Review for Team 4 (share-a-flat)

Design

Violation of MVC pattern

No violation of the MVC pattern discernible. We had a good introduction thanks to the Skeletons and the warm-ups, Team 4 followed those examples consequently and therefore stuck to the pattern. They applied the Spring Web MVC framework well, and made good use of the default handler via the @Controller and the @RequestMapping annotations. However, the files AdType.java and Sex.java should probably be moved from the controller.pojos package to the model package. According to page 6 from the SpringMVC presentation from the ESE-Wiki, those two enums don't belong in the same package as the forms which have other responsibilities as data classes between jsp files and controllers. Concerning the Model, the clear folder structure within the webapp should be positively mentioned.

Usage of helper objects between view and model

There are no helper objects between view and model.

Rich OO domain model

There could be more classes in the model-package, such as a class Picture instead of the byte[] array in Profile.java. With additional functionality, more classes such as Message will surely be created. The four existing classes themselves are good in my opinion. There is some use of aggregation, an Ad has a an owner, an address, a type etc.

Clear responsibilities

In HomeController.java, there is the method login(). This method should probably be moved to LoginController.java as the name of the method suggests. Also, the Profile.java and User.java files were a bit confusing at first. The fields 'sex' and 'age' are most likely in Profile.java since they are edited in the profile. As an outsider, we would expected them in the class User at first due to common sense, but this is probably a minor matter of choice. In TabBarController.java, there are the methods searchList() and searchMap(), which we would expect to be in SearchController.java. In the page, the tab bar contains an entry 'Search', while the 'Search List' and 'Search Map' are not in the tab bar but in the Search page itself. The javadoc comment "Controls all pages / commands concerning ads" in ImageController.java is confusing / wrong since there is a Class AdController.java. The responsibilities of the forms in the pojos package are clear. The responsibility of ReviewService.java is clear, this class saves some users and ads into the database for testing purposes. However, a renaming (and splitting) into something like UserTestDataSeeder and AdTestDataSeeder would be good; the name ReviewService is rather ambiguous. In addition, we strongly suggest that this class is moved into the test packages (src/test/java) since it has another responsibility than the rest of the classes in the package (such as AdServiceImpl etc.). The distinction / responsibilities of LoginService.java and NewAccountServiceImpl.java is confusing, the method loginManually() is in NewAccountServiceImpl.java. Those two classes could probably be merged into something like AccountService.java.

Sound invariants

There are no class invariants and no assertions (except in the tests of course) in the code.

Overall code organization & reuse, e.g. views

In the controller.service package, the usage of interfaces is handled inconsistently. Some classes like AdService implement a corresponding interface, some classes like ReviewService have none. The code organization is good for the majority of the project. However, there is a bad mix of files in the folder src/main/resources. There is a png, js and some xml files, those should be put in separate folders. 7 files at this point isn't a big deal, but for a way bigger project, a good separation is a must. There isn't too much code reuse, but also, a huge class hierarchy certainly wasn't the way to go here. For example, there are currently four classes in the model package, one could factor out the id and its getter/setter into a superclass, which would not make much sense, but apart from that, these classes don't have much more in common.

Coding style

Reading the code was very smooth and mostly self-explanatory. Finding all the relevant information was pretty easy (with a few exceptions) due to intuitive naming grouping of the classes, fields and methods. All in all the code looks clean although there are still many comments hinting unfinished code or further problems that have not yet been solved.

You have many imports in (almost all) your files which were copy pasted and never removed. It does confuse the reader and impacts performance. In the ImageController you have much more unused imports than used ones.

Consistency

The .jsp file naming seems to be a little bit inconsistent. Sometimes you use a dash to separate words (create-ad.jsp) sometimes you use camel case (modifyProfile.jsp). That can make it harder to match and read pages like in the controller classes or even in the header.jsp lines 56 to 60. It would be easier to define a style within the group and follow it.

Sometimes you use in-line css styling sometimes you use it from an external css file. What makes it even harder to read is the fact that you use compressed/minified css and js files. During the development phase readability should be preferred. For production phase a minified version could be faster and therefore better.

