1

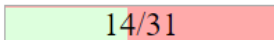
ActivityCheckerService

We had the following mutation score before creating any new tests for this class:

nl.tudelft.sem.project.matchmaking.services

Number of Classes	Line Coverage	Mutation Coverage
2	37% 	20% 

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ActivityCheckerService.java	45% 	22% 

Since we have only a 20% coverage, we can improve it easily by a lot.

We managed to increase it all the way up to 97% missing just one mutation.

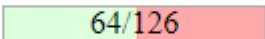
The mutation that was left was the following:

```
public boolean isAllowedToJoinWithTime(ActivityDTO activity, Instant now) {  
    Instant activityTime = activity.getStartTime().toInstant();  
    Instant shouldJoinBefore = activityTime.minusSeconds(secondsToActivityStart);  
    return now.compareTo(shouldJoinBefore) <= 0;  
}
```

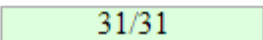
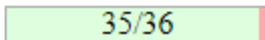
Since we are comparing times and they are almost never equal it is very hard to catch this mutation if not impossible.

The final stats of this class are the following:

nl.tudelft.sem.project.matchmaking.services

Number of Classes	Line Coverage	Mutation Coverage
2	51% 	55% 

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ActivityCheckerService.java	100% 	97% 

The link to the commit that incorporates these changes is the following:

<https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM33c/-/commit/a9b8f4879be2f3444ae64f3935ad984a56050286>

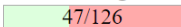
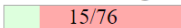
MatchmakingService

Before improving our test suite for MatchmakingService, we had the following PitTest report:

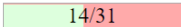
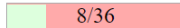
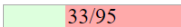
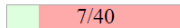
Pit Test Coverage Report

Package Summary

nl.tudelft.sem.project.matchmaking.services

Number of Classes	Line Coverage	Mutation Coverage
2	37%  47/126	20%  15/76

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ActivityCheckerService.java	45%  14/31	22%  8/36
MatchmakingService.java	35%  33/95	18%  7/40

The report was at 18% and after adding the new tests, we improved it to 65%. As a result, line coverage also got a considerable improvement in percentage.

MatchmakingService.java	87%  83/95	65%  26/40
---	--	--

The link to the commit that incorporates the changes is the following:

<https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM33c/-/commit/aa4dfcc049e0cad79b66995a6227e068a14e921d>

AdminController in GatewayMicroservice

The AdminController is in the following state before any changes. Currently, it has 44% line coverage and a low 14% (2/14) mutation coverage

Pit Test Coverage Report

Package Summary

nl.tudelft.sem.project.gateway.controllers

Number of Classes	Line Coverage	Mutation Coverage
8	86% <div><div>232/270</div></div>	53% <div><div>45/85</div></div>

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ActivityController.java	88% <div><div>59/67</div></div>	78% <div><div>14/18</div></div>
ActivityRegistrationController.java	90% <div><div>55/61</div></div>	40% <div><div>8/20</div></div>
AdminController.java	44% <div><div>11/25</div></div>	14% <div><div>2/14</div></div>
AuthenticationController.java	88% <div><div>44/50</div></div>	29% <div><div>4/14</div></div>
BoatController.java	20% <div><div>1/5</div></div>	0% <div><div>0/2</div></div>
CertificatesController.java	100% <div><div>9/9</div></div>	100% <div><div>4/4</div></div>
MatchmakingController.java	100% <div><div>5/5</div></div>	100% <div><div>2/2</div></div>
UserController.java	100% <div><div>48/48</div></div>	100% <div><div>11/11</div></div>

After adding new tests we achieved 100% line coverage and 100% mutation coverage. The image below depicts this.

nl.tudelft.sem.project.gateway.controllers

Number of Classes	Line Coverage	Mutation Coverage
8	91% <div><div>246/270</div></div>	68% <div><div>58/85</div></div>

