Design patterns:

- **Factory**: Data returned by the application back to the client (or between the services) might have to be in the future in different formats than JSON (for example XML), so using a factory makes it easy to add new formats in the future.
- **Strategy**: Depending on the type of user, different methods and implementation will be available for them.

Issues:

Must haves:

- User
  - Users have to be authorised when trying to make a request
  - Users can make job requests
  - Assign user to single/multiple faculties
  - Users can have multiple outstanding requests
  - Users must be stored in the database
- Job
  - Users must be able to create new jobs
  - SysAdmins must be able to request the overview of all schedules
  - FacultyAccounts must be able to approve jobs
  - Job service must provide a job instance for the Scheduler service
  - Job service must be able to update the schedule of the jobs
  - Job service must be able to provide their status to the owner of the job
  - Jobs should be stored in the database (so that they can be accessed by the SysAdmin)
- Schedule
  - Job microservice must be able to request a job to be scheduled
  - Schedule must be able to determine whether or not a given job can be scheduled.
  - Schedule assigns a job to a particular faculty or rejects it based (or to the free pool, or both) on the resources and date preferred by
- Clusters
  - Users must be able to add a new node
  - Users must be able to remove a node
  - FacultyAccounts must be able to free the cluster’s resources for specified days
  - Clusters must contain information about the particular faculty (free pool)

Should haves:

- User
  - Resign user of faculty
- Job
  - Users should be able to remove their jobs
- Schedule
  - Job microservice should be able to unschedule a job
- Clusters

Could haves:

- User
- Job
- Schedule
    - Scheduler could be able to reschedule jobs greedily when new ones come in
- Clusters