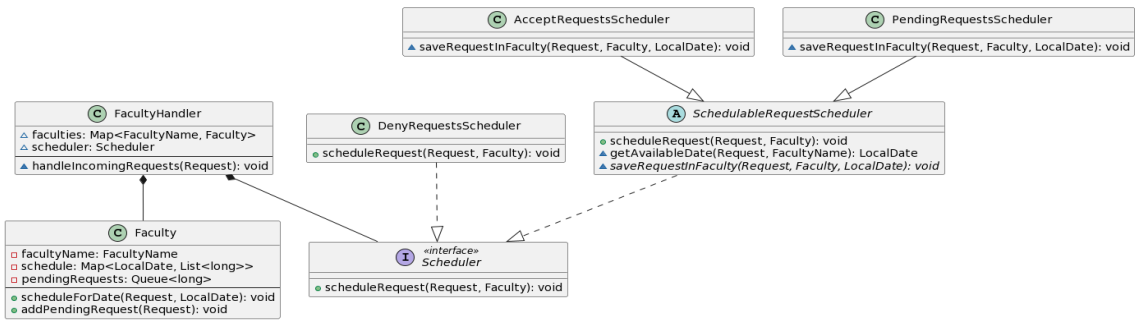
Assignment 1 - Task 2

The Strategy pattern

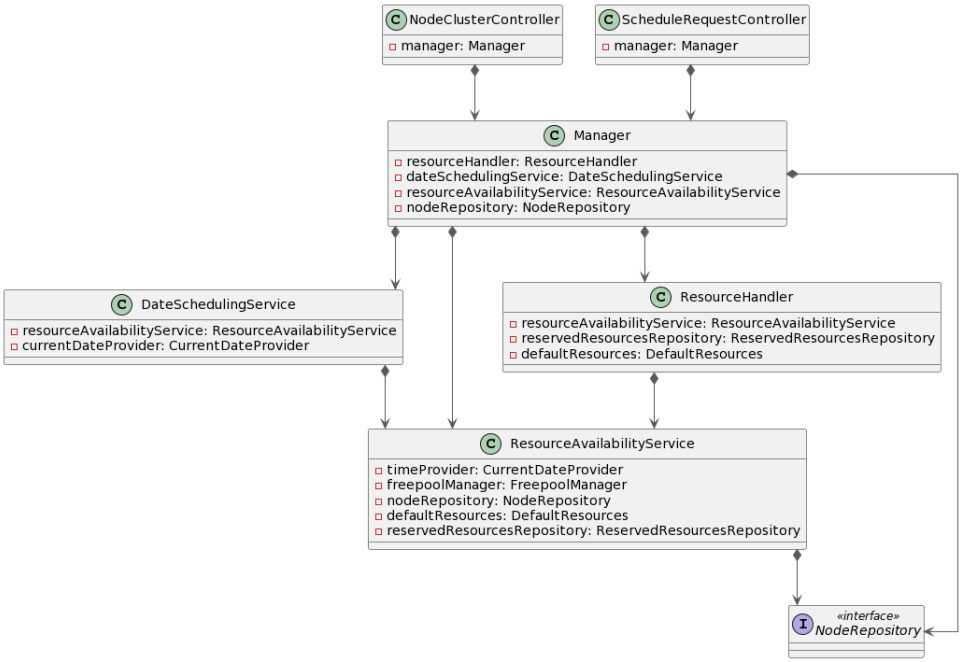
We have chosen to implement the strategy design pattern in the *Faculty* microservice, as it helps us to easily choose between different strategies for handling incoming job requests. When users send a job request through the gateway, we first decide on its validity. This is done by comparing the time difference between when the request was sent, and the day the employee wants to reserve resources in the cluster. We test if the preferred processing date is not within five minutes of receiving the request and if the preferred date is not in the past. If either of these is true, we instantiate a DenyRequestsScheduler(). Here, if the request was already added, it is marked as “DENIED” and deleted from the database. If the preferred processing date is within six hours, but more than five minutes of the current time, we accept the request by instantiating an AcceptRequestsScheduler(). With this strategy, we ask the Resource Manager for a date on which the request can be scheduled. If one is found, the request is marked as “ACCEPTED”, the required resources are reserved, and the job is placed on the schedule of the faculty to which the user belongs. If none of the previous conditions apply, a PendingRequestsScheduler() is instantiated. This one works similarly to the AcceptRequestsScheduler, but it marks the job requests as “PENDING” and sends the request, without reserving any resources for it, to the user’s faculty pending queue instead. Here it will await approval from a faculty reviewer. To minimize code duplication between AcceptRequestScheduler() and PendingRequestScheduler(), an abstract parent class has been created that contains the logic to get the available date from the *Resource Manager*. This allows the child classes to change the status of the job request and store it in its respective place. In the situation where the job requests couldn’t be scheduled for some reason, the user is notified through the *Notification Manager* by this parent class.
The strategy pattern in the *Faculty* microservice can be seen in the class diagram below:



Class Diagram for the Strategy Pattern

The Facade pattern

We have chosen to implement the facade design pattern in the *Resource Manager* microservice in the form of the *Manager* class. This class makes use of multiple services and repositories to provide complex functionality such as scheduling requests, or viewing available resources on a particular day. We have chosen this pattern in order to abstract away the complexity of the *Resource Manager*, and to provide a simple interface to use when implementing communication with this microservice. Since all necessary information is provided by the facade, it is not necessary to access the rest of the logic in the microservice, which makes it easier to maintain the code responsible for communication.
Below is the class diagram for the *Manager* class. Note how the controllers only need the *Manager* to access the functionality they require.



Class Diagram for the Facade Pattern