LYNGBY EVENING SCHOOL

Group 2

GitHub project
https://github.com/DavidCarl/test2019-exam

David Alves
David Carl
Elitsa Marinovska
Tjalfe Møller

Table of Contents

Team collaboration

Our team, composed by four elements, within none of which is a native English speaker, managed to communicate without any major misunderstandings. The team or group #2, was working both together and remotely, polishing all the potential blurred specifications by initiating group discussions where everyone had the chance of expressing their opinion and bringing new ideas to the table. In accordance to our programming equipment (IDEs, drawings, pseudo code, etc.) we have accommodated a Kanban board with the basic functionalities, needed for a successful project management. There we were recording the user stories extracted from the project business case and the meetings with the “product owner” in a very detailed manner. Having the convenient board integration with Github in place, the team members could closely follow the progress of each of the personally assigned tasks. Additionally, before digging into the project corresponding tasks of the day, on every meeting the team members were giving updates to each other on how far has the picked-up task reach towards its completion goal.

One of the key impediments was the fact that the team couldn’t find a frequent common ground regarding schedule between the group playing the role of a product owner and us, however our team always found means of quickly reaching in an agreement whenever some specifications weren’t clearly stated.

Once work has been completed, the team has established a solid workflow which was demanding (an) independent member(s) review which if resulting in an approval, was successfully merged into the main development branch, of course after checking the automated CI pipeline status. If some suggestions arised, those were documented appropriately with some pointers on how the implementation could’ve been approved.

Test automation strategy

The team has relied on the services of CircleCI whenever it came to verify that all the changes requiring some modifications into the main development branch weren't causing any unexpected issues amongst the already proven to work existing implementation. This technology has made the process of comparing the pending Pull Request changes against the master branch fast and convenient. After conducting a personal/group review, the green light from CircleCI was signaling that no obstacles were on the way. This Pull Request method has already become a mark on our everyday work, however to diminish any innocent misstep of pushing into the master branch by mistake, the team decided to continue using the benefits of CircleCI, more specifically, we disabled the ability to push to the master branch. This way any changes done by a project-involved party forces a manual review to be initiated.

The group has also implemented the process of using static tools for verifying if the common code standards and common practices were in place. The team benefitted from those by having automated code checks based on the techniques we deemed important.

Non-functional requirements

Unfortunately this isn't something we have had our main focus at. However, if we had to make a plan on how we would approach that,we should have implemented something like a Jmeter plan to test the performance of our web application - this is everything from stability to our max user count.

An important step in the testing process is to ensure a good quality for the end users who will use our site since it is of great importance that it responds quickly independent of the load, as a study shows that, users feel disrupted after 15 seconds[1]of waiting time. That research is over 20 years ago and everything has only positively evolved in speed, while our patience has decreased with it. Additionally, according to the relatively recent research (at the time of writing) in this website[2], modern users dislike websites with a response time above 3 seconds. We all know this feeling that if a website is too slow we are more likely to find a replacement for our needs, even though it is only a few seconds we talk about.
This is why both our max user capacity and our stability is important to establish tests at, since hitting our max limit of concurrent users is going to slow the site down to undesired speeds, and we might lose the evening school possible customers.

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.99.2770&rep=rep1&type=pdf
[2] https://hostingtribunal.com/blog/how-speed-affects-website/

Testability

We would claim that our program is testable, due to the fact that it has been designed from the button up with testing approach in mind, so we have tried our best to test as much of its aspects as possible, as well as applying different types of tests on it to use the various benefits of the different techniques. We tried to test everything as close to a method level as possible, so in the case that we can test the method itself, that's what we did. This gave us a good overview on what's changed if a method fails.

We tried to keep our methods as small as possible and only made them do certain things, so that way we could ensure a higher testability percentage and produce better and more accurate results by doing it.

An example of ensuring a high testability is by performing TDD. Even though we didn't only do TDD, a majority of our implementation went over this pattern and we could clearly notice all the good benefits that come with this practice.

When we are talking about testability we think it could be beneficial for the reader to find some stats from the project, therefore, the next section is dedicated exactly for that.
We have a high line coverage (in the files where it matters) which is between 75% and 100% on our backend code. Some of the lines that haven't been tested are methods like toString, etc. that we deemed unnecessary to test. We also missed testing on some implementations of interfaces - that is our database layer. Our APIs have a code coverage between 78% and 100% and a singleton that has 0% line coverage. We feel that this is a pretty good overall line coverage and we also think that this is above the average case since many of our files have a coverage above 80%, close to 90%. A high line coverage didn't automatically make us believe that a test is properly implemented, but this is definitely something we had in mind.

Test Design Techniques

White box testing:

We tried our best to comply with recommended white box testing practices as we went along our code spree. This can also be seen in the previous paragraph about our high line coverage, as we tried to test both successful and failing cases. Along the way of the project white box testing has proven useful, since we found bugs we did not expect after changing bits in the method. Even though we already knew the importance of testing, in the presence of such cases, it is still another good example of why it is important to have white box testing.

Black box testing:

The closest to black box testing we got is our Selenium tests on our front-end. With their help we test how our application is handling different kind of user inputs in an automated manner. If we take a look at our 'TeacherSignupTest.java' we have 4 selenium test to test the signup process of a teacher. We made a decision table to represent this test case and we placed it below.

This table is made to show how our test works on a teacher sign up event in the front-end with Selenium.

Decision table:	Test case #1	Test case #2	Test case #3	Test case #4
All fields filled	Y	N	N	N
Missing email	N	Y	N	N
Missing name	N	N	Y	N
Missing education	N	N	N	Y
Result:	Test case #1	Test case #2	Test case #3	Test case #4
Account created	Y	N	N	N
Email missing	N	Y	N	N
Name missing	N	N	Y	N
Education missing	N	N	N	Y

Our decision table could probably be expanded to be more accurate, so it would look like this:

Decision table:	Test case #1	Test case #2	Test case #3	Test case #4	Test case #5	Test case #6	Test case #7	Test case #8
All fields filled	Y	N	N	N	N	N	N	N
Missing email	N	Y	N	N	N	Y	Y	Y
Missing name	N	N	Y	N	Y	Y	Y	N
Missing education	N	N	N	Y	Y	Y	N	Y
Result:	Test case #1	Test case #2	Test case #3	Test case #4	Test case #5	Test case #6	Test case #7	Test case #8
Account created	Y	N	N	N	N	N	N	N
Email missing	N	Y	N	N	N	Y	Y	Y
Name missing	N	N	Y	N	Y	Y	Y	N
Education missing	N	N	N	Y	Y	Y	N	Y

Boundary Value Analysis:

We did perform Boundary Value Analysis in some places. An example of this is our Teacher back-end class. A teacher has previously taught semester hours (how many hours have they taught during their last semester). This is a value that you can set by using a setter, but since it's not possible to teach negative hours, the lowest hours value can be set to 0. In the test file we then tried to set it to -10 and -20, which after checking what is the held value, it should and was set to 0.

Another case, mostly in the same area is when we need to get the teachers' eligibility status for voting on the next semester (what we need is the amount of hours they worked the previous semester). Here we test what boolean we get back if they've worked 19 and 5 hours for example and both of these cases should return false. We also test what happens if they've worked 20 and 21 hours and in this case it should and returned true.

Conclusion

This project provided us with the ability to experiment and acquire new competences in the field of testing by putting in a new context all the knowledge that we were given throughout the semester. Having a greater time frame for the project, much more testing aspects would have gladly been included because testing shouldn’t be underestimated in any way.