Exam for PG5100

8am Monday 17th October – 8am Wednesday 19th October

This assignment is worth 100% of your final grade for PG5100. The assignment has to be submitted on Its Learning by Wednesday 19th of October morning at 8:00am, under the “exam” folder. Link at:

https://woact.itslearning.com/ContentArea/ContentArea.aspx?LocationID=698&LocationType=1&ElementID=18119

The exam assignment will have to be zipped in a zip file with name pg5100_<id>.zip, where you should replace <id> with your own student id, eg pg5100_123456.zip. No “rar”, no “tar.gz”, etc. You need to submit all source codes (eg., .java, .xhtml and .xml), and no compiled code (.class) or IDE files (.idea). Once I got a 175MB rar file as a submission, containing everything, compiled code included… you can guess what grade the student got… an F.

The delivered project should be compilable with Maven 3.x with commands like “mvn install -DskipTests” directly from your unzipped file. Compilation failures will *heavily* reduce your grade.

The assignment is divided in several parts/exercises. The parts are incremental, ie building on each other, for a total of 100 points.

Note: during the exam period, I will NOT answer to any email. However, I will answer questions on the Its Learning forum of the course. I will answer questions regarding possible misunderstanding in the exam's instructions, or more general questions like “can we use library X?”. Questions like “How can we do Y?” will be of course left unanswered… At any rate, during the exam you should check the discussion forum often, as I might post some clarifications or other messages of interest.

You can (and should when appropriate) reuse code from https://github.com/arcuri82/pg5100_autumn_2016, for example the pom files. You will also notice that the exam has many similarities with the exercises you have done during the course (and if you wondered why I did not show the solutions to those exercises before, now you have your answer...). You are allowed to re-use / adapt your own code, but of course not the one from other students...

The source code of the solution (the one used to create the screenshots) will likely be discussed in the first class of PG6100, on Thursday the 20th.

Easy ways to get a straight F with no appeal:

- have production code (not the test one) that is exploitable by SQL injection
- submit a solution with no test at all, even if it is working
- submit your delivery as a rar file instead of a zip (yes, I do hate rar files)

The exam should NOT be done in group: each student has to write the project on his/her own. During the exam period, you are not allowed to discuss any part of this exam with any other student or person. The only exception is questions to me in the PG5100 discussion forum, as discussed

earlier. Failure to comply to these rules will result in an F grade and possible further disciplinary actions.

Note: this exam is deliberately **long** and **complex**. This is to distinguish the As from the Bs. I expect only very few of you can complete all of it, if any at all. A good grade (eg C or B) can still be achieved even if few parts are missing. However, if you have done the exercises throughout the course, it should be much easier.

The MyNews Application

In this exam, you will need to create a basic web application called “My News” (the actual name does not really matter…). In short, people can post news, and other users can write comments on those. In the rest of this text, I will use the terms “post” and “news” interchangeably.

The application should be implemented on the JEE stack, eg JPA, JTA, EJB, and JSF, as shown in class. For this exam, you do NOT need to write any CSS or JavaScript, although you will have to edit some HTML.

Note: when I show screenshots, it is only to clarify the text. You do not have to perfectly 100% match the layout/size of the widgets.

The project should be structured in 3 Maven submodules, in the same way as in the jsf/jacoco and exam example modules shown in class: “backend” (containing Entity, EJB, and all other needed classes), “frontend” (JSF beans and XHTML) and “report” (for aggregated JaCoCo report). Note, if you copy and paste those pom files, you will have to modify the root pom file of your project to be self-contained (ie no pointing to any parent).

You should use Maven to compile a WAR file that should be deployable on WildFly 10. You need to configure Maven to be able to start WildFly and automatically deploy the WAR when running “mvn wildfly:run” from the “frontend” folder (after doing a “mvn install” on the root folder). You have to use an embedded database (eg H2).

