Sprint Retrospective - Iteration #5

Group 58 Week 5

Microservice	User Story (from User Stories doc. numbering)	Task #	Task Assigned To	Estimated Effort (in hrs)	Actual Effort (in hrs)	Done (yes/no)	Notes
Room-scheduler	Teacher/4	Schedule the lecture in the smallest room out of those that have the required capacity	Nikolaos	1	1	yes	Improved the scheduleNewLecture method
		Same year lectures should not overlap	Nikolaos	2	2	yes	Improved the scheduleNewLecture method
		Implement controller method for making room slots: 1) available 2) occupied	Nikolaos	2	3	yes	I had to fix some issues with these two implementations. E.g. if a lunch slot is used for cleaning, it is now correctly marked with ‘occupied’ value ‘3’ when scheduling a lecture and back to ‘2’ when canceling that lecture.

		Use UTC as default timezone for Room and Room Scheduler application	Nikolaos	0.4	0.4	yes	There were some issues with local timezones. Now UTC is used for these two applications.
		Use Amazon Web Service (AWS) for hosting our database	Nikolaos	2	2	yes	We still connect to the ewi server but since there have been many problems connecting there lately, we can change to Amazon if needed.
		Implemented tests for RuleController	Ee Xuan	2.5	2.5	yes	Wrote tests to get new rule, get all rules, edit rule, delete rule.
Room		Fixed the room controller test with mocking the authentication	Ee Xuan	1	3.5	yes	It took so long to figure how to mock static methods, but once this was figured out it made it easier for everyone to write tests.
Authentication		Test the authentication service	Alexandru	5	5	yes	I wrote tests for the user and roles entities, controllers and services with high coverage but there was not enough time to also test the methods in the authentication controller.
		Add authorization to all services	Alexandru	3	10	yes	This took longer than expected because the code had to be refactored a few times to get rid of some errors we ran into, such as the application crashing due to exceptions being thrown.

		Helped Alexandru with improving the authorization of the different services	Alex	2	4	yes	The issue was that the endpoint methods didn't return a relevant error code when not authorized. It turned out that we had to change what these methods return to allow for changing the error code clients received.
		Make a shared module with the authentication class that every service can use	Alex	1	3	no	The authentication class was duplicated along all the different services, I tried to create a separate subproject that all services could use so that this duplication wouldn't be necessary. After it taking way longer than expected (much like all the other gradle stuff I worked on during this project) I gave up as I saw no use in spending several more hours without having a guarantee to make it work.
Courses		Test coursesController methods	Luca	5		no	I did not manage to do this yet, because of other courses that required my priority. This will be top priority in the last days of the project
		Add authentication in coursesController	Luca	3	3	yes	It is difficult to test if it is implemented correctly, because of database problems, but for what I've tested it worked.
		Fixing the one to many relationship	Michel	2	3	yes?	This one was more difficult than I expected, kept getting issues with hibernate and gson. But I think I figured it out and it works?

		Wrote Junit tests for Lecture	Ee Xuan	1	1.5	yes	This took a little longer than anticipated because I had to add the equals and hash method for courses to test.
		Wrote Junit tests for Course	Ee Xuan	1	1	yes	Took slightly longer than expected because I had to add 2 methods and test these.
		Wrote Controller test for Lecture	Ee Xuan	0.5	1	yes	This took longer to do because I had to update all the Junit 4 annotations to Junit 5 annotations.
Student		Letting the student set a preference whether he wants to go or not	Michel	1.5	1	yes	Well this one was relatively easy, it was just creating a function to change the preference of the student and then save it.
		A student is able to see all it's courses	Michel	3	3	yes	This was easier to do, due to reusing some code of getting all the lectures. So it was reasonably doable.
		A student is able to see all the lectures he has sorted in ascending order.	Michel	4	7	yes	This was the most difficult one, as I had to take into account both online and offline lectures. We stored all lectures in general and all the online lectures for a student specifically. So I first added all the on campus lectures I then also made a list with all the online lectures. Then I merged both lists with a

							merge helper function.
		Write some tests for the student service (unit as well as mocking using mockito)	Alex	2	2	yes	Writing tests always takes longer than you first think, because of that I wasn't able to add as many tests as I would have liked.
NA	NA	Helping Ee Xuan with figuring out how to mock static objects	Alex	1	3.5	yes	We kept having issues with gradle and the mix of junit 4 and junit 5 which led to all kind of issues

Main problems encountered this sprint

1. Often throughout the week, we could not connect to the database (too many requests error)
2. Uneven distribution of commits. Some team members have way more commits and lines of code than others. This is something we tried to resolve during the final days by letting members with many commits help the other members via collaborating through voice chat.
3. Some testing still relied on the actual database. We worked on setting up a temporary db for testing purposes and let the tests use an in-memory database where it was possible.

