# ASSIGNMENT 1 PART 1

Software Engineering Methods


Group 11b

Alexandar Andonov

Bart Coster

Kasper van Duijne

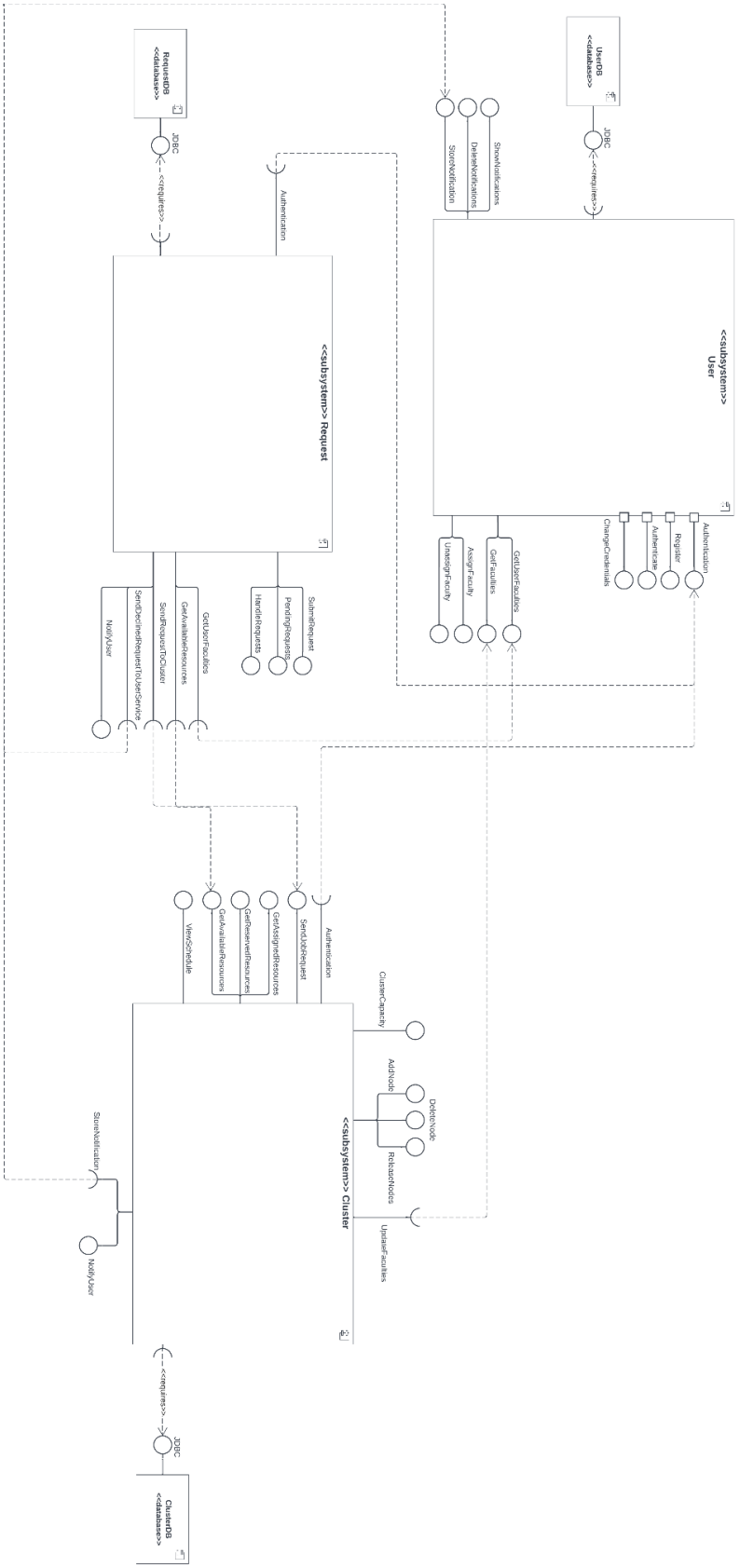
Ariel Potolski Eilat

Alan Kuźnicki

Vladimir Pavlov

# 1. Contents

# 2. Overview of the architecture



# 3. Bounded contexts

## 3.1. User context

We identified the user as a separate context, containing authentication and authorization, information about users and faculties, as well as a history of notifications. Therefore it handles registering and authenticating users, incoming notifications directed at the user from another context, keeps track of users assigned to faculties, and handles changes to user faculty relations. It is a core domain, as it's necessary not only to grant access to the system to registered users, but also to allow them access to status information regarding their requests.