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ActivityController.java	88% <div><div>59/67</div></div>	78% <div><div>14/18</div></div>
ActivityRegistrationController.java	90% <div><div>55/61</div></div>	40% <div><div>8/20</div></div>
AdminController.java	100% <div><div>25/25</div></div>	100% <div><div>14/14</div></div>
AuthenticationController.java	88% <div><div>44/50</div></div>	36% <div><div>5/14</div></div>
BoatController.java	20% <div><div>1/5</div></div>	0% <div><div>0/2</div></div>
CertificatesController.java	100% <div><div>9/9</div></div>	100% <div><div>4/4</div></div>
MatchmakingController.java	100% <div><div>5/5</div></div>	100% <div><div>2/2</div></div>
UserController.java	100% <div><div>48/48</div></div>	100% <div><div>11/11</div></div>

Link to commit with changes:

https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM33c/-/merge_requests/68/diffs?commit_id=1203562a40f58554771af580787886b422b6bd6b

BoatController in GatewayMicroservice

The BoatController started with the following mutation score:

Pit Test Coverage Report

Package Summary

nl.tudelft.sem.project.gateway.controllers

Number of Classes	Line Coverage	Mutation Coverage
8	86% <div><div></div><div></div></div> 232/270	53% <div><div></div><div></div></div> 45/85

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ActivityController.java	88% <div><div></div><div></div></div> 59/67	78% <div><div></div><div></div></div> 14/18
ActivityRegistrationController.java	90% <div><div></div><div></div></div> 55/61	40% <div><div></div><div></div></div> 8/20
AdminController.java	44% <div><div></div><div></div></div> 11/25	14% <div><div></div><div></div></div> 2/14
AuthenticationController.java	88% <div><div></div><div></div></div> 44/50	29% <div><div></div><div></div></div> 4/14
BoatController.java	20% <div><div></div><div></div></div> 1/5	0% <div><div></div><div></div></div> 0/2
CertificatesController.java	100% <div><div></div><div></div></div> 9/9	100% <div><div></div><div></div></div> 4/4
MatchmakingController.java	100% <div><div></div><div></div></div> 5/5	100% <div><div></div><div></div></div> 2/2
UserController.java	100% <div><div></div><div></div></div> 48/48	100% <div><div></div><div></div></div> 11/11

With not a lot of mutants to catch and this being a pretty important class for boat access, we can push it to 100% mutation coverage:

Pit Test Coverage Report

Package Summary

nl.tudelft.sem.project.gateway.controllers

Number of Classes	Line Coverage	Mutation Coverage
8	87% <div><div></div><div></div></div> 236/270	55% <div><div></div><div></div></div> 47/85

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ActivityController.java	88% <div><div></div><div></div></div> 59/67	78% <div><div></div><div></div></div> 14/18
ActivityRegistrationController.java	90% <div><div></div><div></div></div> 55/61	40% <div><div></div><div></div></div> 8/20
AdminController.java	44% <div><div></div><div></div></div> 11/25	14% <div><div></div><div></div></div> 2/14
AuthenticationController.java	88% <div><div></div><div></div></div> 44/50	29% <div><div></div><div></div></div> 4/14
BoatController.java	100% <div><div></div><div></div></div> 5/5	100% <div><div></div><div></div></div> 2/2
CertificatesController.java	100% <div><div></div><div></div></div> 9/9	100% <div><div></div><div></div></div> 4/4
MatchmakingController.java	100% <div><div></div><div></div></div> 5/5	100% <div><div></div><div></div></div> 2/2
UserController.java	100% <div><div></div><div></div></div> 48/48	100% <div><div></div><div></div></div> 11/11

Link to the GitLab commit:

https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM33c/-/merge_requests/66/diffs?commit_id=79614ab0375c9121c7d2cab98a4067558df005c3

Task 2

We have chosen to tackle our activity core domain. This is a critical part of our system as activities are the main component our app revolves around. Activities have connections to every other component of the system and they are integral to the functioning of our app. As such it is really important this part is properly tested and that we can always make sure it works as intended.