Some basic things in the following paragraph which don't fit better into another category. There are many ignored warnings such as unused imports. The code formatting is inconsistent, for example in my-page.jsp (use the format-hotkey, works wonders). There are some undeleted println-statements in submitAd() in AdController.java. Fully commented method in HomeController.java (probably an old version not yet deleted for testing?)

Intention-revealing names

The TabBarController seems to be responsible for several things that are visually together but not logically. For example, searching for the place where SearchForm was added (in the SearchController) turned out to be quite cumbersome. Maybe that could be grouped better?

Sex.java is nice but leaves me thinking something else. Maybe Gender.java would be a better alternative.

Do you really intend to use a form for UploadForm? Most likely it is just a functionality inside AdForm.java therefore it should probably not be a class.

Why do you use a DefaultSearcher with an ISearcher (normally you name the interface and the implementation of the interface differently: e.g. AdService.java and AdServiceImpl.java). DefaultSearcher implies there should be at least one other Searcher.

ReviewService.java is most likely not a service as specified by the @Service annotation to be called when needed but rather test data for development only. It could be placed to src/test/java.

In ReviewService.java, the autowired fileds NewAccountService 'nas' and AdService 'as' aren't intention-revealing names.

Do not repeat yourself

You have two different headers. headerLogin.jsp opens many div tags which you try to close in register.jsp in lines 54 to 56 (without indentation nor comments) and then again in lines 123 to 125.

You have an empty footer which closes certain tags and does nothing else. On some pages (register.jsp, index.jsp, etc.) you specify a footer with the class name mastfoot additionally, on most pages you don't. (Why not separate mastfoot to mast-foot like all other classes within html?)

Exception, testing null values

You are testing for null values in your service classes like in return statements (UserServiceImpl.java) which is perfect. Although you do catch Exception in ProfileServiceImpl.java which is never a good idea, since catching an Exception means catching every possible exception.

You have defined your own three exceptions to make debugging more easy and to further specify what can go wrong and where it goes wrong. That is great. Especially in NewAccountServiceImpl.java you have different messages passing to InvalidUserException.java giving just the right feedback! However, it would be even more elegant to handle this kind of feedback as input validation with javascript or something similar rather than as an Exception.

Encapsulation

The encapsulation seems to be good. All model classes have fields with their respective setters and getters according to JavaBean Definition. Setters and getters can be therefore used by Spring.

What is really unclear is the "default" constructor in User.java where you set the Owner to itself basically. Why is an owner even necessary? Would it not be easier to group these two classes into one? I see that you want to have a User and then link it to a Profile which is nice, but it is not clear how the software profits from it.

In the User.java class you have two methods - a public getAuthorities() and its private translateRole(). You are accesing getAuthorities() from outside its own model; that should be avoided. In the model there should only be setters and getters for the model and if needed private helper methods only (that's for the business logic/class logic of the class only). Further you have four methods at the end which return TRUE only. They are without a field. I guess the code within the body will follow in the next step?

Assertion, contracts, invariant checks

There are no assertions in the sourcecode (but there are some in the tests which is good). No contracts have been specified nor invariant checks implemented.

Utility methods

You are using java's utility methods for many tasks. You also imported external JavaScript classes which you are using throughout the project. There are no own utility classes (java nor javascript) that you use for recurring tasks like validation of input or simple calculations. But you're probably not really missing them, since there is no real recurring task you need to do from different objects many times.

Documentation

Understandable

Most of the SRS is understandable. There are, however, a number of passages that clearly stand in the way of understanding.