## 3.2. Request context

We classified the request logic as a separate bounded context, as it is tangible enough on its own to not be contained within the user or cluster contexts. Furthermore, while the cluster keeps track of the nodes and scheduled jobs, and the user of the users, faculties, and notifications, neither seem to be better equipped than the other to also take care of requests sent by users. The scope of the request service only contains request functionality, such as approving or rejecting requests. It is a core domain of the system, as without it, the supercomputer can only exist without actually performing any operations.

## 3.3. Cluster context

The cluster is the last bounded context we identified, understood by us to be the supercomputer itself. Therefore, it handles the supercomputer functionalities, such as receiving nodes from users, managing and updating the amount of resources available, and managing the schedule of the jobs, as well as granting schedule- and node-related information to authorized users. Moreover, the cluster should output information to the other bounded contexts, such as the amount of resources available, or a job notification. The cluster is a core domain of the system, given that all functionalities need to, somehow, go through or interact with it.

# 4. Mapping to microservices

## 4.1. User microservice

The bounded context is mapped one-to-one to the microservice. This makes the user microservice responsible for assigning the faculties to the users and keeping track of which user is employed at which faculties. Furthermore, it has an endpoint where other microservices can send all the notifications that the user microservice will store. It also provides authentication and authorization services, to ensure that the users accessing the system have the right credentials For authorization, we opted for four roles with distinct privileges: employee, faculty employee, system admin, and system (for when a microservice itself is contacting another microservice with no user input.) It has its own database, which stores both all received notifications (history), but also registered users and their assigned faculties.

