Måns Oskarsson - mnsosk-7@student.ltu.se
Oscar Lundberg - osclun-7@student.ltu.se

Dynamic websystems

Introduction

In the dynamic websystems course we choose to work with MongoDB and Mongoose server side, with node.js and express as backend and middleware respectively. Front end is React.

Instruction

Accessing the server and directory is done by using ssh:

1. ‘ssh -p 25937 mnsosk-7@130.240.207.20‘
   pwd: jetlagg123
2. ‘cd /home/mnsosk-7/mans_serverapp’
3. Done!

Even though it's really bad security practice giving out passwords instead of creating ssh keys we don’t mind.

Manager login

We create manager login directly in the DB for now so:

1. Go to /login and chose manager
2. Email : manager@manager.com
3. Password: manager123

Editing users

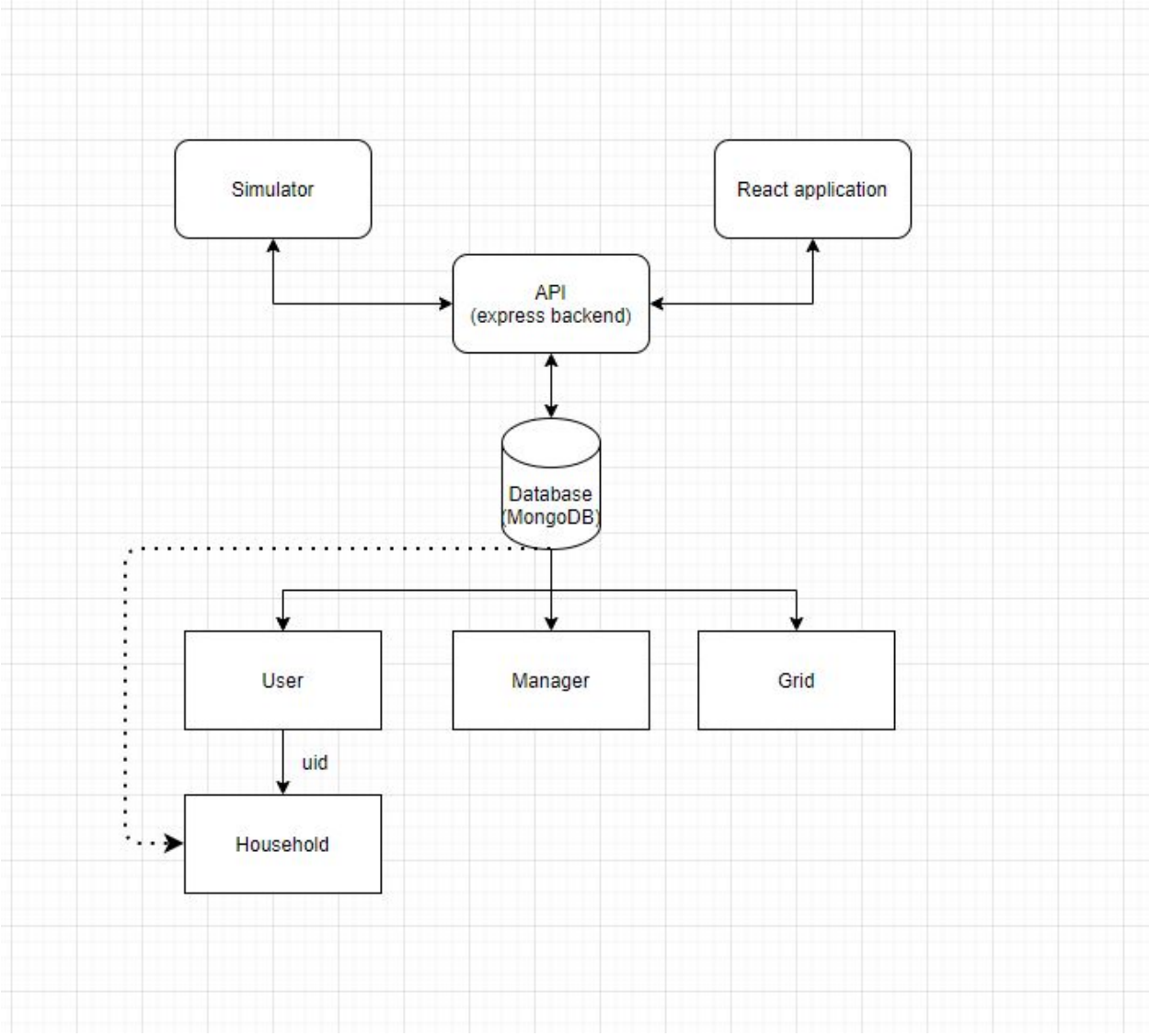
Unfortunately we did not finishing routing to the editing user page, but ALL functionalities are implemented and working. So, to edit a user