1. The use case diagram (p. 3) leaves us wondering at the meaning of "include", since according to the diagram, logging in and out of account apparently includes creating an account. Sure that's a precondition, but it isn't included *every time*. Same thing goes for the "include" statement where the export of a PDF file apparently includes the search for an ad (more on that later).
2. At the beginning, we are missing some kind of page overview, i.e. which buttons/options exist, which of them are always visible, which of them are in sub-menus etc. In the document, it often seems like there is a clear model of where what is, but it is never made explicit. This shortcoming becomes the most tangible when bookmarking favourites is discussed. Descriptions vary between "bookmarking" (e.g.

p. 12), clicking on "the "star" of candidates" (e.g. p. 8), "clicking onto 'star'" (e.g. p. 8), and "clicks onto the "star"" (p. 12). So is it an actual star (image) you click on? Or a button/text named "star"? Or "bookmark"?
3. Throughout the SRS (12 use cases in total), not *once* are any special requirements named, and only two times are notes provided, with one of them ("Account can only be created if the username does not yet exist" in use case "Create Account") being not so much of a note but rather one possible validation step that should be included in the main scenario. The effect of this on understanding is thoroughly detrimental.
4. In the different use case scenarios, after having stepped through the whole use case, the system sometimes takes us to another page (e.g. use cases "place ad", "create schedule", "create account"), sometimes not (e.g. use cases "manage ads", "edit profile", "check-out profile"). It is not clear why after some actions on the system the user would be redirected to the start page and not after others.
5. Generally, the authors seem to not have foreseen the situation that a user is viewing an object or user profile withouth having searched for it *through the site itself* (e.g. use case 9, pre-conditions). However, with a product like this, I want to be able to send direct links to objects to friends, let's say by email. Is this a conscious decision by the authors?

Intention-revealing

Some use cases reveal their intention plain and clear. Others not so much. Especially the order of the use cases is really tough to understand. Is there any order at all? Certainly, there is none based on use cases belonging to the same user story. This is especially frustrating with the use cases concerning the user story of managing visits.

1. Use case 1, "create schedule to manage visit" (p. 5), only targets the advertiser as an actor. For each of her objects, the advertiser selects some times slots on a calendar, and that's that for this use case. It would be *a lot* better for understanding, if either the other use cases belonging to that user story came right after this one, or there was some explanation of the *whole* user story in the "Notes" section (which is literally *never* used in the whole SRS), or, of course, ideally both.
2. The Alternative scenario in use case 7 has no name, and thus does not reveal its intention. Also, it is not really an alternative scenario at all, just another way to achieve exactly the same thing, i.e. another use case.
3. The opposite happens in use case 3 ("search ad"), where it is an alternative scenario of its own if no matching ads are found. This is a highly common case and can simply be included in the main scenario by a line à la "System returns a list of results (maybe empty)".
4. In the same use case, there is an alternative scenario by the title "Searcher wants to see the ads on the map-view". The first step is "Searcher clicks onto 'Map View'", and thus accomplishes the goal. The remaining points are further options concerning the map view, which are great to have, but are clearly mislabelled here.

Describe responsibilities

Mostly given. Even the following shortcomings are by no means serious; most information can be inferred correctly from context. Still:

1. In more than one use case main scenario, points are omitted. These are mostly actions by the system (while actions by the user are covered in great detail, as desired). For example, in use case 4, step 7 of the main scenario is that the user bookmarks a candidate by clicking on a star (or other, similar action). Step 8 reads: "Advertiser clicks 'OK'". There is no mention of the system prompting the advertiser to confirm. In use case 5, step 2 of the process reads "User clicks onto 'Register'", and step 3 immediately has the system confirm that the account has been successfully created. We are missing the whole validation process, and the process of persisting data to the database. Some more examples like these can be found.
2. Sometimes, one simple enough step is further divided in main scenarios. For example, in use case 6, step 3 is "User clicks in the field", while step 4 is "User makes changes". It is pretty obvious, though, that to make changes in a field you first have to click in it.
3. Several times, multiple use cases are wrapped into one. Unlike point 2, this is very serious. For example, the alternative scenario to use case 6 ("edit profile") is "delete account", which is obviously a use case of its own. Also, in use case 4 ("compile promising candidates list"), the main scenario has the advertiser browse through users and bookmark them. Step 12, however, is "Advertiser can export the list by clicking onto 'Export List'", which is just as clearly a distinct use case.
4. In use case 9, "Export information from ad", which is about exporting ad information to a PDF file downloaded to a user's machine, there are no alternative scenarios listed. That seems a bit too optimistic. What if the export fails, for one of numerous reasons? What if the user's hard disk is already full? What if the user's firewall intercepts the download, or some other scenario?
5. It is worth mentioning that the list of functional and non-functional requirements (p. 16f.) is very thorough. Unfortunately it is placed right at the end; a shorter version of it could make the whole document easier to read. Nevertheless, good job on the requirements.