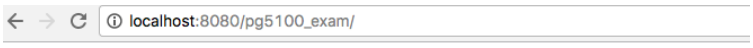
Testing should be based on Arquillian (for EJB) and Selenium (for end-to-end) using Chrome. You can make the assumption that the Chrome drivers are available under the user's home folder (as done in class and in the PG5100 Git repository). All tests should be automatically started (and connecting to WildFly if necessary) when running “mvn clean install -fae” from the root folder. Tests should be independent, ie they should not fail based on the order in which they are run, and neither they should fail if run more than once (e.g., when run several times from IDE without restarting WildFly).

You need to provide a “readme.txt” file (in same folder as the root pom.xml) where you *briefly* discuss your solution: eg, how you structured the code (eg, “@Entity classes are in package X”), and any other important info you want to provide.

If you do not attempt to do some of the exercises, state so in the “readme.txt” file (this will make me correct the exam much faster), eg “I did not do exercises X, Y and Z”.

(E1, 5pt) Base Page

After starting WildFly, the developed application should be accessible at "localhost:8080/pg5100_exam". The default index.html page should have two links, one to a page for the theory questions ("theory.html"), and one for the MyNews application. For example:



PG5100 Exam Autumn 2016

- Link to theory questions/answers
- Link to MyNews

(E2, 20pt) Theory

In the "theory.html" file, you need to answer the following questions (you might want to repeat the text of the questions in that HTML file). Note: because you are doing this exercise at home, you can of course read books, internet (eg, StackOverflow), etc. However, you can "read" but are not allowed to ask questions on forums.

1) Explain the main differences between JPQL and SQL, and why one might rather want to use JPQL when developing JEE programs.

2) In the context of JPA, explain the main differences between a "pessimistic" and an "optimistic" lock, and when one might choose to use one of them instead of the other.

3) In the context of EJB, explain what "Dependency Injection" is, how it works, and why it is useful.

4) In the context of EJB, explain the concept of "Proxy class", how it works, and why it is useful.

5) In the context of JSF, explain the differences and relation between the XHTML files in the "webapp" folder and the actual HTML files rendered on the browser when one accesses those files via a JSF servlet.

6) In a Maven project, explain the steps required to configure automated testing with Selenium, as done in class (eg., what needs to be configured, which Maven phases are involved, why the tests have to be written as integration instead of unit tests, how to automatically get and install a JEE container, etc).

Following screenshot show an example of theory.html page, where of course you need to replace "TODO answer here" with the actual answers… Furthermore, if I updated the text of the above questions, I might not have updated the screenshot: the text above is the one that matters.

Theory Questions/Answers

---

1) Explain the main differences between JPQL and SQL, and why one might rather want to use JPQL when developing JEE programs.
TODO answer here

---

2) In the context of JPA, explain the main differences between a "pessimistic" and an "optimistic" lock, and when one might choose to use one of them instead of the other.
TODO answer here

---

3) In the context of EJB, explain what "Dependency Injection" is, how it works, and why it is useful.
TODO answer here

---

4) In the context of EJB, explain the concept of "Proxy class", how it works, and why it is useful.
TODO answer here

---

5) In the context of JSF, explain the differences and relation between the XHTML files in the "webapp" folder and the actual HTML files rendered on the browser when one accesses those files via a JSF servlet.
TODO answer here

---

6) In a Maven project, explain the steps required to configure automated testing with Selenium, as done in class (eg., what needs to be configured, which Maven phases are involved, why the tests have to be written as integration instead of unit tests, how to automatically get and install a JEE container, etc).
TODO answer here

(E3, 10pt) Backend

In the backend module you need to have 3 entities: User, Post and Comment.

- *User* should have the fields representing concepts like: userId, hash, salt, firstName, middleName, lastName, registrationTime
- *Post* should have a text, the time when it was created, information on who created it. Users can add comments to posts, and vote for/against them (eg, +1 or -1).
- *Comment*: should extend *Post,* and add functionality like the boolean possibility of being “moderated” (in the GUI, a moderated comment will have its text not displayed, and replaced with a warning message)

Add adequate/reasonable constraints to all the fields in those entities.