Main problems encountered throughout the project

1. There were quite a lot of issues with gradle, especially in the beginning when we wanted to change the structure of the project to allow for a single services folder with a subfolder for each service, it took way too much time to set this up properly which led to a slow start.
2. Throughout the project we have had several connection issues with the ewi database we used for some services. Early on in the project we could not get the local database to work for everyone so we thought the ewi database would be a good solution, but this often meant that we could not properly test our changes as there were too many connections. In the future we need to spend more time getting local databases to work, as dealing with connection issues is a waste of time.

3. During the project the workload was not balanced very well. Both the division of work between members as well as throughout the week. The latter issue was solved by starting with our tasks sooner and communication to the other team members when we ran into issues as soon as possible. In the last week we really tried to fix the issue of imbalance in commits between members, as members with more commits took a step back.
4. There were not many merge requests, and the merge requests that were present were mostly near the deadline of a sprint. We tried to resolve this by starting on our work for a sprint sooner and thus being able to make the merge request sooner. This also meant that other members had more time to review the merge request without being pressured by a deadline.

Requirements reflection

Generally we have stuck to the requirement division as the requirements with the highest priority have been implemented. When it comes to the could haves and should haves, in hindsight there are a few requirements for which a swap of category would have been better. For example, the CRUD operations rooms, courses and students would have been useful to have in the tests that use these entities, yet these operations were of low priority. When it comes to lecture invitations that students could accept or decline, we were probably too ambitious when we thought of this requirement, as the main functionality of the system was more involved than anticipated which meant that we definitely did not have enough time to finish a requirement like that. Most of the 'edit' requirements (like editing lectures/courses or moving a lecture online when the teacher has symptoms), have not been implemented for this same reason: the main functionality of the system took more time than expected.

Must haves

1. The system shall schedule lectures before the quarter starts
2. The system shall assign students for the coming two weeks for lectures on campus, prioritizing students that haven't gone to campus the longest
    1. The date a student last went to campus is stored in the Students table
3. Rooms shall only be able to use a percentage of their original capacity (#seats)
    1. For rooms with at most 200 seats, this percentage is 20%
    2. For rooms with 200 seats or more, this percentage is 30%
4. Rooms shall have a grace period
5. Rooms shall be stored in a database with the following information:
    1. Id

    2. location
    3. #seats
6. Teachers shall be able to create their courses before the quarter starts
7. Teachers shall be able to create lectures for their courses before the quarter starts
    1. Lectures shall be given on the day of the week that the teacher specifies
    2. Lectures shall take as many timeslots as the teacher specifies
8. Lectures from courses from the same year shall not overlap
9. Lectures shall be scheduled by the system depending on
    1. The day of the week
    2. The number of timeslots the lecture occupies
    3. How many students will be attending the lecture on campus
10. Users can authenticate themselves (using their netID)
11. Students shall have at least one lecture per two weeks on campus
12. Students shall be able to get a list with all their lectures
    1. This list shall be ordered in ascending order

Should haves

1. Students shall be able to globally specify whether they prefer online or on-campus lectures
2. Teachers shall be able to inform that they have symptoms
    1. The lecture will be held online, the on-campus lecture will be canceled
3. Students shall be able to specify per lecture whether they want to attend the lecture on campus or not
4. Students shall be able to accept or decline an invitation for an on-campus lecture
5. The system shall have a configuration table in the database containing:
    1. Timeslot duration
    2. Percentage of students allowed for the total room capacity
6. The scheduler shall prevent overlap of courses from the same bachelor

Could Haves

1. Users shall see how many students are attending a certain lecture, and how many of them are coming to campus
2. Students shall get notified about upcoming lectures
    1. If they were scheduled for an on-campus lecture, they can decline
3. Canceled lectures (for example due to the teacher having symptoms) shall be disregarded from the students' on-campus lectures
4. Admins shall be able to create, edit and delete rooms
5. Admins shall be able to create, edit and delete courses
6. Admins shall be able to create, edit and delete students
7. Teachers shall be able to edit their courses
8. Students shall be able to see all the courses they are enrolled in.