Architectural Draft

“For each microservice, you should briefly describe its responsibility and its role in the overall infrastructure. Furthermore, you should provide motivations for the architecture and each individual microservice. Of course, there are multiple alternative ways to cluster different functionalities in different microservices and how the context bounds from DDD are mapped into these microservices. We want to understand why you chose your design over alternative ones.

The document must be at least 3-pages long but not longer than 4 pages (A4 format); diagrams are not included in the page count.”

Index

General goal of the system

The main functionality of the system is to allow employees to add requests for resources of the supercomputer. The requests need to be accepted by faculty reviewers if a given faculty has available resources. From such a description, one can easily identify main bounded contexts: User, Faculty and Resource Manager. Each user needs to be authenticated before accessing the system; additionally, the user should be notified in case one of their jobs is dropped. Therefore, we can add two more bounded contexts: Authentication and Notification Manager.

Explanation of each microservice (function and internals)

Each microservice has its own functionality and role in the bigger system. These are the following microservices we have chosen to implement:

1. User

The user microservice will store the base details of the user object. This includes the user's netID as a unique key, the faculties they belong to, and what role they have (Employee, Reviewer, or SysAdmin). It is this microservice that application users will first be in contact with, as well as being the microservice that allows them to make requests to faculties for Delft Blue's resources. They will also have the option to review any requests they have sent in the past. It is important to note that upon starting, the User must log in with their NetID and password, which will be forwarded to the authentication microservice for validation.

2. Authentication

The authentication microservice will be used to ensure that any user that signs in to the system is in fact a legitimate user. It will store the combination of the netIDs and passwords for all users, and can compare them with the user's input. This microservice will also be responsible for generating tokens upon authentication of valid credentials, and verify whether past tokens are still valid.

3. Faculty

After a user makes a request, it is forwarded to the faculty microservice. For each of TU Delft's faculties, this stores a list of pending requests from all Employees that are yet to be processed, as well a list of accepted requests. In the situation where the User is a reviewer, they get directly forwarded to this microservice, and are given the ability to look at the pending requests, and decide whether to accept or reject them. If approved, the request is forwarded to the resource manager for further approval, otherwise the user is informed that the request was denied.

4. Resource Manager

As stated in the name, the resource manager microservice is responsible for keeping track of resources that are both available and being utilized at a specific date and time, whether it is the CPU, GPU, or memory. It keeps track of the availability of resources for each faculty, as well as the free pool and managing the individual nodes of the cluster.

5. Notification Manager

The notification manager can be polled by users to receive information regarding the status of their request. This is mainly to inform users of the fact that their job was dropped due to the cluster being too low on computing power, but can be extended to also include the state of the review process.

Explanation of our DDD and the choices we made

We considered many different ways of implementing our program and making the Domain Driven Design Diagram, but eventually decided on the current implementation. There are several decisions that led to our current model. Here we will discuss the important ones in greater detail.

1. Letting authentication be a separate subdomain

We debated whether the authentication had separate enough functionality from the user context to justify being a bounded-context of its own. Almost all communication done with the authentication context passes through the user. However, there are also instances where the communication comes from other bounded-contexts, like faculty. Furthermore, the user context was already one of the larger ones in our system. In the end, we decided that it would make more sense for authentication to be a separate subdomain.

2. Letting the notification manager be a separate subdomain

The same question we had for authentication, namely whether it was justifiable for it to have its own subdomain, was also applicable to the notification manager. Just like with authentication, most communication would be through the users context. But again, this subdomain was already quite big, and they had quite separate functionalities. We thought that it would be better if all notifications came through a separate notification manager, to increase the readability and clarity of our system.

