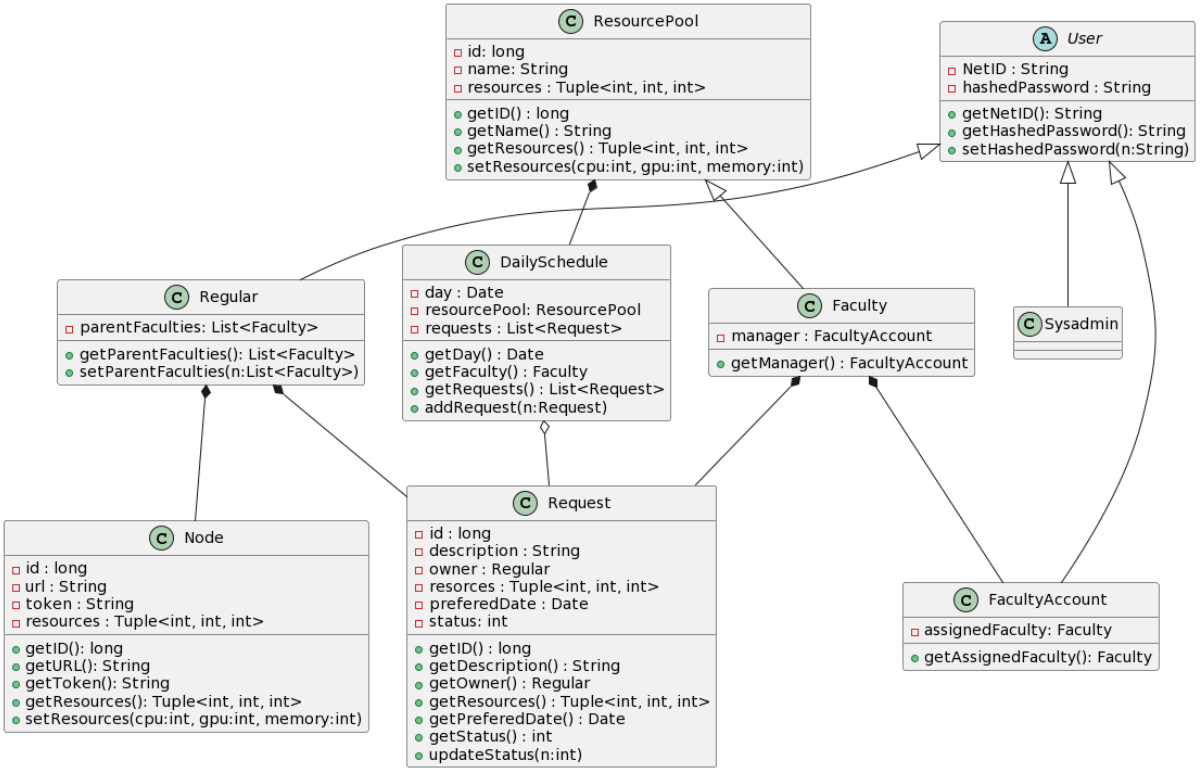
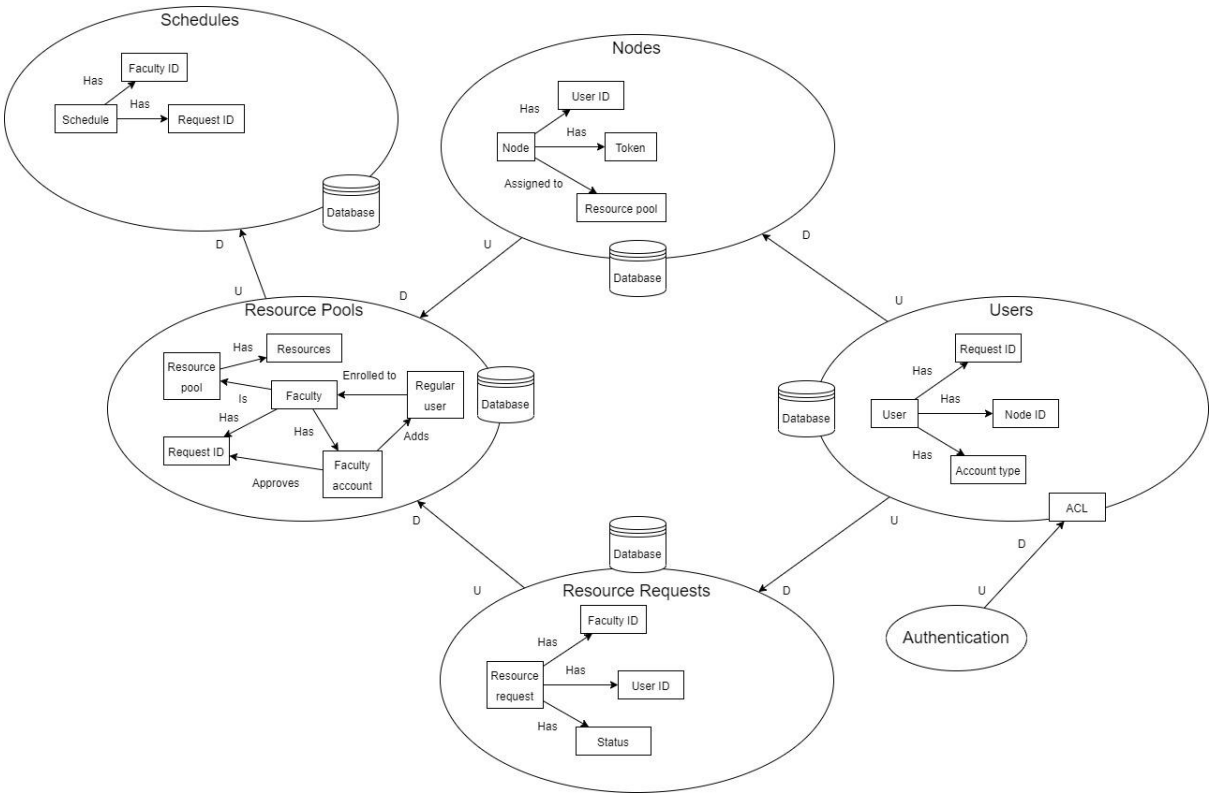
Task 1: Software architecture