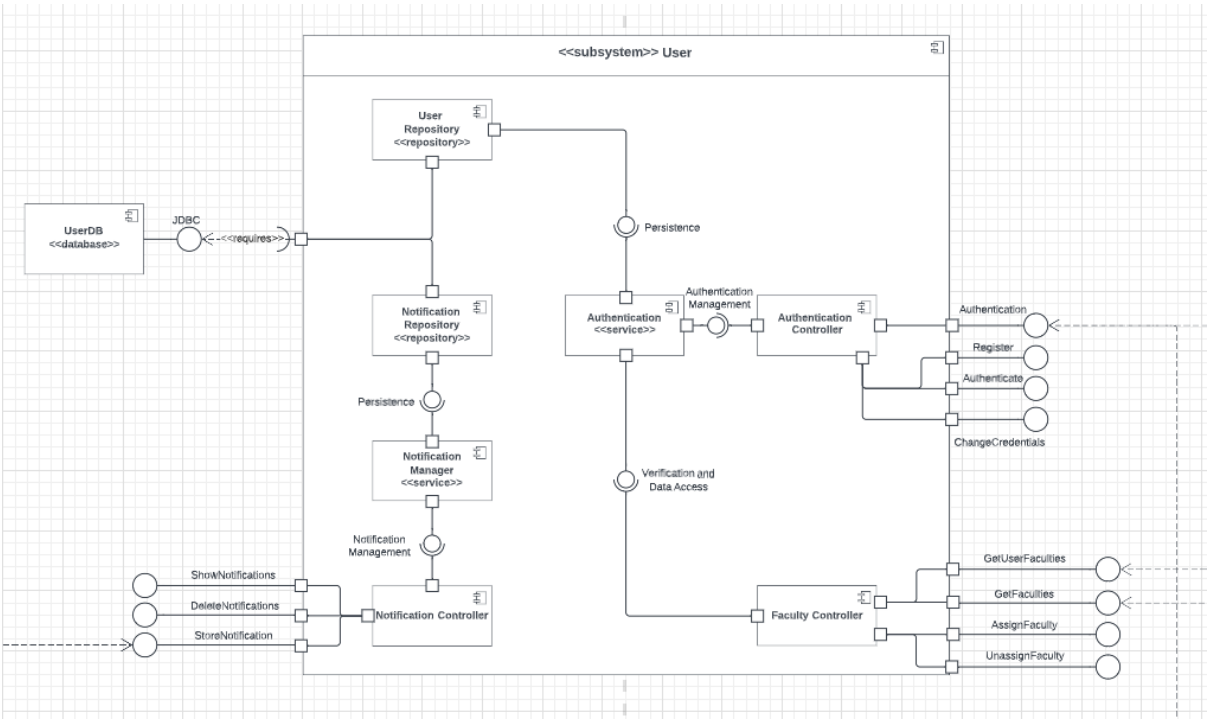
## 4.2. Request microservice

The request bounded context is focused solely on request functionality, which is why we chose to represent this context through a single request microservice. It implements the functionality which facilitates the communication between the user and the cluster/supercomputer. This microservice acts as the intermediate interface through which users can make job requests to the faculties where they are members. The microservice receives data from the authenticated users and information about the available resources that the cluster will allocate to the faculty. Its database stores all the requests made by users.

## 4.3. Cluster microservice

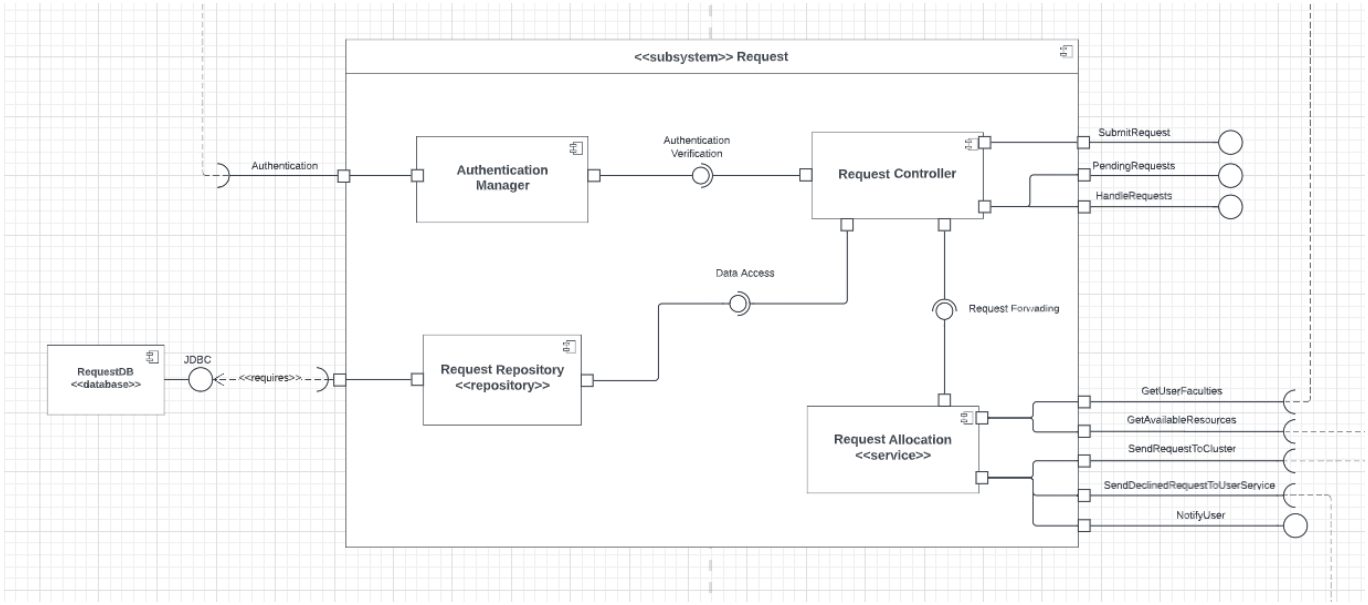
We decided to implement the Cluster context as a unique microservice. As said before, this microservice will be able to handle requests related to node control, addition and removal. Besides that, this microservice will handle the creation and management of a schedule for all jobs that are sent to it by the request service. Additionally, it will grant access to a plethora of information related to its internal structure (nodes and resources, as well as their assignment to faculties,) and the schedule of jobs. All of these will require sufficient credentials, as mentioned in the requirements document. Moreover, it is going to notify the user whenever the status of their jobs changes (e.g., it is delayed) and send those notifications to the user service. It will have its own database, storing all nodes currently in the cluster in one repository, as well as the schedule of all jobs since the service was started in another.