Write EJBs to be able to achieve the following functionality:

- create a user

- login a user
- create a post
- vote for/against a post/comment, and also undo the voting. A user cannot give more than a +1 to a same post, as well as no more than a -1.
- get all posts sorted by time (most recents first)
- get all posts sorted by voting (highest scores first)
- create a comment for a given post
- moderate (ie set to true its flag) a comment. Only the author of the post is allowed to moderate its comments.
- calculate the “karma” of a user: this is calculated by the sum of all the scores of all of his/her posts and comments. Each moderated comment gives an extra “-10” penalty.

(E4, 10pt) Arquillian Tests

Write Arquillian tests for the EJBs, in two different files, called *UserEJBTest* and *PostEJBTest.*

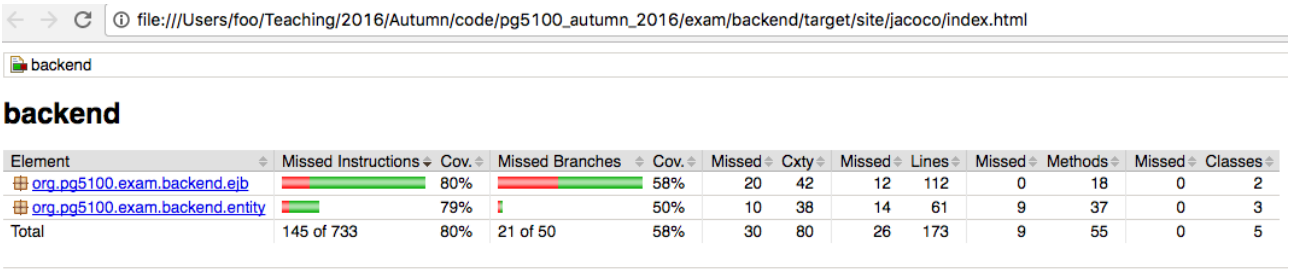
In *PostEJBTest* you need (in this order) *at least* the following. Names should be hopefully self-explanatory… the actual content is up to you, as long as it is somehow related to the name of the test:

- testCreatePost
- testVoteFor
- testVoteAgainst
- testCannotVoteForTwice
- testCannotVoteAgainstTwice
- testUnvote
- testChangeVote
- testGetAllPostByTime
- testGetAllPostByScore
- testCreateComment
- testModerateOwn
- testFailModerateOther
- testVoteForComment

In UserEJBTest you need *at least*:

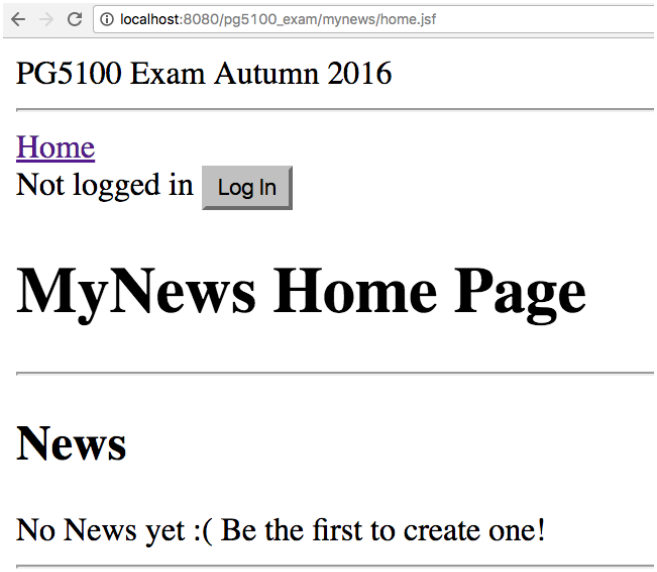
- testKarmaWithModeration
  - create a user, a post, and 4 comments for that post
  - downvote the post
  - upvote 2 comments
  - moderate the other 2 comments
  - create a new user, which will upvote one of the non-moderated comments
  - verify that the “karma” of the first user is “-1 + 2 + (-20) + 1 = -18”