1. Log in as a manager
2. Go to the “Users” tab top left or go to ***/url/householdusers***
3. Copy an ID and in go to ***/url/profiles/ID***
4. Edit whichever field and go back to users (step 2)

Måns Oskarsson - mnsosk-7@student.ltu.se
Oscar Lundberg - osclun-7@student.ltu.se

Production

Design



This is ultimately the design we choose where the simulator makes requests directly to the database, with help of the API. The react application does similarly but the simulator in more general terms, is used in order to update values, while the react application has a more visualization approach, presenting the values in the DB for the specific user and handling different user inputs and so on.

Måns Oskarsson - mnsosk-7@student.ltu.se
Oscar Lundberg - osclun-7@student.ltu.se

API

Since we used express you could do different routes and create a sort of middleware functionality with server and API.

http://<ipadress>/api/grid
This section holds the total values from managers and users accumulated in the grid.
It also has a buffer which the manager can charge used by the users, and a price which the managers can change based on the modelled base price (simple function based on current net production).

http://<ipadress>/api/user
This holds all the user credentials, when creating a user, the users specific id is also used by creating a household. Important to notice is that

http://<ipadress>/api/household
This holds all the different simulation data for the specific user, power production, consumption ,house buffer and so on. A specific household is specified with that user's id. Since the user has more data than the manager we thought it was necessary to split between simulation data and credentials for all the users in order to avoid large amounts of data per document.

http://<ipadress>/api/manager (<admin>)
Managers have credentials and simulation data combined. Manager has status (isrunning, stopped), production (constant output to grid) and so on.

Discussion

The reason we chose this approach is because we wanted the simulator and react application to be as separated as possible inorder to get a steady workflow, using modular design. By the end of the project course both team members thought this procedure was efficient by not introducing any strong cohesion between the simulator/react-app but instead updating the database directly from the simulator with help of the API and doing representation (and minor updates) in the react app with help of API.

The drawbacks are first of all that, if there are alot of users there will be problems in the software in terms of scalability. Since simulators update every specific document in the database, hence there may be limitations in the hardware (in terms of performance). Inorder to improve scalability for new users instead of updating everything in the database we could stream the simulation data over a socket which the frontend could read from (anticipated by teacher).

The benefits with using MongoDB and the document database is the characteristics of it since it's very efficient in treating large sets of data (performance). Therefore since the application is real time dependent (because of the simulator) the use of a document database is a good choice. The benefits of using react is that we have a very robust front end since it has alot of useful functionality when creating web applications. Another benefit is the use of redux on our frontend/backend, we have a good encryption method for the password and store a hashed value. Also we use tls certificate, encrypting all the requests between client/webapplication and client/api.

Another benefit is the robustness of our backend. We have the “correct” amount of get/put statements, and functionality doing what it's meant for.

