well done, corrections in red

Review of the project ese2015-team 2 done by team 7

First of all, we would like to stress, that an in-depth evaluation of the project ese2015 of the team 2 was not an easy task as we were not familiar with the used framework and servlet. Nevertheless, we did our best to understand the codes, the underlying ideas of the project and to provide you with a useful feedback on the completed work.

Design

Violation of MVC pattern

There is no violation of the MVC pattern in the project of ese2015-team 2. The project is very well structured and the code is situated appropriately. For example, if one wants to edit the profile page, one clicks on the “Profil editieren”-button and as a result the view changes to the “editprofile.xhtml”-page. This .xhtml file has no logic and is connected to the EditProfileBean.java class, which controls the whole editing process. This Bean class uses the well-designed model classes Profile.java, Customer.java and the model.dao classes ProfileFacade.java, CourseFacade.java and some others Daos. Therefore, everything runs the way it should be.

Usage of helper objects between view and model

Note here, that the ch.eset2.web.util classes- a collection of small classes with just a few methods- make the code in other classes, particularly in the beans, much more readable. For instance, the InitialsGenerator.java with the method generateInitials(Cusomer c) does exactly what its name predicts. Also, the Navigation.java class defines a few public static final Strings like “REGSUCCESS = “regSuccess.xhtml””. As a result of this class, the method registerNewCustomer() of the class RegistrationBean.java, has as return statement “return Navigation.REGSUCCESS;” which is self-explaining and makes the code very clear.

Rich OO domain model

The model is well designed and rich. We like that every user of your webpage is a customer and has a profile (@OneToOne) and then the TutorProfile and the StudentProfile extends their profile. Thus, we like the structure of the model. Additionally, we like that the CourseProfile class is between the profile and the courses, which in turn has a @ManyToOne relationship to the profile and the course.

Clear responsibilities

The responsibilities are very clear and the javadoc annotations are short but concise. Due to its great design, the project is easily understandable in just a few minutes.

Sound invariants

There are no invariant mentioned.

Overall code organization & reuse, e.g. views

The code is well organised and we did not find any redundant codes. At this stage of the project, the StudentProfile.java and the TutorProfile.java classes are the same, but the javadoc states: “In the current implementation there are no additional fields”

for the two classes. Therefore, this will be extended and is totally normal at this stage of the project.

Coding style

Consistency

what do you mean by codes used? is formatting, naming, etc. consistent?

Codes used are applied consistently throughout the program.

Intention-revealing names

The names are self-explaining and well chosen. We thus have no suggestions for improvement.

Do not repeat yourself

Do not repeat yourself…They do not.

too general: why do they make sense?

Exception, testing null values

are there any null value tests?

The thrown exceptions make sense and the code runs stable.

Encapsulation

In the Customer and the Profile Class are a lot of the variables defined as protected. This is not the best idea, but because of the AbstractFacade.class, which uses the javax.persistence.EntityManager to update the DB, we think this is probably necessary. Hence, we understand the choice of protected variables.

Assertion, contracts, invariant checks

Not found in the code

Utility methods

As already mentioned above, they are perfectly separated form the code and all collected in the ch.eset2.web.util package. We like this proper and tidy design and will try to implement something similar in our code.

Documentation

Understandable

Firstly, we want to congratulate the members of team 2 for the really helpful and funny REDME.md file. It invites the testers and all the other users of the webpage to explore the project built with NetBeans and Glassfish in a user-friendly way.
The REDME.md file gives a short intro, an installation instruction and some preinstalled usernames and passwords for a fast login.

Furthermore, we found in the documentation folder of the project an Entity Relationship Diagram and in the SRS file the diagrams of some important use cases. These explain the design and the ideas of the project in just a few minutes and is very useful.
Perhaps a sequence diagram of one use case would make the project somewhat faster understandable, even without knowing the used framework.

Intention-revealing

The documentation is absolutely intention-revealing.

Describe responsibilities

The responsibilities are well explained in the javadoc annotations. For example, the javadoc of the Profile.java class looks like this:

```
/**
 * Represents a profile of a {@link Customer} to hold extended customer information.
 * The customers are represented by this class. Other customers may access the
 * {@link Profile}.
 * Special fields:
 * id: Every Profile has an unique id.
 * {@link #customer}: Every profile belongs to exactly one customer.
 * {@link #courseProfiles}: Relationship to courses the profileowner attends.
 * @author Marc Jost, 17.10.2015
 */
```

As you can see, it is mentioned in short terms what you need to know about the responsibility of this class.

