Code review for team 6

Performed by team 4

First impression

After importing the project into Eclipse it already shows a lot of error in the test file. Even though the application runs, it should never have compilation errors even in folder not used, because it scares new developer and they're even afraid of touching the code.

Secondly there is no documentation at all. This is specifically a problem as soon as the developer discovers the use of the JavaScript Object Notation (JSON), which is unknown to most students. Without further knowledge of JSON it is not possible to tell, how the application runs at all! Therefore documentation is essential.

Also quite important for the first impression of new code is the coding style, which includes simple formatting used within the project. In this specific project it doesn't follow the Java code standards and convention in large parts, especially with wrapping lines, method declaration and indentation.

Another thing that strikes unfamiliar users quite a lot: It doesn't seem possible to login as a test user. With the user name `Bob` and the password `secret` it doesn't work. We had to create new users all the time.

Overview

Design

I would be quite interested how the team came to the decision to use JSON, a exchange format based on String exchange. The current design with this technology leads to a very small controller class, only one view template and one huge message class. For rendering a page the controller creates new message object which storages all kind of data. Each controller gets a String object called 'blob' as a parameter.

I see several problems with that kind of design:

- No object oriented design: This design hurts several concepts of OOD. The message class is basically a 'god class' which does everything. The same is true on the view side, where you have one master template which renders all the views. Even if you use the now existing design it would be better to make the seperation of views within the controller and render then different templates rather than check the `msg.curCaller` within the view.
- Bad traceability: By just reading the code there is no possibility to trace what the methods in the message class actually do. The just get 'blob' and 'params' which could be everything and nothing...
- Unnecessary complexity: As I mentioned before I'd really like to hear the arguments for this kind of design. I currently don't see any. The call hierarchy for a simple action is very complex. For example the user clicks on one of his calendars in the home view:
    1. The link leads to /username/calendarname which is then routed to the controller method 'master'.

2. The 'master'-method creates a new message object with the current params scope and router scope.
3. The router scope is then used to reroute from the message back to the according template (which is always the same) but with a new message.

Code style

We highly recommend to take a look into the Java code conventions provided by Oracle itself: http://www.oracle.com/technetwork/java/codeconv-138413.html

Understandably it isn't easy to get rid of coding habits once learned, but it makes the code much more readable for others, when it follows certain guidelines.

Be also very careful with your naming. Sometimes you sacrifice readability to save you a few characters. Like in the view you use often `${_}` to refer to objects of a list. Although this works, it wouldn't be much effort to simply give it a useful name.

Documentation

There is no documentation, neither Java Doc nor a README file nor any other sort of manual or documentation. Accordingly it is not possible to review that.

Tests

Obviously there has been done major refactoring (or even rewriting) without adapting the test cases to the new code. Currently they don't even compile and this problem can't be solved by simple renaming all model class names. It would take up too much time to make them runnable.

It is clearly a bad sign, if there are no runnable tests for the new code. Of course it can happen, that some tests aren't adaptable that easy to new code and there isn't enough time to fix it. In that case at least the following two things should be done:

- **Make all the code compilable**
  Otherwise the whole test suite of the play framework isn't even executable. For example there are probably some Selenium tests working (assuming from the last commits), but there is no way to run them out of the play framework, due to the compilation errors.
- **Mark not working test methods accordingly**
  With the `@Ignore` annotation from JUnit there is the possibility to simple ignore specific test methods, because they weren't fixed yet to whatever reason. There is also then the possibility to mark them with comments, so that somebody remembers to fix them.

All in all there isn't a way to perform a review on the testing of the application due to the lack of working tests.

Details

In the following chapter I point out some details within the code. Usually they are commented directly in the code base.

Design of the model

- Some implemented logic doesn't come naturally to mind, why it was implemented at all. The calendar constructor throws an exception if a calendar with the same name and user already exists. There seems to be no technical reason for that, as jpa uses unique ids anyway. If it is though somehow necessary or even a business requirement, it might be better to implement that in the jpa tables itself as a unique key. (`ModCalendar.java; line 25`)
- This exception then gets an issue later. It gets surrounded with a try-catch-block within the method `addEvent`. But in the catch block it only returns null, which can't be considered to be exception handling. That is really bad! If you get to a method call, which throws an exception, either throw it or handle it, but currently it doesn't either! (`ModCalendar.java; line 62`)
- If you have a delete-method which doesn't certainly delete an object (e.g. because it doesn't exist) let the method return the state of success as a boolean. So the method `delEvent` would return true, if there was an event to delete and false if it didn't find an event. (`ModCalendar.java; line 73`)
- Use the possibilities provided by the framework! The play-framework offers great ways for validating data and you can even write your own checks. It has been shown once during an exercise lesson. (`ModEvent.java; line 27`)
- Think about which attribute have to be unique. For example there is no check if the username is unique. This gets a problem with the method `getUser(String user)` as the method just returns the first user it finds and doesn't care if there are more users with the same name. (`ModUser.java; lines 13, 71`)
- I'm actually not sure about this one, but be careful which kind of constructors you use. For the user there exists one with just the username and the password and one which also includes the birthday. Why exactly those two and no others? We usually create constructors with all the required attributes and set the others directly by a reference to the instance variable (if they're public) or by setter-methods. Afterwards we save the object. (`ModUser.java; line 44`)

GUI

The GUI itself doesn't look that appealing except of the calendar presentation which looks pretty neat. There are some minor glitches with the gui, for example if you try to enter an event with a end date prior to the start date (the form looks weird then); or the register form which has some issues too with the indentation of the forms.

Also the interface itself has some issues. For example is the id of an event displayed in the list of event for a certain day, although the id is only used internally and the user has no knowledge about it. And then isn't there even a column header. The same is true for the value of privacy which is just displayed as “true” or “false” but it isn't obvious what this

value stands for (and the values “true” and “false” are rather technical terms which shouldn't be used in a GUI anyway).

Coding Style

- If you already have the id of a persistent object, use that advantage if possible. The method `getEvent(Long id)` of the calendar class return an event of the calendar if it is contained by the calendar. But the method iterates over all the events of the calender, but because we have the id already, we don't need to to that. A better solution would be:

```
public ModEvent getEvent(Long id) {
     ModEvent event = ModEvent.findById(id);
     return event.calendar.equals(this) ? event : null;
}
```

(ModCalendar.java; line 91)