Peer review of the application Boki

Adam Törnkvist, Isak Magnusson, Maria Fornmark, Mathias Lammers, Viktor Fredholm

1. Do the design and implementation follow design principles?

1.1. Does the project use a consistent coding style?

We haven't noticed any inconsistencies regarding the style of coding, you wouldn't really be able to tell who wrote one part of the code from another. As far as we have seen, the code follow good praxis. One example of this is that all the constants are written in capital letters.

1.2. Is the code reusable?

Yes, pretty much. Doesn't seem to be that much work in order to break out stuff and put it in something else.

1.3. Is it easy to maintain?

We would've liked more comments explaining the steps taken inside the methods, so someone who have not written the code could more easily understand it.

1.4. Can we easily add/remove functionality?

One good example of being able to add or remove functionality is the possibility of sorting the adverts in different ways. If one would like to remove the possibility of sorting, let's say alphabetically, you'd only had to remove the class and a few lines of code in SortManager and SortFactory. This could of course be improved, but given our current level of knowledge, we think this is well done.

1.5. Are design patterns used?

Multiple instances of factories are used. This is a good way to decrease the dependencies within an application.

A strategy pattern is used within the sorting package for the sorting of the lists. This is very helpful if some sorting options would change or if they would like to add or remove options in the future.

2. Is the code documented?

A few methods are documented using JavaDoc, but the vast majority isn't. Many, but far from all the classes have class descriptions and there are barely any comments inside the classes explaining the process. All in all, there could be a big improvement on this part.

3. Are proper names used?

The names of variables and methods used are suitable for their purpose and easy to understand.

4. Is the design modular? Are there any unnecessary dependencies?

Yes the design is modular and no obvious bad dependencies could be found.

5. Does the code use proper abstractions?

There are several interfaces which the classes implements. One example of this is how the backend is implementing its interface, it uses a singleton pattern to make sure that there is only a single instance of the backend at a given time. This is done using a factory and an interface, where the backend controller is “disguised” as the interface to increase abstraction.

6. Is the code well tested?

All the methods in FormHelper have been tested and seem reasonable. Though, this is only one of the few classes that have been tested, the majority of the classes have not. In many of the tests that have been made, more than one case has been tested which is good.

In some of the tests (for example the presenter in CreateAdPresenterTest) the object could’ve been initialized in the beginning of the class, for less clutter and to make the tests easier to read.

7. Are there any security problems, are there any performance issues?

The application authenticates the user’s username and password using a FirebaseAuth, which surely must be safer than an authenticator designed by a group of students with our security and programming knowledge. This way, the application only knows the input of the user, and whether the input was correct.

8. Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

The code follows the MVP design pattern, which probably should be mentioned in the System Design Document. As far as we have found, the different packages have the correct dependencies of each other, according to the design pattern.

9. Can the design or code be improved? Are there better solutions?

There are a few instances where improvement might be warranted. Clicking on one of the three conditionbuttons yields no response to the user. Something to mark that you have selected one would be good.