When all these tests run from Maven, you should achieve a JaCoCo instruction coverage of at least 50% (see under “backend/target/site/jacoco/index.html”). For example:



(E5, 5pt) Home Page

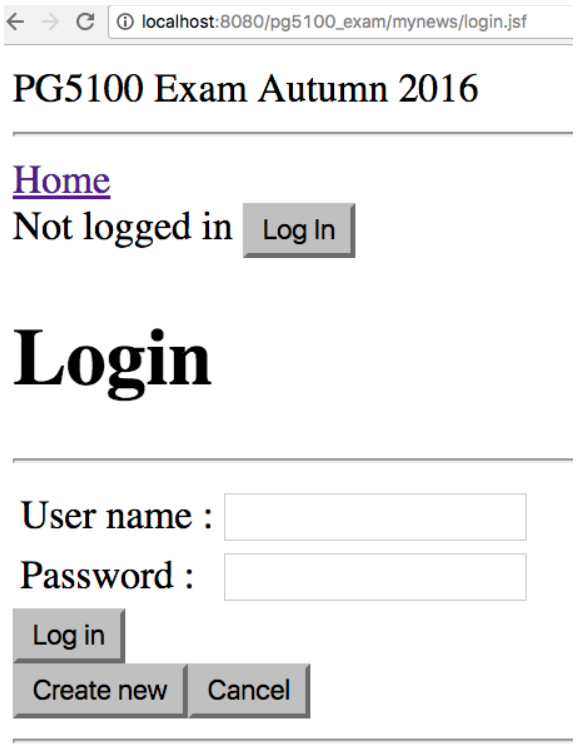
The first time you open it, the home page of MyNews should look like:

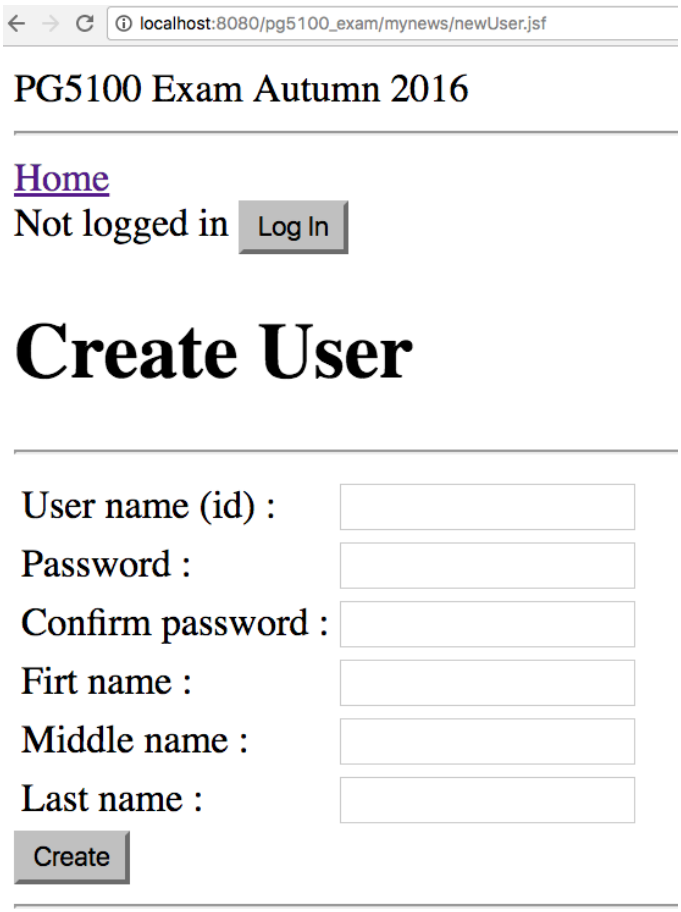


The header “PG5100 Exam Autumn 2016”, “Home” link, “Not logged it” and the “Log in” button should be in a layout.xhtml template, as they will have to be reused on all the pages. As the application is just started, no posts should be displayed yet. You need a text message to specify it, eg “No News yet”.

