Project review of Stutor

General project

Design

Violation of MVC pattern

In general, there is a clear separation between model, view and controller, e.g. the views don't contain any tasks which belong to the controller. There is one little violation in the Lecture model, where an `InvalidGradeException` is thrown, because exception handling belongs to the controller (or a service).

Usage of helper objects between view and model

The general layout is decent, there is a division into controllers and services. The idea of using a factory to create notifications is nice, but what is implemented isn't the factory pattern known from P2, which the naming would suggest.

Rich OO domain model

The domain model is clear and understandable, every model stands on its own and makes sense, except the difference between subject and lecture. The name suggests they are synonymous, but the subject is the name of the major or degree course which causes some confusion. "Timelaps" is also a word, which would better be named differently (proposal: TimeSlot), since this word doesn't exists according to the dictionary.

Clear responsibilities

The search controller has more than one responsibility. It doesn't just handle searching but also has something to do with displaying profiles ("hiddenProfile") which seems to be a responsibility of the `ProfileController` or a not yet existing one. Furthermore sending notifications shouldn't be a responsibility of this controller.

The `SearchController` should have all responsibilities of searching. The differentiation between basic and refined search should be made in different services. The `RefindedSearchController` might then not be needed anymore.

Sound invariants

There aren't any invariants, but they don't seem necessary in the Spring context.

Overall code organization & reuse, e.g. views

There are several examples in the views, where code could have been reused:

- It would make sense to merge "hiddenProfile" and "profile" since they both display a user profile. A method could be used to check if the viewer is the profile owner. If that's not the case, hide the sensitive data.
- "studentmain" and "tutormain" are the same except for two lines. `<c:if test="${user.isTutor}">` could be used to determine which data should be displayed.
- Includes could be used to reduce duplicated code in the JSP-files.

The basic code organization is fine, packages are used to distinguish the different parts of the application. The organizations of the JSP-files could be improved a bit to get a better overview.

Coding style

Consistency

The consistency of the logic is good, that means that if a similar problem occurs, it is solved always the same way. The formatting of the code is clear and as we have learned it in P1 and P2.

Intention-revealing names

Some names don't tell much about what they really do, for example the method `addLectures()` in the `TimelapsController` has nothing to do with lectures because it displays a `TimeLapsForm`. A better name would be `displayTimeLapsForm()`.

Another example is the method `rateTutor()` in the `RatingController`. It suggests that a tutor is rated, but in fact, it's the rating form which is displayed.

The names comment and rating are used in the model Comment and handled in the controller `RatingController`, which is confusing.

Do not repeat yourself

In both, the `SearchController` and the `RefinedSearchController`, there are implemented methods to load the lectures, universities, subjects and genders and to add them to the model. The methods for the lectures and universities are also used in the `AddLectureController`. These methods are looking exactly the same in all three classes, so it would probably be easier to create a `SearchService` which you can access from both controllers and get the data.

A second example is the method `index()` of the `AfterLoginController`. It has an if-else-clause with three lines in both clauses, where two of them are exactly the same.

Exception, testing null values

Exceptions are caught where we found it necessary, without having tested every possibility. Testing for null values is only done very rarely. An easy way to possibly change that is to assert that all the parameters are not null right at the start of each method and to write that down as a contract.

Encapsulation

We haven't found any violations of encapsulation, all class variables are private and accessible through getters and setters.

Assertion, contracts, invariant checks

We haven't found explicit assertions, contracts or invariant checks. Maybe assertions would make sense in the services due to the fact that they should be reusable.

Utility methods

As mentioned before, duplicated code could often be eliminated by using helper methods. Furthermore, some large methods (e.g. `saveStudentFrom` in the `SignUpServiceImplementation`, `create()` in the `RefindedSearchController`) should be split in several smaller methods.

Documentation

Understandable

Most of the comments are easily understandable. Sometimes there was a copy-paste error for example in the `lecture` model, the comment for the university also starts with "subject[…]". Some

comments seem to be rushed as they contain spelling errors or have to be read twice to be fully understood, the `SignUpServiceImplementation` class comment for example.

Intention-revealing

A clear method name would often replace a comment, e.g. the method `newAccount()` in the `SignupController` wouldn't need a comment if it were renamed to `displaySignUpForm()` because then it's intuitively clear what happens in the method and you can save two lines of comment.

A good example of a method naming is the method `saveTutorRating()` in the `RatingController`, where the comment could probably be left out.

Describe responsibilities

Almost all comments describe what the class or method is for, but there seem to be no or very few contracts.

Match a consistent domain vocabulary

The comments are consistent and use the domain vocabulary.

Test

Clear and distinct test cases

Most test cases are labelled clearly and no test cases are overlapping. There are a few cases, which are misleading or unclear. For example the `SignupServiceTest` contains a `testStudentCredentials`(), which creates a Tutor profile and not a Student profile.

A bit confusing are the several Test suites. There are 3 different `ServiceTestSuite` without clear distinction between them. One `ServiceTestSuite` would probably be enough. Instead of naming them `ServiceTestSuite` 1 and 2, it might be clearer with names like `UnitTestSuite`, `IntegrationTestSuite`, etc.

Number/coverage of test cases

Running a coverage test (eclEmma) showed an overall coverage of 64.9% (only considering `src/main/java`). This is a relatively low coverage and seems to be the case because there aren't enough tests checking for all the possible thrown exceptions. At first some errors occurred because the database is named the same way as the one from the skeleton project which created some conflicts if that database was pre-existing and filled with data. It would be advised to change the database name (to StutorDB for example) in the springData.xml configuration file in the property "jdbcUrl". There were also some compilation errors because eclipse didn't like an `@Override` annotation in 3 different classes, which had to be removed to get all the tests running.

Easy to understand the case that is tested

Each test case is small and only one specific method/action is tested.

There are two test cases (`TimelapsServiceTest {getDayTest, getHourTest}`) which are not easily understandable, what they are for and what the conclusions of these tests are.

Well-crafted set of test data

For the available test cases the used test data is adequate. The `NotificationControllerTest` has a helper method to create a new tutor, but for one of the two tutors, this helper method isn't used for some reason.

Also when a new test data entity is created, the id of the entity is set to a negative value. This is done right before the entity is saved, which then has no effect, since the id is automatically generated (as defined with the @GeneratedValue annotation in the model) when saving the model with the DAO.

Readability

The tests are easy to read in all tests, though the white spacing differs in several test classes, which could be improved to lead to better readability. This is only a minor readability hindrance though.

Class analysis: AddLectureController

The Controller auto wires four different DAOs. The use of this many DAOs in the controller might be a sign for too many responsibilities.

All the logic inside the `@ModelAttribute` Methods could be swapped to a Service because the Service should be the middleman between the Database and the Controller.

For example:

```
@ModelAttribute("lectures")
public List<Lecture> allLectures() {
	List<Lecture> allLectures = new LinkedList<Lecture>();
	Iterable<Lecture> lectures = lectureDao.findAll();
	for (Lecture lecture : lectures) {
		allLectures.add(lecture);
	}

	return allLectures;
}
```

Could be like this:

```
@ModelAttribute("lectures")
public List<Lecture> allLectures() {
	return someService.getAllLectures();
}
```

If this approach is implemented consistently, the controllers become more readable and understandable but nevertheless the class does what it's supposed to.