TEST-SEMESTER-RAPORT

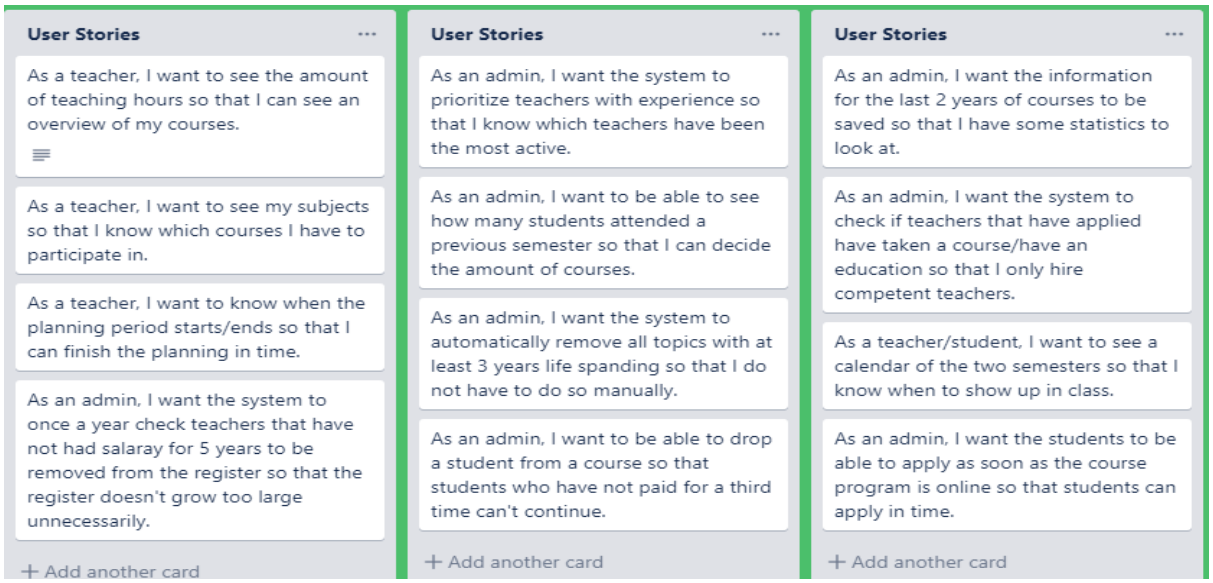
By Group 4 - Murched Kayed, Hallur vid Neyst & Zaeem Shafiq

MAY 23, 2019

COPENHAGEN BUSINESS ACADEMY

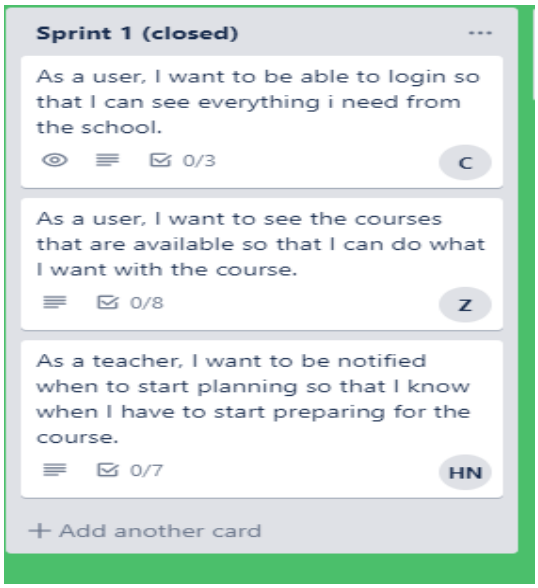
Team collaboration

We started with reading the assignment carefully, then making a brainstorm and discussing if we understand the needed requirements. After that we agreed that we are going to work in scrum which means we will work in sprints, so we used trello for creating user stories depending on the description of the assignment.
that how our backlog looks like after creating the user stories and dealt them into tasks:



We made sure that all the user stories are independent, which it will make the user stories have a good test-manner face, that would be an advantage since it will make it easier to test each one of them independently.

.



After creating the user stories and dealing them into tasks, we start the sprint one by adding 3 user stories that gives good meaning to start with, so every group member can have user story to work on. We decided also that all the team should work in TDD test-manner which means creating the test's before implementing the code(logic).

