Norwegian University of Science and Technology

TDT4242 Advanced Software Engineering, spring 2022

Exercise 2

Group Number: 22
Peer Group Number: 41

Trym Grande Hans Kristian Granli

June 1, 2022

1 Introduction

For this exercise we have implemented the feature requirements outlined in exercise 1, with the exception of FR8 which we were allowed to change. We have also created backend tests for UserSerializer, and Workout permissions in the aforementioned secfit project.

After implementing the features and backend tests we created black-box tests using the Selenium framework to test the user-registration page, exercise registration as well as workout registration, and our new file data-file uploader. We got approval by the TA to skip Selenium test of FR1 & FR2 regarding reset password.

We developed the CI/CD pipeline to automatically update the changes on the project on a heroku host: `http://secfit-22-frontend-ntnu.herokuapp.com/`

2 Implementation of the new features by using DevOps environment

2.1 DevOps

We decided to use gitlab-ci configuration included in the base project. Our only changes to the gitlab-config were the heroku project name.

The configuration has two stages: test and staging. The only test job outlined in the test-stage perfoms the django tests. If the django tests are all successful pipeline continues to the staging-stage. This stage will push the changes to the remote heroku host, utilizing secure api keys defined in the gitlab project. The pipeline will only run on changes made to the master-branch. This means that the heroku-frontend will always host an instance of the code in the project, as long as the project passes all the defined tests.

2.2 New Features

As mentioned in the introduction,with approval from the TA, we made a change to the feature requirements outlined in our FR8 under exercise 1. This was changed from parsing workout data to edit workout data. Our feature requirements for this exercise was thus:

In broad terms these feature requirements will enable the user to:

- Reset a forgotten password using email
- Like and receive likes on their workouts

FR1	Generate password reset link
FR2	Reset user's password
FR3	Like a workout
FR4	Remove like from workout
FR5	Upload file to server
FR6	Link between workout and file
FR7	Download file related to workout
FR8	Edit workout file

Table 1: Feature Requirements

- Upload workout data. This can be any type of file.

2.3 Design & Implementation

For the **reset password** feature, we decided to use built-in methods provided by django. This reduces the amount of coding we had to do on both front- and backend.

Workout Like

This feature had to be implemented as a new django class. It has both user and the related workout as foreign keys. We also created a new api endpoint so that the requesting client only got the likes if it wanted to.

The frontend was extended with two new buttons: like and remove-like. These can be found on the workout page.

Upload file

This feature required a lot of work. Luckliy we were able to re-use a lot of code from the WorkoutGallery, both on frond and backend. The visual design of workout-data is the same/similar to workout-gallery where it makes sense.

The biggest challenge was altering the json parser. The parser was already altered for the gallery feature to work, but we had to extend it further to allow for multiple fields of binary data.

2.4 Using DevOps to develop new features

When developing new features in a DevOps environment it is possible to define wanted behaviour before writing the code. For instance the workout-like feature wants to know the number of likes given to a specific workout. This makes for an easy but effective test outline. We had no experience working with django before this project, so we had to do a lot of trial and error, in which the testing tools were of great help.

Additionally the selenium framework was something we had no previous experience with, but we both plan on using in the future for our web projects. The black-box tests allowed us to skip tedious tasks such as registering a new user, registering workout and uploading files.

2.5 Meeting with peer group

Unfortunately both members of our peer group (21) from exercise 1 have dropped the course. We therefore had to get a new peer group, which we didn't get until very close to the assignment deadline. This meant that we didn't get to have more than one meeting about the features developed, and the vision of the previous group has been lost.

For the meeting both of us and all members (one) from the peer group attended.

3 Automated test scripts to get full statement coverage

The test scripts can be found in:

- backend/secfit/users/tests.py
- backend/secfit/workouts/tests.py

Note that in our final delivery we have flipped the logic of the password-validation-test. We did this to allow the testing-stage in the gitlab-pipeline to succeed, as we weren't supposed to fix any bugs we encountered. Outside of the requirements of the task at hand workouts/tests.py also contains some tests related to workout-like and workout-data.