Match a consistent domain vocabulary

Room for improvement here. Not all remarks given are strictly concerned with *consistency* of the vocabulary, a few more general remarks are also in order.

1. The most irritating inconsistency is with navigation elements on the website. These are sometimes called "tab" (p. 7, 10), but sometimes also "register" (p. 4); mostly, though, they are not referred to by any specific type. The wording "... clicks on [name]" is very common throughout the document. Here we are again missing a layout/menu overview as described in "Understandable", paragraph 2.
2. The entities referred to by ads are not named consistently. We have "object" (p. 4, 14), "apartment"/"room" (p. 5), as well as "property" (p. 16).
3. General vocabulary remark: Use case 7 is entitled "Check-out profile", the profile in question belonging to another user of the platform. As long as you're not a frat house

social media website, I would strongly suggest "inspecting" or "viewing" a profile instead of "checking it out".

4. Sometimes, (semi-)technical and everyday vocabulary is used inconsistently. In some cases, we read that the "System asks whether [user] is sure or not" (p. 5) or similarly colloquial (e.g. p. 6) wordings. On the other hand, the word "prompt" is used correctly on multiple occasions (e.g. p.9). Another example would be that on p. 4 a field is "mutate[d]", while on other occasions (e.g. p. 10) it is simply said that a user "makes changes" or similar.
5. All in all, the use cases seem to have been written by different people without only a bare minimum (if any) of proofreading or editing. (For example, use cases 8 and 9 reveal a minimalistic approach). The fact that this is still visible after at least two revisions (together with a plethora of typos and very German expressions) is proof that other tasks, such as developing, have been given much higher priority by the authors.
6. One last point concerning the consistency of the domain vocabulary: Unfortunately, in one aspect the vocabulary is extremely consistent, and that is in the generic masculine used literally everywhere (!) where users are referenced. Throughout the document, it's always "he searches" etc., not one single time "he or she" or other including wording is used. No offence, but what is this, the early 1800s?

Test

Clear and distinct test cases

The test cases are clearly separated. Care was exercised that only one aspect of an object was tested per test case. It is not possible to further divide a test case into several new ones, which is further evidence that they are distinct. Since the `SearchTests` class contains unit tests for the class `DefaultSearcher`, it would make sense to rename it to `DefaultSearcherTest`. This suggestion stems from two main reasons. First of all, it would comply to common convention. JUnit and especially Maven Surefire unit tests are commonly named with the pattern `ATest` for class `A`. Per default, Surefire even relies on this naming to find the test cases within a Maven project. The second reason is that the current naming is confusing for someone searching for the class that is under test. Which class is tested can currently only be found out through looking at the source code of the tests. The second testing class is called `LoginTest`. Although there is no class `Login.java` this name is okay, since `LoginTest` contains integration tests instead of unit tests, it tests several components at once.

Number/coverage of test cases

The project contains two test classes. In total 18 test cases can be found. One class contains unit tests and the other one integration tests. The classes that are covered through these tests are tested thoroughly. Common uses of the involved classes are tested as well as unexpected situations, such as exceptions or empty results. The number of tested classes could be higher. Although the two tested classes are tested very well, there remain a lot of classes that are untested. Of course, there are classes that do not absolutely need to be tested, for example a form consisting solely of getters and setters methods. On the other hand, a class like `ImageServiceImpl.java` would justify adding some more tests. Its functionality is quite complex (especially the parts that are currently commented out). As of version 1.0, uploading a picture does not work, an exception occurs. Maybe this could have been prevented by some more unit tests. It would make sense to make sure that all tests in the project are run by Maven if its `mvn test` goal is executed. Currently, only the tests in `LoginTest.java` are executed by this goal. In our opinion, all tests of a project should be runnable in one step only.

