Group-DB-09b-Assignment-1-Part-1

Bounded contexts

In the DelftBlue scenario a scheduling system should be implemented that gives university employees the possibility to request resources for their jobs to be done from the DelftBlue supercomputer. The system can be split into several subdomains: employees, faculty accounts, sysadmins, jobs, clusters of nodes, and the scheduler. Users must be authenticated before making requests. The main function of the system is resource allocation. The bounded contexts of the system include: Users & Authentication (which includes employees, faculty accounts, sysadmins, and authentication), Jobs, Clusters, and the Scheduler.

The following UML Component Diagram shows the organization and relationships of all bounded contexts and microservices.

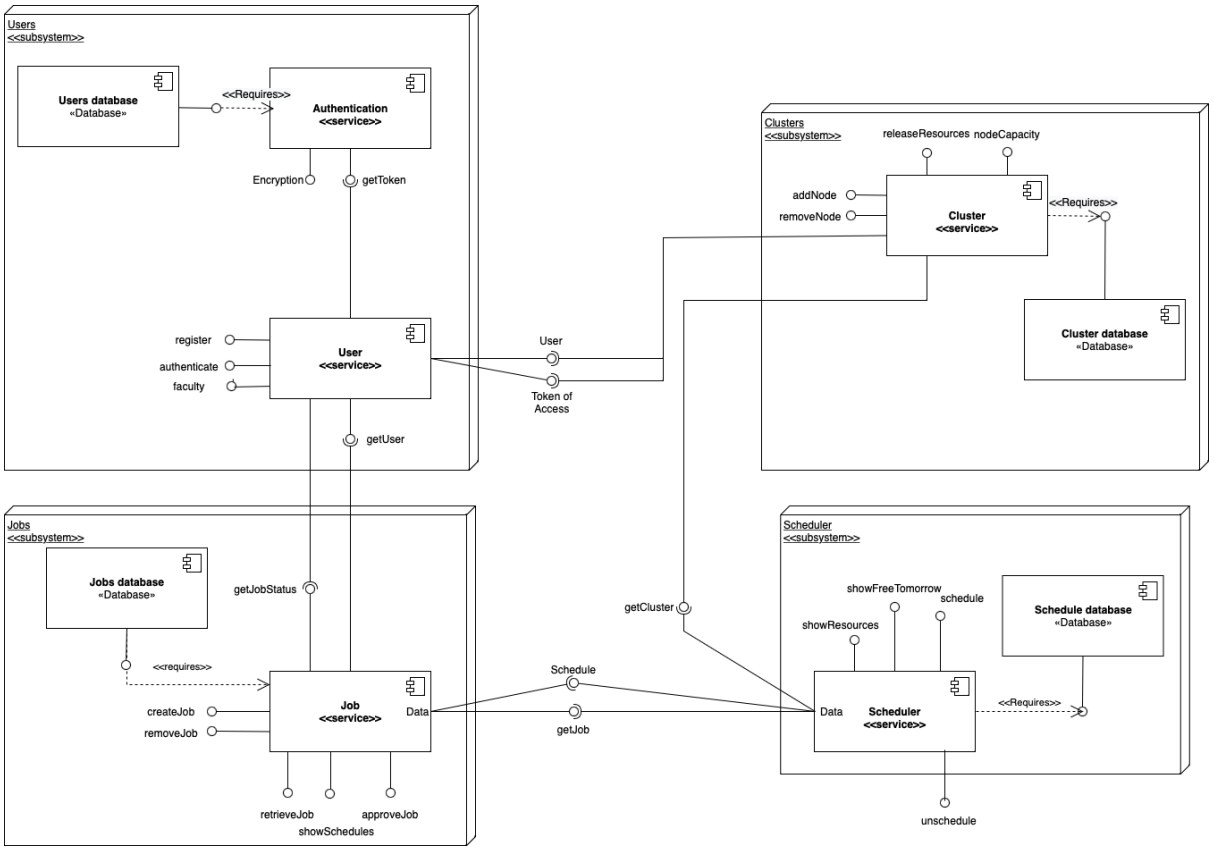

Figure 1: UML Component Diagram

Microservices

How are bounded contexts mapped into microservices?

We have decided that every bounded context will be a microservice on its own. Bounded contexts were constructed in such a way that every bounded context had the most common functionality. We wanted to keep it that way so we opted for this solution of having four microservice: Users&Authentication, Jobs, Clusters and the Scheduler.