The biggest takeaway from the testing-task was that the password validation didn't work as intended. We did not find any bugs in the workouts app from our tests.

The tests can be ran by issuing the command: python3 manage.py test

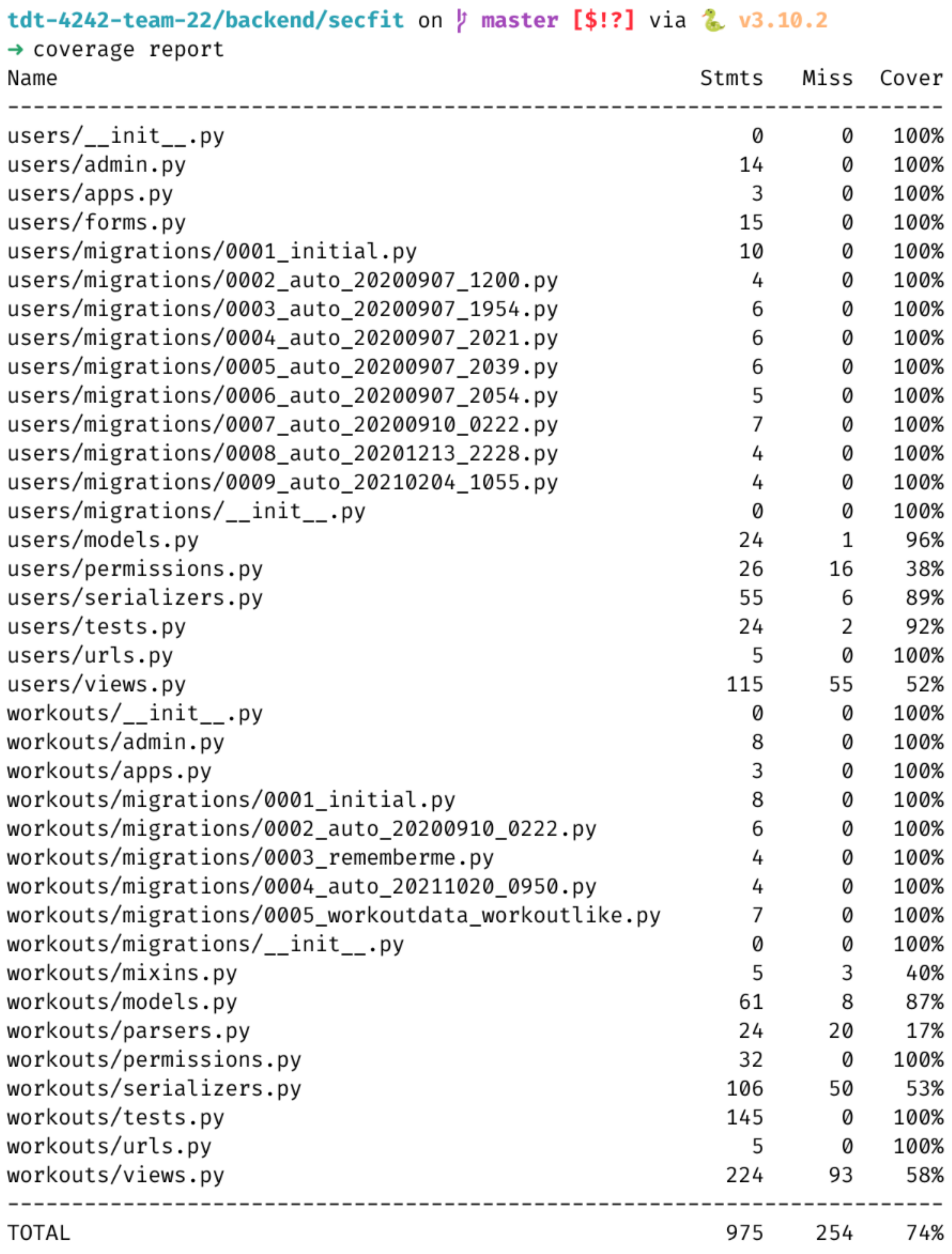
3.1 Test Coverage Report

In order to generate a coverage report we utilized the python package coverage:

```
$ coverage run --source="workouts,users" manage.py test
```

```
$ coverage report
```

