ESE Cross-Review for Team 3

reviewed by Team 8

Note: We had a problem opening the project: first nothing worked at all, because the SpringData.xml was missing. Renaming the SpringDataTest.xml to SpringData.xml only fixed part of the problems, so we had to look for an old version of a springData.xml in the git repository.

Design

- Violation of MVC pattern

  MVC pattern is not violated, model and controller are clearly separated

- Rich OO domain model

  Good Object Oriented models, maybe just for the StudyCourse.java, the Faculty could also be an object.

- Clear responsibilities

  Due to a good choice of names for classes and so on, it is easy to understand what each object does, as a result, responsibilities are quite clear.

- Sound invariants

  There are no invariants in the Project, but since they used javax.validation.constraints, this doesn't seem necessary.

- Overall code organization & reuse, e.g. views

  Just a lot of imports that aren't used and even some variables and autowired fields. There is also part of the code in "comments" which could have been deleted.

Coding style

- Consistency

  The code is clearly written, and in a consistent way. Classes which belong to the

same domain (Controller, Service,...), have an identical approach, which simplifies later changes.

- Intention-revealing names

  The class, method and variable names are chosen in a good way, and describe what they are doing.

- Do not repeat yourself

  In the java classes, there wasn't any big repetition, and even in the JSP section, only the editDone.jsp, notutorfounf.jsp and submitPage.jsp could maybe be put together, and Display a text given by the responsible controller.

- Exception, testing null values

  The exceptions are all captured and handled, but they still use tests for null values.

- Encapsulation

  Encapsulation is fine, the variables are private and accessible through getters and setters. Also their management is kept in the class which is nice.

- Assertion, contracts, invariant checks

  Some rare assertions are used, but most are replaced with javax.validation.constraints which also is a good choice in my opinion since the project uses the Controllers and Views to display the errors. But except this there is no invariant check.

- Utility methods

  There is one utility method for the controllers which is used in two controllers, so it's a good choice.

Documentation

- Understandable

ListHelper doesn't have any documentation. No doc on jsp's either. Some mistakes like "informations" but generally well understandable. Maybe some more punctuation could help, e.g. when telling what the returned ModelView contains, write:

"param" - it does that and that;

"param2" - it does this and this;...

Maybe mention when an error occurs, for instance in the SearchController write how the searchForm cannot be filled in correctly. Also in the "SearchController: what is a "feem"? -> Eclipse has a spell-checking function, use that. Good doc on ControllerIntegrationTest, although a bit complicated.

- Intention-revealing

In the EditForm, I'd mention what values to edit, same with information in RegisterForm, TutorForm and TutorEditForm. TutorFormService doc may be a bit vague. Almost no doc on tests; sometimes there are only normal comments instead of javadoc.

- Describe responsibilities

Be more specific with model docs; these comments are worth nothing (tutor model - "models the tutor", that's clear); same for ClassesDao doc; maybe write what this Dao is specifically used for.

- Match a consistent domain vocabulary

The vocabulary is adequate. The right class names are used, the parameters are described and the return variable is specified. Nothing to criticize here.

Test

Remark: We haven't been able to run the integration tests properly, we get exceptions and errors which stop the run of the tests. We think it's due to our problems with the springData.xml file.
All service tests run properly though, and all pass.

- Number/coverage of test cases

  The unit tests have an average coverage of *78.9%* of the relevant classes (pojos and service, excluding SearchService), measured with 'EclEmma'. While the tests concerning the services easily achieve more than 80%, the e.g. TutorEditForm class is merely covered by 55.4%.

  The method `public TutorEditForm(User user, Tutor tutor)`
  is not being tested at all. A start could be to test this method to increase the coverage.

  The number of controller tests is ok, there's a test for every controller except the IndexController, which has basically only one small responsibility. A simple check of a working RequestMapping would be nice. The coverage (not possible to measure) seems to be fine, as every integration test checks all features and responsibilities of a controller.

- Clear and distinct test cases

Every test consists of understandable cases, none of which are repeating or overlapping with other cases. The vast majority of cases is compact and short, which makes their purpose very clear. In the TutorFormServiceTest there is a test method called `CorrectDataSaved()` which is almost 70 lines long and contains multiple if/else conditions and is thus way too big. The size and structure of this case makes it very

unclear to see what this is about, there should be intentions to split this up. Putting the creation of the courses and tutors into separate methods would already be helpful.

- Easy to understand the case that is tested

Generally speaking, yes, this is the case and well done. Both unit and integration test cases show through the name of the case which functionality is being tested.
There is maybe a little room for improvements in the SearchControllerIntegrationTest, where it could be defined in either the name of the cases or a comment if it's a valid input, e.g. `submitSearch()` could be extended to `submitValidSearch()`.

- Well crafted set of test data

Nothing to improve here. Test data is complete, well-chosen and understandable.

- Readability

The names of the objects, methods and tests are understandable, the formatting is good and the structure of the tests allows a good overview of the test cases as well.
However, there seems to be a lot of commented code which harms the readability:
At the end of the EditControllerIntegrationTest there's a whole test case which has been put in a comment:

```
// TODO Find out how to add a form to a mockmvc request or how to add course and classlist as parameters
      // so that this test can be completed
      /*
      @Test
      public void editTutorDone() throws Exception{....lines....}
```

This should've been taken out for the v1 release and only be implemented once the problem's been figured out. There's also another TODO as well which should be removed in the EditFormServiceTest. The not working test 'RegisterControllerTest' should also not have been part of the release, even if it's all commented. In the TutorFormServiceTest there's a randomly commented line within the test:
`//nextClass.setStudyCourse(courses.get(i%2));` that should be fixed.

In-Depth Review of the SearchController

- Responsibilites and Design

This SearchController class has the right amount of responsibilities, it only manages requests concerning the search. The retrieval of all the courses and tutors is done by the searchService and not directly by the controller, which is well done and complies with the MVC pattern.

- Documentation

While the methods are very well documented, there could be some more general class documentation at the top. Maybe list all basic responsibilities and add things like @author, @version etc.

- Coding style and Readability

The naming, style and formatting of the code is good. Using the final strings like

```
public static final String PAGE_NORESULTS = "noSearchResults";
```

was a good idea and makes the readability great in the methods.

- Conclusion

Very well written controller class with the right amount of responsibilities, mostly sufficient documentation and great coding style and readability.

Controller Class Analysis: RegisterController.java

The RegisterController is responsible for the creation of the User and the Tutor. For this it @Autowired 2 Services and 2 Daos. Because of that, the controller is larger and a bit more difficult to understand than the others. This could be handled by 2 separate Controllers, which would shorten a bit this Controller.

Also in updateListsForTutorForm(...) and submitTutorForm(...), the modification of the tutorForm is exactly the same, and could be handled in an Helper Class or a Service.

But overall, the Controller seems well designed and since Tutor is related to Student, it can be legitim to put them in the same Controller.