Security and token validation

When a microservice receives an API request with an authorization token, then the token needs to be validated, in order to check if the initial request is accepted. If there is no authorization token, the request will be immediately declined. The validation check will be done by an additional request to the Users&Authentication microservice which will validate the token and send their response. Afterwards, if the token is valid, the request can be processed by the microservice. The token contains the information about the netId and the role of the user. Finally, after the token has been verified, the microservice which handles the request still needs to check if the user with his role can make such a request.

Microservice: Users&Authentication

This microservice is responsible for storing all users in the database. They will be assigned roles that will allow them to interact with methods from other microservices. Their authentication is handled by the Authentication module that is built into this microservice. After providing a correct password, each user receives their private token which will be provided as a parameter to microservices that will determine whether or not perform a requested action. All requests will be synchronous since the response is necessary to continue the code execution.

Users with proper roles will be able to create new jobs with the Jobs microservice, where each job must be assigned to a user. Besides this relationship, users are also allowed to add new nodes to the Nodes microservice, where each node must be assigned to a user. Added nodes are automatically assigned to its creator’s faculty. Users will not interact with the Scheduler, as this is the role of the Jobs microservice after a job gets approved.

This microservice was chosen by us, because we identified Users as a bounded context in our scenario. It makes sense for it to be a separate microservice – it is only

responsible for storing and authenticating users, which can then interact with other functionality.

Microservice: Clusters

The purpose of the Clusters microservice is to handle and store nodes belonging to faculties, and a free pool. Each node will have a specified amount of resources that can be later used by the Scheduler to schedule the jobs. If the faculty decides to release some nodes from their own pool, they get updated with the start date and end date of their release and they can be used for jobs from any faculty. Cluster stores the information about the number of nodes and the amount of resources a certain node adds. Employee and faculty accounts can add new nodes for their faculty. If an admin adds a new node, resources are added to the faculty called free pool. The Cluster controller gets the role of the user from the JWT token and the faculty by requesting the authentication service for the faculty of the user.

The Clusters microservice interacts synchronously with other microservices. It provides endpoints for services to be used by the Scheduler and it sends an update to the Scheduler microservice as soon as a node is deleted. Its role is to maintain a database of nodes and allow scheduling based on that information.

We have decided to choose this microservice because there are many nodes to be managed that belong to certain faculties (where one faculty corresponds to one Cluster). We merge all the functionality of the nodes under one microservice to decrease coupling.

Microservice: Scheduler

The scheduler interacts with Clusters microservice to obtain information required to schedule jobs. For each job it synchronously requests available resources. If the available resources are enough to suffice the job requirements, the job is assigned to this particular faculty. In the event that the faculty is short of resources, the scheduler checks whether the free pool has enough resources to make up for the shortage and if so, the job is split between the faculty and the free pool. Otherwise, the job is rejected.

The reason behind choosing scheduler as a microservice is that it acts as a logical connection between jobs and clusters. It provides an interface for the Jobs microservice so that it can easily request a job to be scheduled, while not being concerned about the scheduling logic itself. The Scheduler will perform this action and check with the Clusters microservice if there are resources available needed to schedule the job. This isolated the scheduling logic well from the other bounded contexts.

Microservice: Jobs

The jobs microservice stores requests with their name, description, the amount of resources it needs to use and the preferred day to have it run on or before. Every job is made by an Employee (User). This microservice provides an interface for the Users to schedule a job. Once a request for a job to be scheduled arrives, it makes an asynchronous request to the Scheduler. If the Scheduler is able to allocate resources for the job, it later sends another request to Jobs microservice to update the job's status - it will be marked as scheduled or not. This information will later be used by the Users microservice, so that each user can check the status of their jobs.

The reason behind choosing jobs as a microservice is that it isolates the logic of making a request and scheduling it. Moreover, following the Decompose by Subdomain decomposition strategy, it seems like a reasonable choice as we have a subdomain Jobs.

Architecture of the Code

This project is based on the Domain-driven design (DDD) which is a software development approach that focuses on the business domain. This can lead to more effective communication and better-designed software that is better aligned with the needs of the business. Furthermore, the hexagonal architecture is used in combination with microservices. The hexagonal architecture, also known as the "ports and adapters" architecture, is a software design pattern that can be used in combination with microservices to improve the flexibility and modularity of a system. In this architecture, the core functionality of the system is isolated from the external dependencies, such as databases and user interfaces. This makes it easier to swap out or replace these dependencies without affecting the core functionality of the system. The decision to use the hexagonal architecture in combination with microservices was driven by the requirement to build an application that is easily extendable, easy to integrate with other systems, and scalable.