Review for team 7

Design

- Violation of MVC pattern

No violation of the pattern was found.

- Usage of helper objects between view and model

Many helper objects including interfaces are being used. Some of these helper objects contain functions that are not used (for example getUserById in UserService.java) and should therefore be cleaned.

- Rich OO domain model

The domain model is very rich. It comprises many characteristics of a tutor (Address, Grade, TimeSlot) and it also provides a fine-grained definition of an academic course (University -> Course -> Subject).

- Clear responsibilities

The responsibilities, even if usually not explicitly stated, are clear for all the classes.

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

The code seems well organized with clear packages distinction.But some unused code was kept in the project which should be removed (invalidTeamException)

Coding style

- Consistency

The code, written by different authors, is quite consistent. The use of the brackets is mostly uniform and the spacing is regular. Only in very few cases there is, for example, some inconsistency in the number of blanks of an assignment (in ProfileController line 60-70).
Messages to the user are communicated sometimes in German and sometimes in English.
The naming of the interfaces and implementations in the service package is not perfectly straightforward:
IRequestService -> RequestService
IUserDataService -> UserService
SearchService -> SearchServiceImpl

- Intention-revealing names

The naming of the variables and of the functions is in most of the cases concise and clear. The method naming in the request controller is rather unclear (requestGet, myRequestGet, myRequestAction, requestAction)

- Do not repeat yourself

No case of repetition was found in the javacode. The jsp pages are sometimes repetitive. The options for timeslot selection are in editprofile and timeSlotRow, if you want to add something you have to change both.

- Exception, testing null values

Only a couple of exception were implemented. Sometimes you catch exceptions and do nothing with them, which seems very strange, like in profileController getSignUpForm(). If its really needed, comment it.
Null values were not usually tested for and also not assert to be not null. The documentation doesn't tell for most parameters that they shouldn't be null.
You use exception for bad user input, although bad user input isn't exceptional. Although its well implemented you may want to find a way around it. A problem with that use of exceptions is for example that you cannot show multiple errors at once (like empty name + wrong password verification)

- Encapsulation

Variables and variables management is mostly kept inside the class to which they pertain. But you forgot some private modifiers for example in RequestForm for example

- Assertion, contracts, invariant checks

No assertion or invariant checks were found in the code although you implicitly make often the assertion that something is not null: Try to make them explicit. (see also Exception, testing null values) In general, the documentation is not extensive enough to provide proper contracts.

- Utility methods

I did not see any utility class or method.

Documentation:

- Understandable

The documentation is not too extensive (mostly one-liners), but it's usually exhaustive and almost all functions have a javadoc. Annotations like @param and @return are often present, but some of them are left blank (see ProfileController().profile()). None of your services has documentation. For someone not knowing your project more documentation would definitely be helpful.

- Intention-revealing

The naming is very often enough to understand the intention of the object.

- Describe responsibilities

The responsibilities are not explicitly stated for every object, but the efficient naming makes them easy to understand.

- Match a consistent domain vocabulary

The domain objects are referred to in a consistent way.

Test:
In general, there were way too few test so most things said here don’t help if you do not implement more tests.

- Clear and distinct test cases

The test cases present are all clear and distinct.

- Number/coverage of test cases

The coverage of the test cases is not vast, though. Only two of the three services are tested and no controller is tested which is way too low.

- Easy to understand the case that is tested

The name of the test makes it clear what the aim of the test is.

- Well crafted set of test data

Only some unit-test are present, no integration, system or acceptance test was implemented.

- Readability

The readability of the tests was satisfactory.

Pick a class of choice from the controller package and analyze its code. Does the class have too many responsibilities ? Is there some logic that should be moved to another class ? If so, why?

ProfileController is the controller responsible for editing and displaying the profile data. In theory, these two functionalities could be split in two separate controllers. But it is also reasonable that they are in the same class, since they are closely related and well integrated.
The functions to get lists of universities, courses and subjects also appear to be misplaced, since they don’t relate to the responsibilities of the class.