The output was then:



```
tdt-4242-team-22/backend/secfit on  master [$!?] via 🐍 v3.10.2
➜ coverage report
Name                                                      Stmts   Miss  Cover
-----------------------------------------------------------------------------
users/__init__.py                                             0      0   100%
users/admin.py                                               14      0   100%
users/apps.py                                                 3      0   100%
users/forms.py                                               15      0   100%
users/migrations/0001_initial.py                             10      0   100%
users/migrations/0002_auto_20200907_1200.py                   4      0   100%
users/migrations/0003_auto_20200907_1954.py                   6      0   100%
users/migrations/0004_auto_20200907_2021.py                   6      0   100%
users/migrations/0005_auto_20200907_2039.py                   6      0   100%
users/migrations/0006_auto_20200907_2054.py                   5      0   100%
users/migrations/0007_auto_20200910_0222.py                   7      0   100%
users/migrations/0008_auto_20201213_2228.py                   4      0   100%
users/migrations/0009_auto_20210204_1055.py                   4      0   100%
users/migrations/__init__.py                                  0      0   100%
users/models.py                                              24      1    96%
users/permissions.py                                         26     16    38%
users/serializers.py                                         55      6    89%
users/tests.py                                               24      2    92%
users/urls.py                                                 5      0   100%
users/views.py                                              115     55    52%
workouts/__init__.py                                          0      0   100%
workouts/admin.py                                             8      0   100%
workouts/apps.py                                              3      0   100%
workouts/migrations/0001_initial.py                           8      0   100%
workouts/migrations/0002_auto_20200910_0222.py                6      0   100%
workouts/migrations/0003_rememberme.py                        4      0   100%
workouts/migrations/0004_auto_20211020_0950.py                4      0   100%
workouts/migrations/0005_workoutdata_workoutlike.py           7      0   100%
workouts/migrations/__init__.py                               0      0   100%
workouts/mixins.py                                            5      3    40%
workouts/models.py                                           61      8    87%
workouts/parsers.py                                          24     20    17%
workouts/permissions.py                                      32      0   100%
workouts/serializers.py                                     106     50    53%
workouts/tests.py                                           145      0   100%
workouts/urls.py                                              5      0   100%
workouts/views.py                                           224     93    58%
-----------------------------------------------------------------------------
TOTAL                                                       975    254    74%
```

Figure 1: Coverage report

4 Automated test scripts for black-box testing

We decided to use Selenium to perform the black-box test. This was because selenium has support for python-scripting, which makes the test easy to get up and running. The selenium-test can be found in selenium/seleniumtest.py. This script requires the project to be running. If the project isn't running on http://localhost:9022. The script can be pointed to a different address by running:

```
$ python3 seleniumtest.py http://<url>
```

or just

```
$ python3 seleniumtest.py
```

if the default url is used.

4.1 Results

By creating and running the black-box test we discovered that the project's password validation is lackluster. This is something we knew was an issue from writing the backend-tets, but the frontend should still try to validate the data input. The only constraint we found was that the password field must be filled with "something" and the email can be empty (which we think is an issue). In other words entering two different passwords is not an issue according to the front-end, which we consider a failed test.

5 Acceptance tests

Test Case Name	View milestone status
Test Case ID	TC-29
Description	Coach is able to view if a milestone has been completed or not for a given time period
Preconditions needed to run this test case	1. Coach is registered and logged in
	2. Athlete have been given milestone
Event Sequence	
Input events (performed by tester)	*Output events (observed by system tester)*
	1. Screen shows milestone status stating whether it is achieved or not