Challenges

Getting familiar and even decent at a lot of new tools is always a big challenge, one could argue the biggest challenge. Two major hurdles were firstly understanding how reactjs, components and a single page app works since it’s very different from what we’ve done before. The second one being implemented react-redux. We had discussions early on and talked about it at the presentation whether it’s worth the setup or not. We’re still not sure but in the last days of this project it kept coming up useful once the system was fully implemented and somewhat understood. Otherwise the setup of the entire project was a bigger challenge than expected. It took way more effort than anticipated to setup the server and reactjs, deciding on all the different tools and understanding why to choose them.

Future work

In terms of the front end, there are a lot of improvements on the current components that could be done, and of course new components or completely new functionalities. For the current components one could recreate base components from the bottom up to make more code reusable. This would not necessarily be more difficult or more time consuming than what was created now, but with the learning curve of using reactjs for the first time one easily realizes structural “mistakes” in hindsight.
Transitioning out of class components and into functional components so one can use react hooks and newer functionality could be a rework worthwhile. This, tying into the current component structure would not have been implemented with our current understanding of reactjs. It was only recently it came to our understanding that since 2019 react hooks were implemented in a capacity where it almost outdated the old class component structure. This created some difficulty understanding reactjs since it seems to almost be in a transition between the two approaches.
Due to lower priority there are still some semi-non user friendly interactions in the frontend, such as showing links to routes for everyone, even the roles that cannot access it, not setting up a proper navbar to have all the relevant routes.

Måns Oskarsson - mnsosk-7@student.ltu.se
Oscar Lundberg - osclun-7@student.ltu.se

A really important yet small improvement would be to figure out how to route a manager to the user profile the manager wants to edit. Unfortunately with some poor time management on that functionality we just missed to implement it. It was somewhat more difficult to get the users ID while logged in as manager to route to the page than anticipated with the current structure. Although, as listed under editing managers on page 1, the functionality is completely there.

React-redux could also be fully fleshed out and very well used if one was to scale out the system. This was a design choice that's not really needed for the current scale of the system but would be immensely useful when scaling it up.
The backend doesn't need any modifications, only if some new data is added into the document database. The simulator could need a socket stream to the front end instead of the current solution. Also the simulator could be modified in terms of viewability. Its quite complex in its current form but it works.

Finishing thoughts

As always, with a little more time it would have been great. At some points in the project, more the later on, it was a lot of fun, and at other points very frustrating. Deploying this system from scratch has been a big learning experience, as well as having loose reigns on what tools to use. Having to take such decisions and researching their pros and cons was harder than one would initially think.

Personal efforts

Week	**Task**	**Måns (time)**	**Oscar (time)**
45	Startup	8 hours	6 hours
46	Backend/frontend startup General work	18 hours	16 hours
47	Simulator and backend, creating separate	17 hours	-
47	Frontend, states for different users/roles. Research and create update functions and get in react.	-	18 hours
48	Simulator and	25 hours	-

	backend, starting using each module together		
48	Frontend and backend, using modules together	-	27 hours
49	Simulator and backend, starting using each module together	13 hours	-
49	Frontend and backend, using modules together	-	11 hours
50-51	Getting everything to work together. Presentation	21 hours	18 hours
52+	Finalizing, completing and doing general testing. Some bug fixes.	33 hours	31 hours

The contributions in general in this project have been fairly similar, Måns was more focused in doing the simulator while Oscar created the frontend, both team members were involved in the backend. No problems or limitations with a member not doing required work. In summary I believe our group deserves a **3 as a grade** since we fulfilled basic requirements and presented basic knowledge in dynamic websystems by developing a fully functional API. More functionality probably could be implemented but it doesn't necessarily fulfill a particular grade criteria, the presented work is good enough to demonstrate basic knowledge in API development.

References

Mongoose : https://mongoosejs.com/docs/
Express : https://expressjs.com/en/api.html#app
Normaldist : https://stackoverflow.com/questions/25582882/javascript-math-random-normal-distribution-gaussian-bell-curve
Postman : https://learning.postman.com/docs/getting-started/introduction/
React : https://reactjs.org/docs/getting-started.html
Learn the MERN Stack : https://www.youtube.com/watch?v=7CqJlxBYj-M