Sprint Retrospective, Iteration #2

Microservice	Issue	Assignee	Effort	Done (y/n)	Notes
Cluster	The cluster service contains its own database, with a repository for nodes and a repository holding the schedule	Alan Kuźnicki	4h	Yes	
	The cluster service contains a scheduler which accepts incoming job requests and schedules them	Alan Kuźnicki	8h+	Yes	It turned out that scheduling was essentially half of the service's functionality and was very interconnected with every component. Several services and design patterns (builder and strategy with three implementations) were used to tackle this monumental task, but it works.
	The nodes that users contribute are assigned to a faculty	Alan Kuźnicki	3h	Yes	Uses a strategy to assign the node to a faculty. Two implementations were coded.
	The cluster should have better testing coverage, with a focus on the controller and integration testing	Alan Kuźnicki	4h	No	Testing is not done until we reach coverage that we deem satisfactory. The controller has, for now, been relatively well tested, however.

	An approved request is received from the request service and processed	Alan Kuźnicki	1h	Yes	
	The request service gets the available resources for a given faculty for specific days	Alan Kuźnicki	1h	Yes	
	The cluster microservice must have a node class that handles operations related to nodes	Ariel Potolski Eilat	20m	Yes	We ended up creating a NodeBuilder class due to the high number of fields in the node class.
	There must be three types of resources that the system distinguishes between: CPU, GPU, Memory	Ariel Potolski Eilat		Yes	This was implemented during the creation of the Node class.
	A node contributed by a user should have at least as much CPU resources as memory or GPU resources.	Ariel Potolski Eilat		Yes	This was implemented during the creation of the Node class.
	A user must be able to contribute new nodes to the cluster	Ariel Potolski Eilat	45m	Yes	Added an endpoint that adds a new node to the system. Contains various checks, and uses the check regarding the amount of resources implemented in the issue above.
	A user must be able to remove nodes that they contributed to the cluster	Ariel Potolski Eilat	45m	No	The first part was completed, which is simply removing the node when the user calls the delete endpoint. The second part - removing on the next day- will be completed on the next sprint.

	There must be a controller for the cluster microservice	Ariel Potolski Eilat	1h	Yes	Added a controller with some endpoints
User	A user should be able to change it password	Bart Coster	5h	Yes	It took way longer because updating the database did not work for me. So I found a work around.
	A user must be able to remove a faculty form a user	Bart Coster	2h	Yes	
	A user must be able to see their assigned faculties	Bart Coster	1.5h	Yes	
	A user must be able to assign a faculty to a user	Bart Coster	5h	Yes	
	The request microservice should be able to ask for faculties a user is employed to	Kasper van Duijne	2h	Yes	
	The user should be notified when their job is delayed and the reason for that	Kasper van Duijne	5h	Yes	Entire notification system, with endpoints, spring boot service, and tested in code and manually.
	The user should be notified by the cluster service about the job's status	Kasper van Duijne	3h	Yes	Only needed to change a few lines to have the endpoint support notifications from the Cluster Service.
	Testing user service	Kasper van Duijne	8h	Yes	Took a long time to debug IntelliJ and problems running the tests.

Request	Core functionality of the JobRequestController class must be tested	Alexander Andonov	5h	Yes	The essential functionality of the job controller is tested and functions as discussed.The send requests and send approvals mapping are correctly implemented; also the endpoint for the pending requests is tested as well.
	A user should be able to have multiple outstanding requests at the same time	Alexander Andonov	2h	Yes	Our original implementation having requests as a user needed only a bit of tweaking, but it mostly worked like this already. Just had to make sure the logic is implemented properly and is congruent with what other services would expect as behavior.
	All methods in RequestAllocationService must be tested	Alexander Andonov	3h	No	The core sending and pending requests are mocked and tested, but can be tested more thoroughly for specific cases. Furthermore, the methods that need a response from the cluster have to be tested through mocking that we didn't have the time to figure out.
	An authenticated user must be able to send requests through the facility they are assigned to	Vladimir Pavlov	5.5h	Yes	Implemented the "pipeline" for the requests to always check if the faculty matches and is able to be handled through the respective faculty.
	An approved request is forwarded to the cluster to be scheduled	Vladimir Pavlov	2h	Yes	Endpoint sending a request for the cluster for handling.

	A faculty should be able to request the current reserved resources for the specific faculty from cluster	Vladimir Pavlov	3h	Yes	The Requests service requests information about the resources the cluster has assigned for the faculty, therefore the endpoint is established for communication.
	A request must be handled by a faculty account	Vladimir Pavlov	3h	Yes	Establishing the core logic of a faculty to work with a request by a user in that faculty.

Problems encountered

1. Very late merging caused issues with conflicts, evaluation, and delays
    1.1. Description: due to us only merging at the very end of the sprint, the merge requests were extremely large, many merge conflicts popped up last-minute, and we not only faced many difficulties in evaluating them but were also delayed with the final merge to main.
    1.2. Reaction: it was too late to change so we worked through it but we should make a difference in approach for the next sprint. We will focus on making smaller, more frequent merge requests directly to dev.

2. When trying to update an object in a repository it will not change when using “save”.
    2.1. Description: when implementing the password-changing functionality, we realized that using repository.save() did not allow us to alter existing objects.
    2.2. Reaction**:** deleting and then again saving worked so we used that to fix it. In the future we will strive to avoid such issues using our newfound experience with Java’s libraries.

3. Unevenly distributed workload
    3.1. Description: sprint 1 was unproductive in terms of coding results. We only started having results when actually programming and that did not happen until sprint 2, which means the time has not been evenly distributed over the past 2 weeks.
    3.2. Reaction: from now on we clearly know what the issues contain and what the assignments entail, and how long they should take to implement, so we can evenly distribute work over the coming weeks.

Adjustments for the next sprint

1. We will try to plan ahead and distribute the work more evenly across sprints.
2. We will open merge requests more frequently and directly to dev to ensure that we can easily evaluate others' work and merge calmly and effectively.
3. We will strive to immediately test our work, to avoid having to cram testing entire microservices in half a day.
4. We will be more involved with each other's services to maintain cohesiveness of the project and avoid splintering into "subprojects".