We have chosen to tackle four different classes from this microservice. We will tackle the Service and Controller of both activities and boats. Boats are connected to activities and as such, we grouped them together in the same microservice.

ActivityService

The activity service class ensures that the communication between the controller and the repository happens without any issues. It handles things such as adding a boat to an activity and adding activities. Thus, it is a critical class.

The `addBoatToActivity()` method should normally throw an exception if the activity requested is not found. We concluded that a potential mutation could be not throwing this exception.

```
Adds a new boat to an activity.
Params: activityId – The activity to change.
        boat – The new boat to add. This boat is not validated to be present within the boat repository.
Returns: An activity object with the new boat added.
Throws: ActivityNotFoundException – Thrown if there is no such activity.

public Activity addBoatToActivity(UUID activityId, Boat boat) throws ActivityNotFoundException {
    var optActivity : Optional<Activity> = activityRepository.findById(activityId);
    if (optActivity.isEmpty()) {
        throw new ActivityNotFoundException(couldNotFindActivityErrorMessage);
    }
    var activity : Activity = optActivity.get();
    activity.getBoats().add(boat);
    return activityRepository.save(activity);
}
```

We introduced the following mutant: removing the exception throw if the activity is not found. The mutant is of type “Void Method Call Mutator”.

```

    Adds a new boat to an activity.
    Params: activityId – The activity to change.
           boat – The new boat to add. This boat is not validated to be present within the boat repository.
    Returns: An activity object with the new boat added.
    Throws: ActivityNotFoundException – Thrown if there is no such activity.

public Activity addBoatToActivity(UUID activityId, Boat boat) throws ActivityNotFoundException {
    var optActivity : Optional<Activity> = activityRepository.findById(activityId);
    if (optActivity.isEmpty()) {
        ;
    }
    var activity : Activity = optActivity.get();
    activity.getBoats().add(boat);
    return activityRepository.save(activity);
}

```

The following test case ensures that the proper exception is thrown in case the activity is not found:

```

@Test
void missingActivityThrowsException() {
    var activityId : UUID = UUID.randomUUID();

    when(activityRepository.findById(activityId)).thenReturn(Optional.empty());
    assertThatExceptionOfType(ActivityNotFoundException.class)
        .isThrownBy(() -> activityService.addBoatToActivity(activityId, new Boat()));
}

```

The test case now ensures that the method works as intended for this edge case.

Link to commit: [Add manual mutation tests for BoatService and ActivityService \(bd5186cc\) · Commits · CSE2115 - Software Engineering Methods / 2022-2023 / SEM-33c · GitLab \(tudelft.nl\)](#)

BoatService

The boat service class deals with adding and removing boats to and from the database, changing details about specific boats, and updating them. Thus, it is a critical class.

The `changeCoxCertificate()` method changes the certificate that is assigned to a specific boat.

```

Changes the required certificate for the boat.
Params: boatID – The id of the boat to edit.
        newCertificateID – The id of the new certificate to require.
Returns: The boat with the certificate changed.
Throws: BoatNotFoundException – Thrown if there is no such boat.

public Boat changeCoxCertificate(UUID boatID, UUID newCertificateID) throws BoatNotFoundException {
    var boat : Boat = this.getBoatById(boatID);
    boat.setCoxCertificateId(newCertificateID);
    return boatRepository.save(boat);
}

```

One potential mutation could be that the method does not actually change the boat's certificate, as seen below:

```

Changes the required certificate for the boat.
Params: boatID – The id of the boat to edit.
        newCertificateID – The id of the new certificate to require.
Returns: The boat with the certificate changed.
Throws: BoatNotFoundException – Thrown if there is no such boat.

public Boat changeCoxCertificate(UUID boatID, UUID newCertificateID) throws BoatNotFoundException {
    var boat : Boat = this.getBoatById(boatID);

    return boatRepository.save(boat);
}

```

The mutant introduced is of type "Void Method Call Mutator".