3. Where we store the resources and the specific requests/jobs

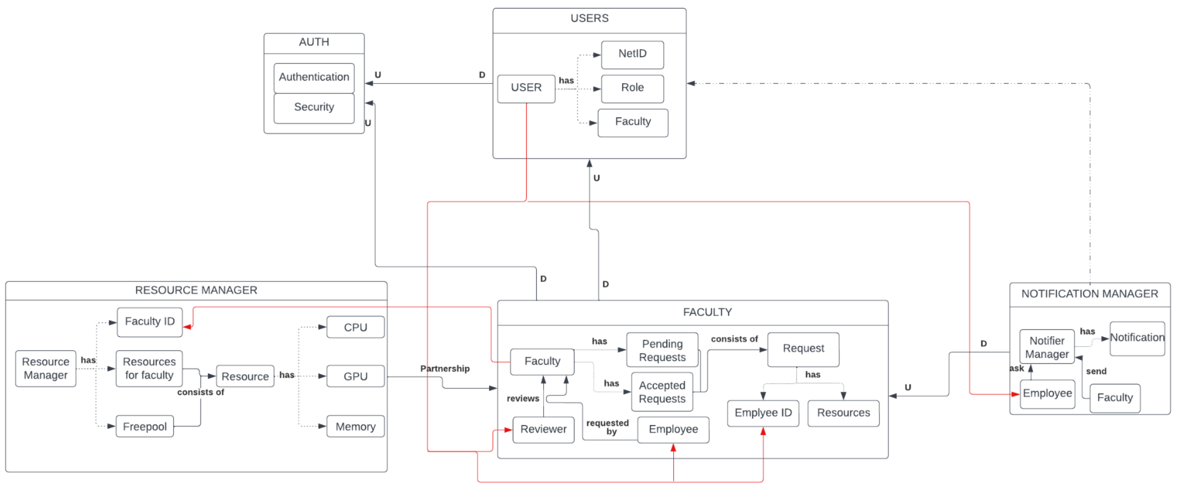
A major decision for our system was the choice of where to store the available resources, used resources, all the jobs, and whether these jobs are already accepted, or still have to be reviewed. We decided on a resource manager quite early in the process, and having faculty as its own context was also decided on rather swiftly. However, it took more time to decide which bounded-context would keep track of which information. Eventually, we decided that the resource manager would keep track of all resources, as its name suggested. The faculty context will keep track of all jobs, accepted or not, because the jobs have to be reviewed by a faculty member.

4. Relationships throughout the system

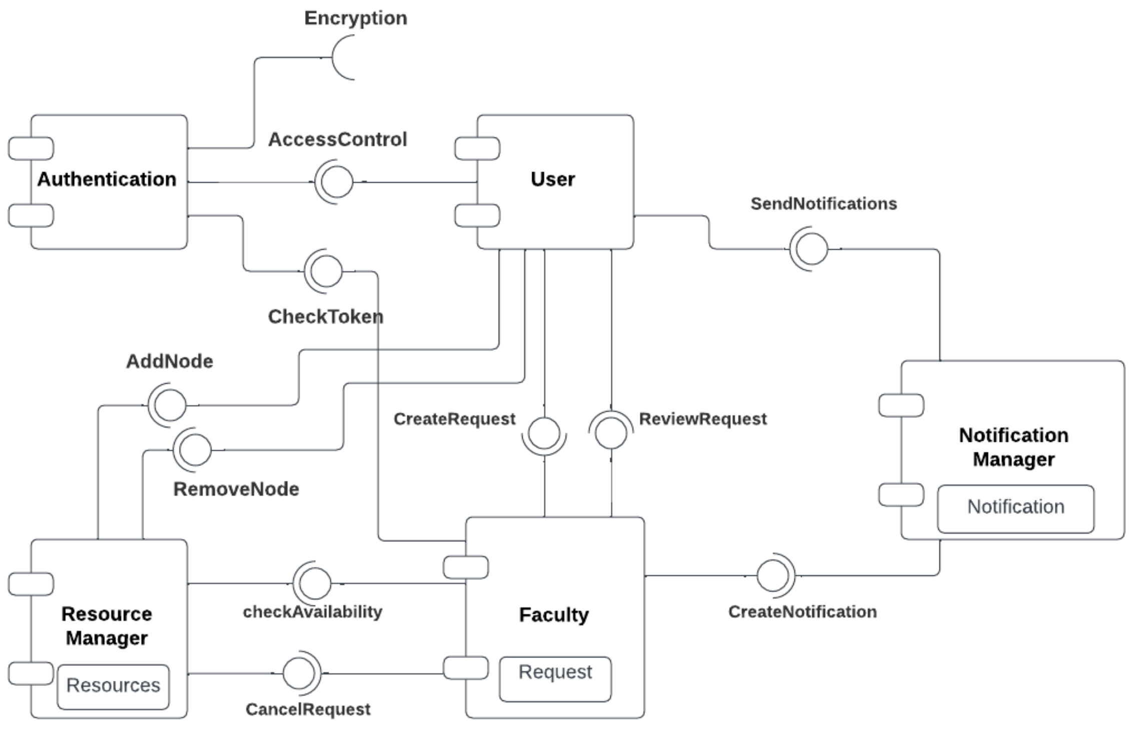
Most of the relationships in our system are basic up- and downstream ones. In these cases, the upstream (U) subdomain only influences the downstream (D) subdomain. However, one relationship stands out, namely the one between faculty and resource manager. This is a partnership. The faculty and resource manager both influence each other, while having their own dependent set of goals. This means that they are in a cooperating relationship, or in other words, a partnership.

5. Not modeling the gateway

Is not modeling the gateway a specific choice or just the way it is supposed to be?



The DDD Diagram



The component diagram