Table 2: Test Case 1 - View milestone status

Create new milestone with negative target goal on first try	
Test Case ID	TC-30
Description	An athlete registers a new milestone with negative target goal on first try
Preconditions needed to run this test case	1. The target goal must be positive
	2. Milestone creation screen is active
Event Sequence	
Input events (performed by tester)	*Output events (observed by system tester)*
	1. Milestone creation page is active
2. Enter -5 target	
	3. Screen shows -5 target
4. User hits enter	
	5. Screen error
6. User fixes mistake by setting target 5	
	7. Screen shows target 5
8. User hits enter	
	9. Screen shows newly created milestone status

Table 3: Test Case 2 - Create new milestone, negative input on first try

Test Case Name	Delete milestone
Test Case ID	TC-31
Description	An athlete is able to delete his own milestone
Preconditions needed to run this test case	1. Athlete is registered and logged in
	2. Athlete already has a milestone and is on the milestone-view page
Event Sequence	
Input events (performed by tester)	*Output events (observed by system tester)*
	1. Screen shows the athlete's milestones
2. User clicks the milestone they want to delete	
	3. Screen shows detailed information about the milestone
4. User clicks the delete button	
	5. Screen shows milestone overview, and the deleted milestone is gone

Table 4: Test Case 31 - Delete milestone

Test Case Name	Edit milestone
Test Case ID	TC-32
Description	An athlete is able to edit his own milestone
Preconditions needed to run this test case	1. Athlete is registered and logged in
	2. Athlete already has a milestone
Event Sequence	
Input events (performed by tester)	*Output events (observed by system tester)*
	1. Screen milestone overview page
2. User clicks the milestone they want changed	
	3. Screen shows detailed overview of the milestone with an edit button
4. User clicks the edit button	
	5. Screen shows a milestone-edit page
6. User edits the desired fields and clicks save	
	7. Screen shows detailed milestone view with updated information

Table 5: Test Case 32 - Edit milestone

Test Case Name	Coach notification on milestone creation
Test Case ID	TC-33
Description	An athlete's coach gets notified whenever the athlete creates a milestone
Preconditions needed to run this test case	1. A coach is registered and logged in
	2. The coach is assigned to an athlete
Event Sequence	
Input events (performed by tester)	*Output events (observed by system tester)*
	1. Athlete's screen shows 'create new milestone' button
2. Athlete creates new milestone using the button	
	3. Coach receives notification stating that the athlete has created a new milestone

Table 6: Test Case 5 - Notify coach on milestone creation

Test Case Name	Coach overview
Test Case ID	TC-34
Description	A coach can see an overview of athletes they coach that shows progression for milestones for each athlete
Preconditions needed to run this test case	1. A coach is registered and logged in
	2. One or more athletes are assigned to the coach and the athletes have milestones
Event Sequence	
Input events (performed by tester)	*Output events (observed by system tester)*
	1. Screen shows home page
2. User clicks on milestones button	
	3. Screen shows milestone overview for the user
4. User clicks coach-overview-button	
	5. Screen shows overview of the user's athletes and their milestones

Table 7: Test Case 34 - Coach milestone overview

6 Data flow test

6.1 Description

These results are from a *manual* test of ”all uses (AU)” of the retrieveWorkoutImages function in the gallery.js file, found in /frontend/www/scripts. Due to the lack of documentation and general support for data flow testing, we have decided to implement the test manually by hand after failing to do so as intended. In addition, there was a published message on one of the blackboard threads stating that the testing could be done like this if needed. All uses (AU) includes all predicate use (APU) and all computational use (ACU) of variables inside the given function. This is a type of ”white box testing”. As opposed to ”black box testing”, where the code is considered invisible to the tester, white box testing allows the tester to see how the code is working ”inside the box”. This has allowed us to draw out a detailed map of how the variables ”flow” through the execution of the code inside the function being tested.

The flow chart along with a textual description can be found in /frontend/AU test/.

6.2 Results

Through the test, we were able to find the following properties: - We can observe that **almost all variables that have been used, also have been declared within the same function**, as none of the ”declaration line” fields are missing. The only exception found is the 'HOST' variable. - We can also observe that **all declared variables have been used**, as none of the ”use line” fields are missing. - This was then further explored in a manually generated flow chart diagram by seeing that **there exists a path from declaration to use for all variables**.