Test Report

Team 2

I. JUnit resources tests (back-end testing)

Our application has 9 resource classes. Each of these resources has a number of methods, most of which are covered with the unit tests located in the *test* folder (*test.JerseyClientLiveTest*). Tested are the following methods:

- HTML-producing methods. The tests check the response status code, the response content type as well as the content length.
- JSON-producing methods. The tests check the response status code, the response content type as well as the actual JSON content of the response.
- database-invoking methods. The tests check the response status code, the response content type and the actual content of the response. The data received is checked, then changed with the method in the database, then pulled again to verify that the change was successful and after that the default value is inserted back again.

II. Selenium tests (front-end testing)

We use Selenium to test the interaction with the webpage from client's perspective. The purpose of these tests is to get an error-free performance from the webpage while the tests are being run with either of the following methods:

- Selenium IDE: we prepared .side file which is a Selenium project. To use it, install Selenium IDE, insert this file into IDE and run all tests. This is a preferred way to run our Selenium test cases.
- JUnit: this version of the same Selenium test set as in .side file can be run in Java. However, *selenium-server-jar* is intended to be run as a standalone application and contains its own servlet container for this purpose. As this jar interferes with the normal work of the project's servlet and prevents it from running, *test.selemium.SeleniumTest.java* and *org.seleniumhq.selenium* dependency in pom.xml are commented out. To run the tests, these files need to be uncommented. To run the application, the test file and dependency need to be commented out again.

Our Selenium tests cover a variety of use cases, ranging from the registration page availability and validity check and up to the interaction with the actual dashboard.

III. Manual testing

We also tested the application manually by covering the user interaction flows, in particular, we registered a new account, upgraded user’s status to premium, checked filter’s work, verified all the functionality of the dashboard, uploaded new profile pictures, reset passwords, experimented with valid and invalid tokens.