# Mini Project - Part -1 (period 2)

This exercise starts the mini project briefly described here.
Here we will create the backend architecture, models and facade, used for all the mini project examples, that follows.

**a) Create a basic Express application**, using the generator, setup with your "favorite" template engine (pug, ejs or handlebars)

**b)** Install nodemon (unless you have installed it globally): `npm install nodemon --save-dev` and change the start script in package.json to use nodemon.

**c)** Create two new folders in the root of the project: **models** and **facades**

**d)** Create a new database "miniproject" on your mlab-account and create a user for this database

**e)** Create (in the root of the project) a file dbSetup.js and add the code below, to connect to to your Mongo-database (remember to install mongoose with the --save option first).

```
var mongoose = require('mongoose');
const dbURI = "CONNECTION STRING FROM MLAB";
mongoose.connect(dbURI);
mongoose.connection.on('connected', function () {
  console.log('Mongoose default connection open to ' + dbURI);
});
mongoose.connection.on('error',function (err) {
  console.log('Mongoose default connection error: ' + err);
});
```

**f) require this file** in the top of the app.js file

**g) Creating the Schemas and Models**
Inspired by these six rules of thumb, define the necessary schemas for the mini project inspired by the domain model given in the project description.
( we will do this as a class exercise in the class)

Hints: Each schema must go into a separate file, and the Schema for location, used by User (but not necessarily, by LocationBlog) must be implemented like this (you will see why in a later exercise):

```
const MINUTES = 1;
var EXPIRES = 60 * MINUTES;
var LocationSchema = new mongoose.Schema({
  //Make sure that next line reflects your User-model
  user: {type: Schema.ObjectId, ref: 'User', required: true},
  created: { type: Date, expires: EXPIRES, default: Date.now() },
  loc: {
       'type': { type: String, enum: "Point", default: "Point" },
       coordinates: { type: [Number] }
  }
})
```

**h) Create a series of test data** (some users, some LocationBlogs, and a number of Positions) to verify the behaviour of your schema and models.

**i) Create a facade** with, as a minimum, the following methods:

- `addUser(..)  //Forget about the Position in this part`
- ~~`addJobToUser(..)`~~ `Don't focus on the jobs unless you have a lot of time`
- `getAllUsers()`
- `findByUserName(username)`
- `addLocationBlog (..)`
- `likeLocationBlog(..)`

**j) Write Mocha/Chai tests** to test the facade methods.

**k) Implement the web-pages** necessary to: add a user, let a user "log-in", let a "logged-in" user add LocationBlogs

*Note: If you have implemented the exercises, already given, that provides you with examples to demonstrate server side rendering (with pug, ejs or handlebars) you can skip this part, and alternatively just add a simple Rest API that will make it possible to do the steps from either a SPA and/or a mobile app.*

**HINTS:** I have made a few changes to part i)
To get you started, I have also added some example code to the project we have worked on for the last two days: https://github.com/fulsstackJS-spring2018/miniproject.git

You should observe the following changes:

**In dbSetup.js:** You can pass in an alternative dbURI string (for your test database) See in the top of the makeTestData.js file for how to call it now .

**In userFacade.js**: Observe how all methods returns a promise, and also how to export your facade.

**In test/testUserFacade.js:**

- Read (and understand) the comments on top of the before-methods.
- Observe that the test uses an alternative test database (so create this for your project, and add the connection String in line 4).
- There are three test, which should explain themself (and the facade we are testing). Two should pass, and the third should **fail**, since it tests a method not yet implemented (write tests first)