The following test case ensures that the method properly changes a boat's required certificate, thus covering this functionality.

```

@Test
void changeCoxCertificateActuallyChangesIt() {
    var certificateId : UUID = UUID.randomUUID();
    Boat addedBoat = boatService.addBoat(normalBoat);
    boatService.changeCoxCertificate(addedBoat.getId(), certificateId);
    assertThat(addedBoat.getCoxCertificateId()).isEqualTo(certificateId);
}

```

The mutation is now successfully killed.

Link to commit: [Add manual mutation tests for BoatService and ActivityService \(bd5186cc\)](https://github.com/tudelft/cse2115-2022-2023/commit/bd5186cc) · Commits · CSE2115 - Software Engineering Methods / 2022-2023 / SEM-33c · GitLab (tudelft.nl)

ActivityController

This class is critical to our system. It is the controller for activities and handles tasks such as creating activities, managing them, and getting information. As such the quality of this class is of utmost importance to us.

We identified one method by which we could find a mutant that we could introduce without our tests detecting it. The following method looked like this before the mutant:

```
/**
 * Adds a new boat to a given activity. If the given boat is not one
 * in the database it is also added on top of that.
 *
 * @param activityId ID of the activity.
 * @param boatDTO Boat DTO object to add.
 * @return The activity object after it has changed.
 */
1 usage 1 Stiliyan Nanovski
@PostMapping("/add_boat_to_activity")
/PMD/
public ResponseEntity<ActivityDTO> addBoatToActivity(
    @Valid @NotNull @RequestParam UUID activityId,
    @Valid @NotNull @RequestBody BoatDTO boatDTO
) {
    BoatDTO boat;
    try {
        boat = boatsClient.getBoat(boatDTO.getBoatId());
    } catch (FeignException e) {
        boat = boatsClient.addBoat(boatDTO);
    }
    var activity : Activity = activityService.addBoatToActivity(activityId, boatConverterService.toEntity(boat));
    return ResponseEntity.ok(activityConverterService.toDTO(activity));
}
```

This method adds a boat to an activity. If the boat could not be found in the database it also inserts this new boat.

We have introduced the following mutant:

```
/**
 * Adds a new boat to a given activity. If the given boat is not one
 * in the database it is also added on top of that.
 *
 * @param activityId ID of the activity.
 * @param boatDTO Boat DTO object to add.
 * @return The activity object after it has changed.
 */
1 usage 1 Stiliyan Nanovski *
@PostMapping("/add_boat_to_activity")
/PMD/
public ResponseEntity<ActivityDTO> addBoatToActivity(
    @Valid @NotNull @RequestParam UUID activityId,
    @Valid @NotNull @RequestBody BoatDTO boatDTO
) {
    BoatDTO boat;
    try {
        boat = boatsClient.getBoat(boatDTO.getBoatId());
    } catch (FeignException e) {
        boat = null;
    }
    var activity : Activity = activityService.addBoatToActivity(activityId, boatConverterService.toEntity(boat));
    return ResponseEntity.ok(activityConverterService.toDTO(activity));
}
```


This is a Constructor Call Mutation.

We have replaced the statement that adds our new boat with just a null assignment. The method will now not add the boat anymore to the database constituting a large bug.

In order to cover this mutant we added the following test case to our test suite:

```
@test
void addBoatToActivityNewBoat() {
    UUID activityId = UUID.randomUUID();
    UUID boatId = UUID.randomUUID();
    BoatDTO boatDTO = BoatDTO.builder()
        .boatId(boatId)
        .build();

    Boat boat = Boat.builder().id(boatId).build();
    when(boatsClient.getBoat(boatId)).thenReturn(new FeignException());
    when(boatsClient.addBoat(boatDTO)).thenReturn(boatDTO);

    when(boatConverterService.toEntity(boatDTO)).thenReturn(boat);

    activityController.addBoatToActivity(activityId, boatDTO);

    verify(boatsClient, times(wantedNumberOfInvocations: 1)).getBoat(boatId);
    verify(boatsClient, times(wantedNumberOfInvocations: 1)).addBoat(boatDTO);
    verify(activityService, times(wantedNumberOfInvocations: 1)).addBoatToActivity(activityId, boat);
}
```

This test now checks this functionality and the mutation is killed successfully.