We agreed that it would be good to start with creating the login since there is 3 different roles (users) that's going to use the system, which means the system should send contains 3 different pages/servlets for each user, after creating that so it will be much easier to deal the user stories.

We decided and agreed to use servlets and jsp since we don't have so much time to implement it using the 3 lags structure (frontend, backend, database), that's why it is a advantage to use servlets and jsp, since it contains backend and the frontend in a one project. We also agreed that our focus should be most on testing the software.

We used discord and anyDesk for the online meetings, in the same time we had also some liv meetings in the school, we used GitHub to deal the project with each other, and we agreed that for every user story should we create a branch to, we agreed also the before creating pull request to merge with the master, that we should be sure that the code change has not adversely affected existing features by using regression testing technique.

Test automation strategy

The following criteria were considered during our decision on what to apply so our automation (Automation is the technology by which a process or procedure is performed with minimal human assistance) is effective and efficient:

- We want to make our test automation strategy result-oriented

  We felt it was important that we could understand and define the objective of the automation strategy. This was done by using a tool called jacoco coverage and also by taking a look at our acceptance tests on Trello. Jacoco would tell us which methods we were missing, while the acceptance tests could determine if the missing test areas were necessary to implement.

- We want to work with a suitable test automation tool. (selenium, Travis)

  Selenium is made precisely for the purpose of improving the test automation strategy. We really liked the idea of it being able to perform without human assistance, whereas without it we would have to manually perform in order to test the web pages etc. We also liked the idea of Travis, because for one it was a requirement of the assignment, but also it gives up the setup needed for CI/CD. This assured that we have a working project on our GitHub repository where every time changes are being made, then the Travis will run the build and tell if the build/needed tests are successful.

- Are our test scripts reusable?

  We wanted to make sure that our tests are all reusable, and that they are in no way non-repeatable, and do not defect the system and/or the production database. This was being done by using a test database each time we needed it, and then also clean it either before or after the test.

Non-functional requirements

We used usability testing in a way to see how easy to use something is by testing it with real users. We could use JMeter and blaze meter, as a non-functional testing so we can test how the software performance, but we did not have enough time doing. We could also use non-functional security testing as will to our login, but as said we did not have enough time for that.

Testability

- **Observability**
  - While writing our methods, we had observability in mind to make it possible to see where it went wrong if a method crash. In our case we did this by “printing” to detect where the problem is, we used that most in the if statements so to know if we even come in the statement.
- **Controllability**
  - We reduced the amount of dependencies by ensuring that every class had a constructor. This separated the application logic from the instantiation logic and made it possible for us to make us of the dependency injection when testing. Now if we instantiate in our test classes, it should then have its own act and therefore never interfere with the system.
- **Smallness**
  - All our tests meet the criteria of being in the “smallness”. In terms of the size of the project, we have reduced the amount of lines of code as much as possible, so that there is less to test. When it comes to the actual tests, we also want them to have a “singularity”, meaning that the instantiations used should be as little as possible, which also means that the test should focus on one feature at a time.

Test design techniques

Among other things, we have chosen to make use of Specification-Based/Black-Box techniques in the form of Equivalence Partitioning (EP), Boundary Value Analysis (BVA) and Use Case Testing. This has proved to be extremely beneficial. These techniques helped us, among other things, to reduce redundant test cases, achieve a very high-test coverage and structuring the test cases. We also used white box testing technique to test the software solutions internal structure and cod.

Equivalence Partitioning (EP) was used to make sure that the test input data was partitioned into a number of classes having an equivalent number of data. The test cases were then designed for each class/partition. This helped us to reduce the number of test cases.

Boundary Value Analysis (BVA) was used to explore errors at the boundary of the input domain. It is observed that in most applications, errors occur at the boundary values. We used Boundary Value Analysis as the next part of Equivalence Partitioning where we selected test cases at the borders of the equivalence classes. This helped us a lot to catch any input errors that might interrupt with the proper functioning of our program.

Use Case Testing was used to make sure that the test cases were designed to execute different business scenarios and end-user functionalities. We made different

use cases containing a description of a particular use of the system by a user. Use case testing helped us a lot identifying test cases that cover the entire system.

Code coverage technique we used Jacoco coverage to measure how many lines of our code are tested.
Jacoco coverage was useful for us, because we knew what we need to test and what we had already test.

$$\text{Coverage} = \frac{\text{Number of coverage items exercised}}{\text{Total number of coverage items}} \times 100\%$$