Match a consistent domain vocabulary

The domain vocabulary is consistent, there is nothing to criticise about.

Test

Clear and distinct test cases

The names of the tests are self-explanatory and for example the CourseTest.java is well documented. The tests of the model are good, but you could perhaps use some mockito and a bit more junit. Also, a bit more integration testing would be great.

where to use mockito, where to use JUnit? where use an integration test?

Number/coverage of test cases

We think your code is better than your tests, but they are ok, at this stage of the project. Perhaps you could test one or two more Beans.

Easy to understand the case that is tested

We had no problem to understand the tested cases.

Well crafted set of test data

We like the three DevelopementData* classes, which fill the DB automatically with some test data. It is simple, but very useful.

Readability

The tests are very readable, nothing to complain about.

Pick a class of choice from the controller package and analyze its code. Does the class have too many responsibilities ? Is there some logic that should be moved to another class ? If so, why ?

A short comment to the **EditProfileBean.java** class with the javadoc description:

```
/**
 * EditeProfileBean manages changes of the profile.
 * It allows to persist edited profile variables to the database.
 * {@link EditProfileBean#retrieveCustomer() } should be called from the server
before
 * using any service of this class.
 * @author Marc Jost
 * @version 1.0
 */
```

Code analyse and should some logic be moved to another class

Lines	Comments
77 – 79	This method gets the initials from the current customer, but the logic is not in the Bean class, it is in the InitalsGenerator.java a util class, they defined. This is a very proper way to do it, because the logic would be fast implemented in the Bean class, but like this it's really good readable and easily understandable.

The method in the EditProfileBean.java class:

```
72      /**
73       * Generates the Initials uses for the profile picture (if no pic is present)
74       * @return String representation of a customer's initials. E.g Marc Jost
75       *           returns MJ
76       */
77      public String getInitials(){
78          return InitialsGenerator.generateInitials(customer);
79      }
```

The logic in the InitialsGenerator.java class:

```
17  public class InitialsGenerator {
18
19      /**
20       * Generates Initials from the Customers First and Last Name.
21       * E.g. A Customer called Marc Jost would generate initials MJ
22       *
23       * @param c the customer to generate initials from
24       * @return the initials. E.g. Marc Jost returns MJ
25       */
26      public static String generateInitials(Customer c){
27          StringBuilder sb = new StringBuilder();
28          return sb.append(c.getFirstName().charAt(0)).append(c.getLastName().charAt(0)).toString();
29      }
30  }
```

Lines	Comments
85 – 95	In the method saveProfile() the profile changes will be saved in the database.

```
85      public String saveProfile() {
86          try {
87              profileFacade.edit(profile);
88              customerFacade.edit(customer);
89              return Navigation.VIEWPROFILE + "?faces-redirect=true&id=" + userHelper.getMyProfileID();
90          } catch (Exception e) { // TODO
91              System.out.println("ch.eset2.web.beans.EditProfileBean.saveProfile()");
                e.printStackTrace();
93              return null;
94          }
95      }
```

Also, here the chosen solution is very readable and there is not too much logic in this Bean class.

87 For example, the line `profileFacade.edit(profile)` calls a method of the AbstractFacade.java class, which updates the database with the entity manager.

AbstractFacade.java method edit(T entity):

```
24      public void edit(T entity) {
25          getEntityManager().merge(entity);
26      }
```

89 As a consequence, the return statement of the try bloc is well designed. Again, here you are using two little helpers, the Navigation.java and the UserHelper.java classes.
For example, instead of writing "viewprofile.xhtml" you implemented it as Navigation.VIEWPROFILE, which makes, even semantically, a lot of sense.

The Navigation.class with all the navigation Cases we really liked:

```
public class Navigation {
    public static final String INDEX = "index.xhtml";
    public static final String REGISTRATION = "registration.xhtml";
    public static final String REGSUCCESS = "regSuccess.xhtml";
    public static final String EDITPROFILE = "editprofile.xhtml";
    public static final String VIEWPROFILE = "viewprofile.xhtml";
    public static final String SENDSUCCESS = "sendSuccess.xhtml";
}
```

102 – 122 In these lines you have two methods. One is the removeCourseProfile(), which removes a courseProfile from the current profile, from the course and from the table CourseProfile. The other method is removeProfile(), which removes the whole profile from the database.
Both work like the saveProvile()-method and the logic is not in the wrongly placed. All the above mentioned three methods are very consistent with each other.

125 – 142 Some getters and setters