(E6, 5pt) Login and Create New Users

You need to reuse and *adapt* the code shown in class to login and create new users. Recall: it is very "unsecure". At this point, you do not need to worry about cookies. You will have a page for login (when clicking on the login button in the headers) and create new users like:





(E7, 5pt) Create News

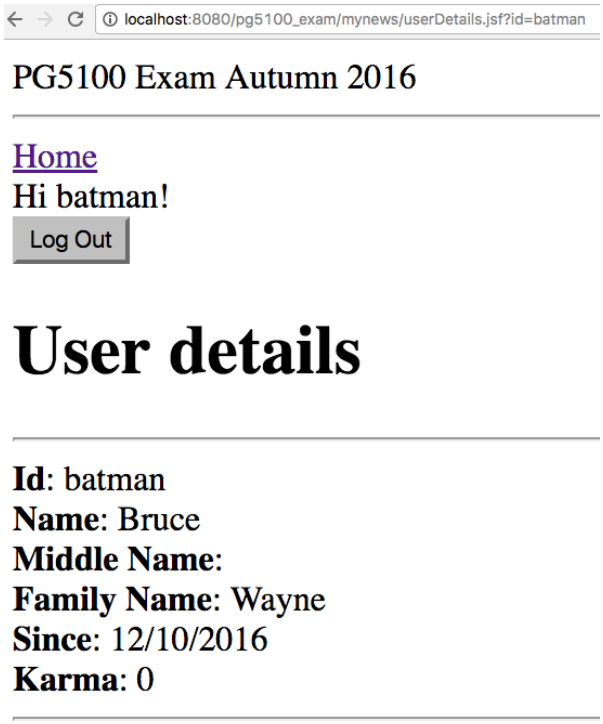
Once a user is logged in, and only then, on the home page he should be able to create a news. Furthermore:

- All news created so far should be displayed
- Each news should have a column showing: Time, Author, News, Score, Your Vote
- Only the first 30 characters of News's text should be visible. If longer than 30, show only the first 26, followed by " ...".
- "Your Vote" should only be visible when a user is logged in (as you need to keep track of who is voting what, non-logged in users cannot vote), and should contain 3 radio buttons. When a voting on a post changes, its Score should be updated
- Add option to sort the table by either "Time" or "Score"

See for example:

(E8, 5pt) Author Details

The userIds in the Post table should be links directing to a new page, showing all the details of a user, including its karma. For example:



(E9, 15pt) Selenium Tests:

Write Selenium tests in a file called *MyNewsIT.* Each web page should have a Page Object. You should write the following tests, in this order:

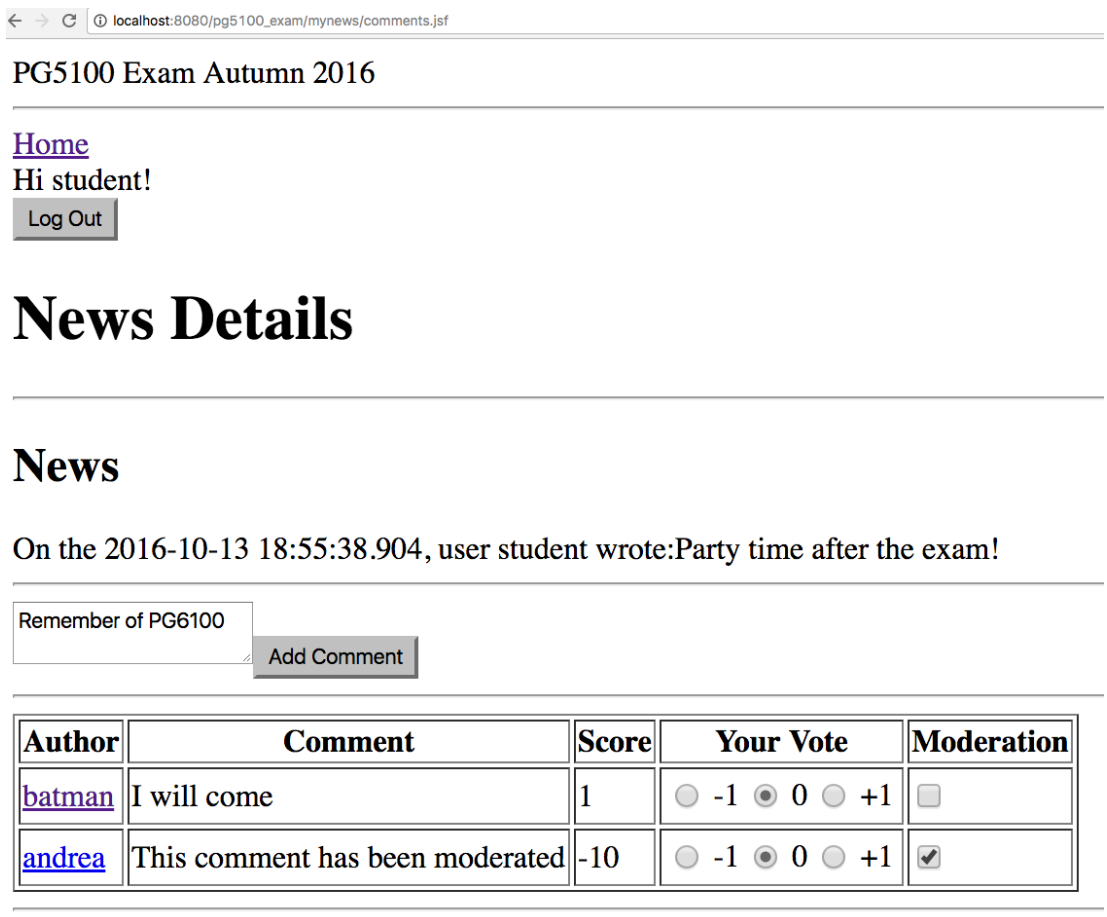
- testCreateNews
  - create and log in with a new user
  - assert that new user has no News in the home page
  - create a news
  - assert there should be 1 news from that user
  - create another news
  - assert there should be 2 news from that user
- testNewsAfterLogout
  - create and log in with a new user
  - create 2 News
  - assert those 2 posts are visible
  - logout
  - assert those 2 posts are still visible
- testUserDetails
  - create and log in with a new user
  - create a News
  - click on the “author” link of this news
  - verify that the application is led to a new page with the details for that user
- testCanVote
  - create and log in with a new user
  - create a News
  - verify that it is possible to vote for news
  - logout
  - verify that it is NOT possible to vote for news
  - login again with that user

    - verify that it is possible to vote for news
- testScore
    - create and log in with a new user
    - create a News
    - sort posts by time
    - assert that the most recent post has a 0 score
    - upvote the post
    - verify that post has now score +1
    - downvote the post
    - verify that post has now score -1
    - unvote the post
    - verify that post has now score 0
- testScoreWithTwoUsers
    - create and log in with a new user
    - create a post
    - sort post by time
    - upvote that post
    - verify that post has now score +1
    - logout
    - create and log in with a new user
    - upvote the previously created post
    - verify that post has now score +2
- testLongText
    - create and log in with a new user
    - create a post with a long text
    - verify that the text of that post has been properly trimmed
- testSorting
    - create and log in with a new user
    - create a post

- sort by time
- upvote the post
- create another post
- verify the posts are NOT sorted by score
- sort by score
- verify the posts are sorted by score
- sort by time
- verify the posts are NOT sorted by score

(E10, 5pt) News Details

The texts in the Post table should be links directing to a new page, showing the Post in full (ie who wrote it, when, and the full text), and also giving the ability to create and vote comments, but only for logged in users. The author of the post should be able to moderate each single comment. When moderated, the text of a comment should be replaced with “This comment has been moderated”. See for example:



(E11, 5pt) More Selenium Tests

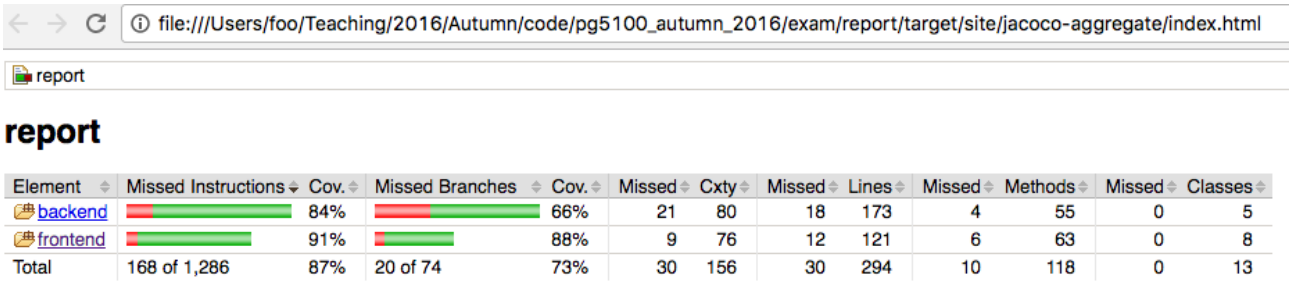
Add the following Selenium tests to *MyNewsIT:*

- testCreateComment
  - create and log in with a new user
  - create a post
  - open the details of that post
  - verify the post has no comments
  - create 3 comments
  - verify the post has 3 comments
- testCanModerate
  - create and log in with a new user
  - create a post
  - open the details of that post
  - create a comment
  - verify that the user can moderate comments
  - logout
  - re-open the details of that post
  - verify that a non-logged in user cannot moderate comments
  - go back to home page
  - create and log in with a new user
  - re-open the details of that post
  - verify that a non-author cannot moderate comments
  - logout
  - log in with the first user, author of that post
  - re-open the details of that post
  - verify that the user can moderate comments
- testKarma
  - create and log in with a new user
  - create a post

- upvote the post
- open the details of that post
- create two comments
- moderate both comments
- go back to the home page
- open the details of the author of that post
- verify that the user has karma “+1 -10 -10 = -19”

(E12, 5pt) JaCoCo

Configure JaCoCo in such a way that, when running “mvn clean verify” from the root folder, a test report should be generated under the report/target folder. You need an average instruction coverage of AT LEAST 80%. If it is lower, add more tests. Note: the score for this exercise is awarded if, and only if, all the previous exercises are (mostly) completed. See for example:



(E13, 5pt) JavaAgent

In class, when showing examples of “frontend” pom files with JaCoCo, there was:

```
<plugin>
    <groupId>org.wildfly.plugins</groupId>
    <artifactId>wildfly-maven-plugin</artifactId>
    <configuration>
        <!--
            This is quite complex (ie Java Agents and bytecode instrumentantion
on the fly)...
            for now just copy&paste it.
            If you are really interested in understanding what it is actually
doing,
            ask me in the breaks after the lectures
        -->
        <java-opts>
            -javaagent:${settings.localRepository}${fs}org${fs}jacoco$
{fs}org.jacoco.agent${fs}${version.jacoco}${fs}org.jacoco.agent-$
{version.jacoco}-runtime.jar=destfile=${basedir}${fs}target${fs}jacoco-it.exec
        </java-opts>
    </configuration>
</plugin>
```

Do explain what “it is actually doing”, eg why that string in the java-opts tag has to be written that way. Add such explanation as an extra point in the “theory.html” file. Note: I did NOT explain it in class. This is to test if you actually tried (or are able on your own) to find out what it is doing, or if you simply copy&paste code without actually understanding it :-) (and no, no student did “`ask me in the breaks after the lectures`”)

EXTRA

In the eventuality of you finishing all of the above exercises, and only then, if you have extra time left you can add functionalities/features to your project. Those extra functionalities need to be briefly discussed/listed in the “readme.txt” file. Note: there is no marking associated with this “extra” exercise. I will use it *only* in some border cases when deciding between a B and an A grade. Depending on the overall results of the *whole class*, it might also be possible to get an A without doing any extra feature.

THIS MARKS THE END OF THE EXAM TEXT