# 5. Responsibility and role in the infrastructure



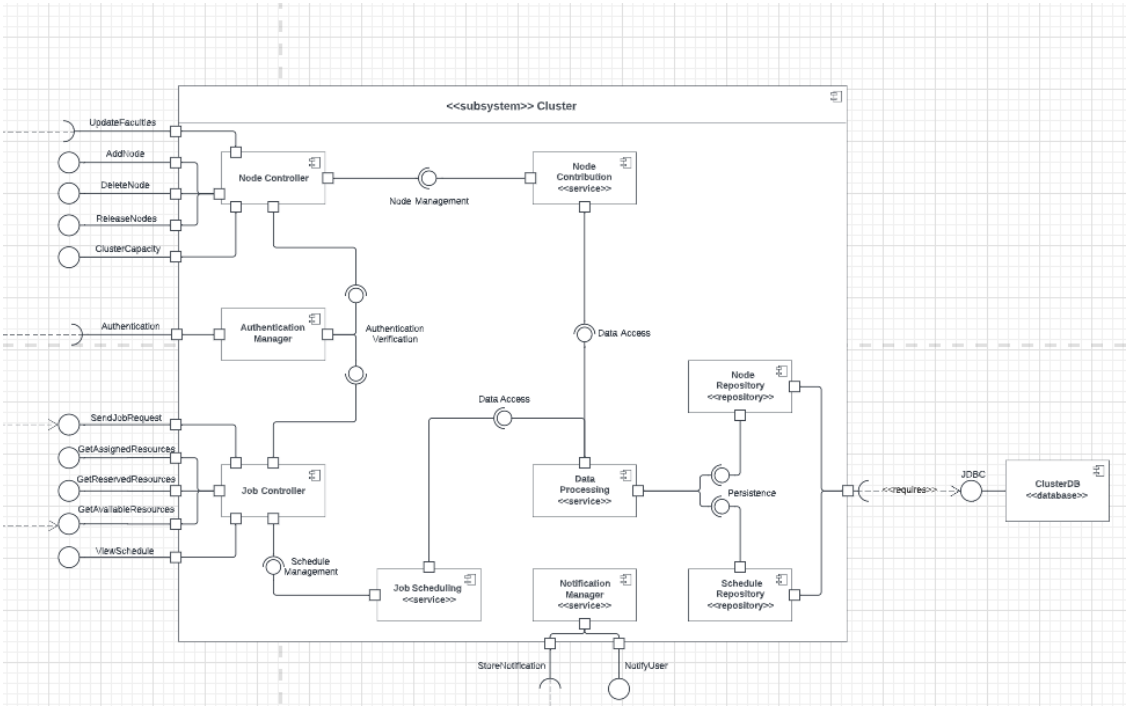

## 5.1. User microservice

1. Employee-faculty relations, functionality to verify user's ability to send a request to a given faculty ("GetUserFaculties", called by the request service; "AssignFaculty" and "UnassignFaculty", accessed by system admins to control which user is assigned to which faculty)
2. Authentication and authorization, registering and logging in, changing one's credentials ("Register", "Authenticate", "ChangeCredentials")
3. Provides the other services with the ability to limit their functionality to authenticated and authorized users ("Authentication", connected to request and cluster) by giving authenticated users a token which is verified by all services
4. Provides the cluster with the knowledge of all existing faculties ("GetFaculties", sending the information to the cluster)
5. Notification history ("ShowNotifications", "DeleteNotifications") as well as ability for the other services to store notifications for users in its database ("StoreNotification")
6. Database containing a history of all notifications, as well as all registered users and their assigned faculties and credentials