The link to the commit which adds the test is the following:

<https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM33c/-/commit/ee75f102e3df5a5b465c3b841d88803d8f76ea1f>

BoatController

The BoatController class is responsible for managing boats in our system. The get_boats endpoint is responsible for getting all the boats available in the system and thus is crucial for creating activities that need to know what boats can be used.

We decided to introduce a mutant that simulates not using the boat converter service. And thus we've introduced a Null Returns mutant.

```
/**
 * The get all boats endpoint.
 * This endpoint gets all the boats.
 *
 * @return the list of boats.
 */
no usages  David Dinucu-Jianu
@GetMapping("/get_boats")
public ResponseEntity<List<BoatDTO>> getBoats() {
    var boats = boatService.getAllBoats()
        .stream().map(boatConverterService::toDTO).collect(Collectors.toList());
    return ResponseEntity.ok(boats);
}
```

Now, this is how the method looks.

```
@GetMapping("/get_boats")
public ResponseEntity<List<BoatDTO>> getBoats() {
    var boats = boatService.getAllBoats().stream() Stream<Boat>
        .map(b -> (BoatDTO) null) Stream<BoatDTO>
        .collect(Collectors.toList());
    return ResponseEntity.ok(boats);
}
```

And here are the tests. The mutant survived because we only check if the returned had the expected size.

```
@Test
void getBoats() {
    List<Boat> boatsInTheSystem = List.of(
        Boat.builder().id(UUID.randomUUID()).name("B1").availablePositions(List.of(BoatRole.Coach)).build(),
        Boat.builder().id(UUID.randomUUID()).name("B2").availablePositions(List.of()).build(),
        Boat.builder().id(UUID.randomUUID()).name("Another boat").availablePositions(
            List.of(BoatRole.PortSideRower, BoatRole.ScullingRower)).build()
    );
    for (Boat boat : boatsInTheSystem) {
        when(boatConverterService.toDTO(boat)).thenReturn(
            BoatDTO.builder().boatId(boat.getId()) BoatDTOBuilder
                .availablePositions(boat.getAvailablePositions())
                .name(boat.getName()).build() BoatDTO
                .withCoxCertificateId(boat.getCoxCertificateId())
        );
    }
    when(boatService.getAllBoats()).thenReturn(boatsInTheSystem);

    var response = boatController.getBoats();

    verify(boatService, atLeastOnce()).getAllBoats();
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(response.getBody()).hasSize( expected: 3);
}
```

In order to kill this mutant we should be checking that we got the boats that we expected to get. The following change will make sure the mutant is detected.

```
var response = boatController.getBoats();

verify(boatService, atLeastOnce()).getAllBoats();
assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
List<BoatDTO> expectedBoats = List.of(
    BoatDTO.builder().boatId(boatsInTheSystem.get(0).getId())
        .name("B1").availablePositions(List.of(BoatRole.Coach)).build(),
    BoatDTO.builder().boatId(boatsInTheSystem.get(1).getId())
        .name("B2").availablePositions(List.of()).build(),
    BoatDTO.builder().boatId(boatsInTheSystem.get(2).getId())
        .name("Another boat").availablePositions(
            List.of(BoatRole.PortSideRower, BoatRole.ScullingRower)).build()
);
assertThat(response.getBody()).containsExactlyInAnyOrderElementsOf(expectedBoats);
```

Now our test case is stronger - it checks for more things and thus is able to detect the mutant.

Link to the commit:

https://gitlab.ewi.tudelft.nl/cse2115/2022-2023/SEM33c/-/merge_requests/69/diffs?commit_id=b72d8b6db4a127ce3f665344698db59d7937f08a