CODE REVIEW
by Team 7

for

ESE 2014 Project by Team 2

Version 1.0

S. Berger, J. Niklaus, J. Schaerer, A. Sellathurai

Contents

1 General Review

1.1 Design

1.1.1 Violation of MVC pattern

Method enlistad in AdController is huge. Sure more could be extracted that does not belong to the controller. Method getUserImage in AdController deals with images as well as enlistad. Maybe a PictureService is missing. PICTURE_LOCATION and servletContext belong to the PictureService as well. The same applies for enlistroomie in RoomieController.
Why does does not the Adservice return directly the interessents, but they are constructed in besichtigungsterminSetzen in AppointmentController?

1.1.2 Usage of helper objects between view and model

Method enlistad in AdController uses helper object PictureManager.

1.1.3 Rich OO domain model

Pretty good. I cannot think of any missing model object.

1.1.4 Clear responsibilities

All success messages collected in success.jsp: They don't have anything in common. Why not in individual pages? Redirects necessary but why? (user cannot interact with website during redirect.)
Good use of ErrorSaver object.

1.1.5 Sound invariants

I did not find any invariants in the code.

1.1.6 Overall code organization and reuse, e.g. views

Exceptions folder and thus InvalidUserExceptions exists twice.
But otherwise reuse and organization fine.

1.2 Coding Style

1.2.1 Consistency

The naming is Consistent, except the URLs they sometimes use a capital letter on new words and sometimes not. UserController "/setInformFilter", "/myprofile", ...

The code formatting is sometimes very inconsistent. It would make the code much more readable if there was a better use of whitespaces. AppointmentController Line 102 to 118, use tabs to create visual and locigal blocks.
ApplicationsController Line 127 if a method header is longer than one line use tabs to indent the second line.
RoomieController Line 89 to 106 use tabs to visually group the for loop.
Try to use always the same amount of line breaks to separate methods. In many classes for example ApplicationForm Line 34 to 38.
Sometimes there are line breaks after the method header and sometimes not. If the method header has more than one line it might be more readable if there is a empty line after it. But it should be always the same count to increase the consistency and that readability.
Also the .jsp files lack a consistent formatting. With every opening tag a tab should be added and with every closing tag a tab should be removed to make the code much more readable. There are also a lot of unused line brakes like in register.jsp 77 to 97.

Else the common code conventions are adhered and the code is consistent. Similar problems are solved in a similar way, like its well done in the Controllers.

1.2.2 Intention-revealing names

The names are mostly chosen wisely and reveal their intention. One exception is the variable named getters which occurs in multiple Classes like FilterAdsController and UserController. Out of the context its not clear what this variable contains.

1.2.3 Do not repeat yourself

In the AdForm are four getter and setter methods for pictues. A dynamic method could remove duplicated code and remove the the limitation to four images.

1.2.4 Exception, testing null values

Exception handling is only done on a few places. The catch clause is mostly not used, so that the try clause is abused to inhibit errors. Inform the User when something went wrong, that he can recover and fix the error. If it is not necessary to push an error up to a user, the error message can be printed to a log or the console so that it is visible to the developers. Then if an error always occurs something might be wrong and should be improved.

The code can be improved by the use of more exception handling like its done with the InvalidUserException in the UserController line 129 to 135.

1.2.5 Encapsulation

The encapsulations is ok and follows the conventions of Spring MVC.

1.2.6 Assertion, contracts, invariant checks

Clear and good contracts but no assertions and invariant checks at all. Use assertions and invariant checks to find violations of the contracts and improve the stability of your software.

1.2.7 Utility methods

There are some huge methods which can be split up in some Utility methods. For example in RoomieController on line 66 to 116 a path to a picture is assembled. We find almost the same code in the AdController on line 153 to 207. The duplicate code in those sections could be put into a Utility method.

1.3 Documentation

1.3.1 Understandable

In general, the documentation is very understandable. It creates a clear idea of what the code does but is still not too long. Still there are some minor issues to be noticed. An example is AdController.showAdId: The comment isn't very understandable. It talks about "needed criteria" but it is not clear what these are. Similarly, the doc comment for FilterAdsController.filterAdsIndex talks about a "smaller filter" but it is not obvious what this is and why it is needed. Another problematic documentation is the one for AdController.placeYourself. It says it redirects to the "place yourself" page. Unfortunately, it is not clear what this page is or does. As far as it goes for the method name and request mapping, it looks like it is responsible for placing a user, room mate or something like that. This wouldn't really make sense in that class, so it remains unclear what the method exactly does.

1.3.2 Intention-revealing

Throughout the documentation, the intention of each method is clearly shown. The comments give a brief but revealing overview about what the methods are used for, which is going to be a big help for everyone to maintain the code in the future.

1.3.3 Describe Responsibilities

Many parts of the documentation lack a clear description of responsibilities. Often the just say "...is triggered, when...". Sometimes the comments even state which element on

which .jsp triggers the method. While this is really helpful when maintaining the code, it is not really the idea of responsibility driven design. If there is going to be another view to use the same method, the documentation will no longer be fully correct. An even worse case could be when a part of the system is changed and the method is not even invoked by the stated view any more. Nevertheless, most responsibilities are still clear, since they're implied by the method name and the commentary.

1.3.4 Consistent Domain Vocabulary

Most of the used vocabulary seems to be consistent throughout the documentation. This is, with one exception: the term for a person to look for an apartment. It is referred to as "mate", "yourself", "interessent", "interested user", "applicant", "person" and "roomie". There should be one single, exactly defined word for that. Apart from that, the vocabulary is very accurate and adds to quite understandable documentation.

1.4 Tests

1.4.1 Clear and distinct test cases

We have found the tests clear and distinct. They are easy to read and it is very easy to see what is tested thanks to correct and understandable naming and design of the test class.

1.4.2 Number/coverage of test cases

There's only one test for FilterLogicService. The overall coverage and number of test cases has to be improved.

1.4.3 Easy to understand the case that is tested

The cases that are tested are easily understandable thanks to the good readability and coding logic.

1.4.4 Well crafted set of test data

The test data is very well crafted and can further be improved by quantitative entries, nevertheless the provided sample data is sufficient for the used testing.

1.4.5 Readability

The overall and detailed readability of the test is very high. The developer followed all the naming and coding conventions. Is also a master example of test writing.

2 Code Analysis: AdController

The AdController class has quite a lot of responsibilities. Most of them make sense as a part of that class since there are a lot of different tasks and views related to ads but there could be a way to split the responsibilities more clearly. There could be made a distinction between displaying and managing ads. This would mean to move methods like "showAds", "showUnfiltredAds" and "showAdId" to a class like "AdDisplayController" and methods like "createAd" and "enlistad" to a one like "AdManagementController". Like this, the responsibilities of the AdController would be split up and could be more clearly defined.

Another questionable point is the placeYourself method. It's not exactly clear from method name and documentation what the method does, this is, what the "placeyourself" page is. Looking at method name and request mapping, as well as placeyourself.jsp, it seems to be a page where a user or room mate can somehow be added. If this is the case, then that method should be moved to another class, such as UserController or RoomieController, since it doesn't seem to be in direct relation to ads.