## 5.2. Request microservice

1. Verifying that a user is authenticated and authorized to perform an action (“Authentication”, service provided by the user microservice, although the verification itself is local)
2. Provides a user with the service of submitting a request to a faculty of their choice (“SubmitRequest”)
3. Checks if a user belongs to a faculty they are submitting a request to (“GetUserFaculties”, provided by the user microservice) and if enough resources are available for the given faculty in the indicated time period (“GetAvailableResources”, provided by the cluster)
4. Users with faculty privileges are provided with the service of seeing all pending requests and being able to approve or reject those for their faculty (“PendingRequests”, “HandleRequests” - can be specified which requests are to be approved)
5. Forward approved requests to the cluster (“SendRequestToCluster” - the service to do this is provided by the cluster, which is why we made this a socket)
6. Notify the user and store notification in user service’s database (“NotifyUser” and “SendDeclinedRequestToUserService”)
7. Database stores all requests submitted to the service along with their status




## 5.3. Cluster microservice

1. Accepts a list of all existing faculties from the user service (“UpdateFaculties”)
2. Provides all users with the service of adding nodes to the cluster (“AddNode”, assigned to a faculty automatically if not specified) and request the removal of any of their nodes (“DeleteNode”). This removal only takes place at midnight, when the service checks if any jobs need to be delayed/dropped and performs rescheduling
3. Allows a system admin to check the cluster capacity per node (“ClusterCapacity”) as well as remove certain nodes with immediate effect (“DeleteNode” with the right credentials)
4. Provides a faculty account with the service of “releasing” some of their nodes for a given period of time, which are added to a free pool (“ReleaseNodes”)
5. Verifying that a user is authenticated and authorized to perform an action (“Authentication”, service provided by the user microservice, although the verification itself is local)
6. Accepts a job request from the request service and schedules it automatically (“SendJobRequest” - a lollipop as we believe that it is the request service that requires the cluster service to implement its functionality, not the other way around)
7. Gives information about available resources to the request service (“GetAvailableResources”) as well as extensive access to schedule- and resource-related information for system admins and accounts with faculty privileges (“GetAssignedResources”, “GetReservedResources”, “GetAvailableResources”)
8. Allows the system admins to view the complete schedule (“ViewSchedule”)
9. Notifies the user of any status change of their job (“NotifyUser”) as well as stores the notification in user service’s database (“StoreNotification”)
10. Database contains all nodes currently in the cluster along with their resources, contributing users and assigned faculties in one repository, as well as the complete schedule in another.

# 6. Rationale and motivation

## 6.1. User service

The user service is a well needed standalone microservice. This is because employee-faculty relation data is now stored in a separate microservice. The authentication is also a crucial part of this service, and fits well into the existing service, as it handles a project user to user-microservice communication also. Thanks to that, all user-related information and functionality is kept separate from the rest of the system, as are sensitive data such as passwords.

## 6.2. Request service

We have chosen to separate the request service from the rest of the infrastructure, as communication would become extremely cumbersome and convoluted if we lacked this middleman structure, which filters the flow of requests to the cluster by allowing faculty-authorized users to directly control which requests are fulfilled using their faculty's resources. Moreover, it would be functionally impossible to integrate this microservice into any other, as it would introduce either a lot of overlap or skew the balance greatly towards either service, defeating the purpose of this style of architecture.

## 6.3. Cluster service

Given that the cluster has an important role in the system, and it is responsible for a big share of its functionalities, we decided that it should be a service itself. This is supported by the fact that the cluster carries concepts that are fully unique to it, such as nodes and the schedule. Finally, we believe that creating a microservice to handle the cluster's functionality will ease the parallelization during the development of our project, as well as provide a better structure and organization for our implementation.