Architecture Diagram

Class Component Diagram



- Here the red squares represent private methods/fields and the green circles represent public methods/fields
- Notes on some the relationships between classes:
    - DailySchedules are an aggregation of requests
    - Faculties are composed of a FacultyAccount
    - Requests have exactly one resource pool, a resource pool can have multiple requests
    - Nodes have exactly one owner but a regular user can contribute multiple nodes
    - A daily schedule has strictly one resource pool associated with it but a resource pool can have multiple schedules.
    - A request is done by strictly one regular user but a user can have multiple requests
    - A user is an abstract class with three children, a sysadmin, a regular user account or a faculty account.
    - We chose to make a faculty a specific type of resource pool to increase cohesion. The free resource pool is therefore a resource pool that is not associated with a particular faculty. We however have not found a way to ensure that there is only one free resource pool. We considered making the resource pool abstract and having a free resource pool and faculty extend from that but that would unnecessarily increase coupling. We also considered a singleton free resource pool but that may make the application less modular.



Design Patterns

Design Pattern 1 - Chain of Responsibility

Implementation Description - Can be used for the process of authentication, authorization and approval/rejection when a request is submitted.

Class Diagram

Design Pattern 2 - Facade Pattern

Implementation Description - Can be used to implement a higher level interface that will take in all requests/calls from a user of the system and then forward it to the right microservice. We have requests for resources, calls to read the log about status of requests, requests to offer a personal node, etc. Can be combined with the Chain of Responsibility pattern.

Class Diagram

Design Pattern 3 - Proxy

Implementation Description - Can be used for loading the overall schedule for the sysadmins. As the system scales in size, faculties, number of requests and users, the list of Schedules will grow very big in size, thus using eager function calls to load it, instead of lazy ones could become a bottleneck of the system. Using a lazy Proxy would solve this issue and improve scalability.

Note: Depends on whether we are planning for the system to handle a large load in the future. If this is not expected then we are increasing the complexity of the system for no reason.

Class Diagram