Functional requirements:

Must have:

- Employees **MUST** be able to submit a request to a faculty that they are assigned to.
- The request **MUST** have a name, description, the amount of resources it needs, the preferred day to have it run on, and the status of the request.
- Employees **MUST** be able to send multiple outstanding requests to a faculty they belong to.
- The faculty account **MUST** be able to review and approve or reject requests of their faculty.
- The faculty **MUST** have a limited amount of “x” resources in a cluster per day.
- The faculty **MUST** be able to decide the specific day an approved request will be handled.
- All users **MUST** authenticate themselves using their unique universal NetID and Password.
- Employees **MUST** be able to only see the available resources for tomorrow.
- There **MUST** be 3 types of resources - CPU, GPU and memory.

Should have:

- An employee **SHOULD** be allowed to be assigned to multiple faculties.
- A request **SHOULD** be run on the preferred date or before.
- A faculty **SHOULD** be able to decide to release their resources for one or multiple days to the freepool.
- A faculty **SHOULD** be able to use available resources from the free pool for that day if all its own resources are being utilised.
- Requests **SHOULD** no longer require approval to be run on the supercomputer 6 hours before the day starts.
- Requests **SHOULD** be accepted on a first come first serve basis until 0 resources are available.
- Requests **SHOULD** be declined 5 minutes before their preferred date starts.
- Requests **SHOULD** ask for at least as much CPU resources as GPU and memory resources.
- The sysadmins **SHOULD** be able to see the schedules and available resources from all days and the current capacity of each node.
- A node **SHOULD** have a name, the URL of where to find the node, the token to access the node and the amount of resources added to the cluster.

Could have:

- A user **COULD** contribute with a node of their own to the cluster.
- The resources from a node **COULD** be added to the free pool.
- Nodes **COULD** have CPU resources >= GPU/Memory resources.
- The users **COULD** take their nodes offline from the cluster.

- A node **COULD** remain in the cluster the day of the removal request.
- A node **COULD** be removed starting from the next day of the removal request.
- The cluster **COULD** drop jobs if it is too low on resources.
- The user **COULD** be notified if his request is dropped.
- The user **COULD** access a node by using a token.

Won't have:

- The system **WON'T** have a graphical user interface.
- The system **WON'T** be connected to the actual TU Delft authentication system (Single-Sign-On).

Non-functional requirements:

- The communication between microservices is made using **Apache Kafka.**
- The system should be built in such a way that it can be extended with extra functionalities later (modular).
- The system should allow for easy integration with other systems (API).
- Individual components of the system need to be scalable so that when the demand for that service increases, the service does not get overloaded (microservices).
- The system should be written in the Java programming language (version 11).
- All interactions of users with the systems shall be handled through endpoints of a gateway
- The gateway forwards the requests to the corresponding microservices.
- The system must be built with Spring Boot (Spring framework) and Gradle.
- The password needs to be stored and encrypted in a safe place
- Store Requests, Users, Nodes.
- TODO: Mention the testing metrics…
- TODO: Mention how communication with gateway works