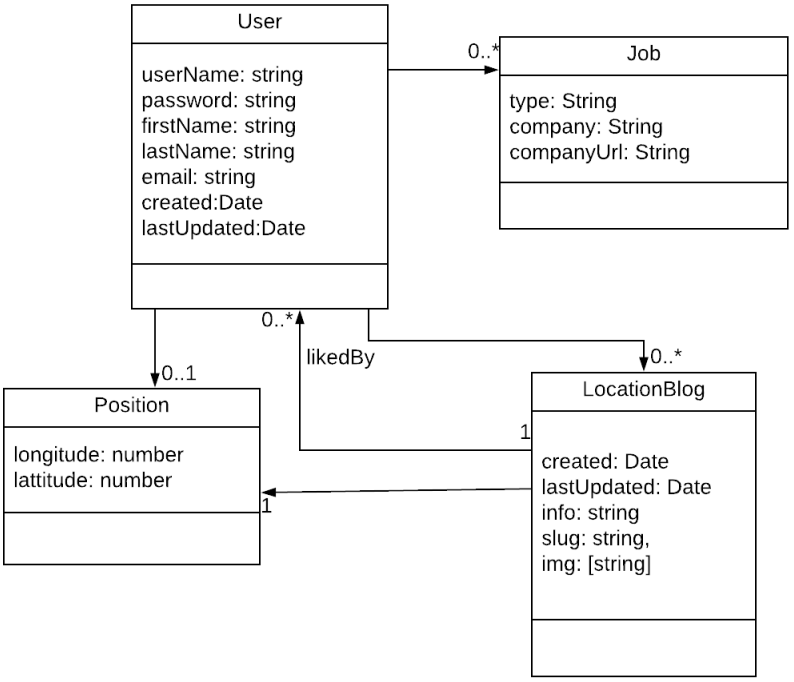
**Important:** Observe how I use async/await to solve the async problems in the tests. Check these slides (four pages), especially the one that focuses on Mocha with promises and async/await.

# Mini Project

To make the technologies we are going to cover throughout the semester a bit more interesting, many of the coming exercises will be focused on a common project idea, as outlined below:

## Model/idea

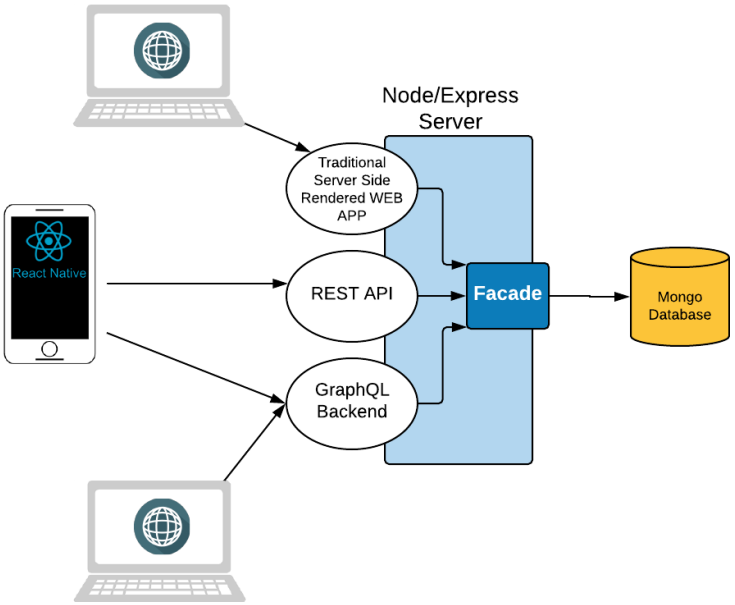
The idea is to have a backend for users, as sketched in this model.
A user can have a geo-position, which will be sent from his phone. The position will only "live" for a short time, in order to be up-to-date. If not updated before this time, it will be removed. Users can also hold several job positions (programmer, student, football trainer etc.)
Authenticated Users, can create LocationBlog-entries as sketched in the figure and, also, authenticated users can "like" a blog-entry (once only).




The model is deliberately held very simple, and miss many parts necessary for a "real" system. The model is also meant ONLY as the initial thoughts, which should be used as inspiration during the Schema Design.

## Architecture (what it is you have to build)

This diagram illustrates what you are expected to build throughout the semester. This is not a single system, but rather three demo systems with a lot of redundancy meant to try out different technologies. Having a common database + facade for all projects makes it easier to get time for all of it.




**Part-1:** The first part represents a traditional web application, built with server-side rendering (pug, ejs or handlebars) and a MongoDB/mongoose database. Via this app, you should be able to log-in, create blog entries, and ideally also watch and like these entries.

**Part-2**: A simple friend-finder app, where you can log-in, via a phone, provide a radius to search for friends, and will be provided with a map, with the position of all your friend (inside this radius)

**Part-3:** Will introduce features from 1+2, this time using Graph-QL, in order to demonstrate the advantages of this new technology