Easy to understand the case that is tested

All test cases are reasonably named and therefore also easily understandable. Test cases within one class all have a similar structure, which makes it easy to distinguish what exactly is tested in each test case. Furthermore, as already mentioned, variable names were chosen reasonably.

Well crafted set of test data

The tests cover common uses cases well. For most parameters of a test case, several values and combinations of those are used. For example, in `SearcherTest.java` every test ad has a different address, a different price and a varying number of rooms. This is good design, since it makes sense to test the searcher methods with several different value combinations. One thing that could be improved is the coverage of uncommon use cases with uncommon data. This would mean adding some corner and special cases to the test data, for example a very high or very low ad price, zero roommates (for a studio) or city names with special characters like 'é' in them. The tests make good usage of mocking. Through mocking of the ad data access object, the tests are separated from the actual database, which ensures that only the searcher class is tested and not other classes as well.

Readability

As already mentioned before, the test cases are named reasonably, therefore the readability is good. Also variables are named according to what they are used for, although sometimes naming conventions are not followed (e.g. `String EMAIL` instead of `String email`). In addition, mocked objects are clearly separated from real objects through appriopriate naming. The two test classes do not have consistent names. While one class is named `LoginTest` (singular) the other one is named `SearcherTests` (plural). It would be better if both test classes used the singular. This might seem overly picky at first, but it is mainly due to the fact that Maven Surefire only runs the test cases of classes using the singular version per default. The way the test files were configured at the time of the release, Maven Surefire only ran the tests from `LoginTest`.

Analysis of a class within the controller package

We picked the class TabBarController.java from the controller package.

The tab bar of the website contains a home icon and the entries 'My Page', 'My Favorites', 'Search', 'Create an ad' and 'Logout'. To be precise, there is a second 'Logout' at the top right corner, but one of the two 'Logout's will surely be removed as the site is under construction, so let's assume that there is only one. According to the name "TabBarController", one expects that it handles the mappings of all those 6 entries, but the class contains only the mappings for 'My Favorites' and two mappings for 'Search', depending whether the user wants to perform a search by some criteria or by google maps. Furthermore, there is a mapping for a security-error: going to the page .../security-error manually results in getting logged out. This mapping should be moved to LoginController.java, but the author may had her/his reasons to put it into the class TabBarController. The grouping of the controllers could clearly be improved. As said, there is a home icon in the tab bar, but the mapping isn't in the TabBarController, there is a seperate Controller named HomeController which handles this. The HomeController maps only "/home" and "/test" (which crashes at the moment with a 404 Not_found Error). Same stories with the tab bar entries 'My-Page', 'Search' and 'Create an ad'. There are the classes MyPageController, SearchController and AdController which handle these mappings already. In the case of 'Search', "/search" is handled in SearchController while "/search-list" and "/search-map" are handled in the TabBarController.

So it's hard to tell whether the TabBarController class has too many responsibilities because first of all, the responsibilities seem to be handled / split a bit unclear. If it would contain all the mappings of the tab bar as the classname suggests, then it steals responsibilities from other classes such as AdController, SearchController etc. And that's why creating a TabBarController class at all is the wrong way to go here; grouping the controllers in a slightly different way would yield better clarity. Controllers should be regrouped, for example in files called 'AccountController', 'AdController', 'IndexController', 'MessageController', 'SearchController' etc. If one groups the controllers like this, it is clear where to put mappings for ads for example, it doesn't matter if a link is in a tab bar or elsewhere.

If one decides to delete the class and therefore move its mappings, the two search mappings would go into the SearchController, the security error mapping would go into LoginController, and the favorite-mapping would need an own controller if one continues the given grouping here (Home, MyPage, Search, CreateAd all got their own controllers).