M7011E - Design of dynamic web systems

Simon Pontin, simpon-7@student.ltu.se
Anders Dahl, naddah-5@student.ltu.se

January 2022

1 Introduction

This project is part of the course M7011E - Design of dynamic web systems at Luleå Technical University, referred to as LTU from here on, and is intended as a exercise in creating distributed systems and creating APIs, application program interface. The project objective is to simulate an electric grid control system which would be used by electric companies to monitor and control electric consumption, production, and sales. This will combine a continuous simulation that uses stored data in a database to increment the data over time. A web based user interface will communicate with the database to retrieve and display current data as well as modify the state settings. In short the project consists of three main parts, the database, the simulation, and the front-end. The system will be accessed by two types of users, prosumers, and managers.
Prosumers are customers of the electric company, they can buy electricity from the grid, consume, and/or produce electricity, produce, that may then be sold to the grid.
Managers are the owners of the power plant in the electric grid, they can see who is connected to their grid and how much demand for electricity there currently is in the grid. They can then adjust the production to suit their needs and may also block individual prosumer from buying and selling electricity. Should the electricity demand surpass the production houses will begin to experience brownouts which should be visible to the manager.

2 Method

2.1 Design pattern

We chose to use a model-view-controller pattern, MVC, as it is simple and amongst the easiest to implement. In this pattern we chose to use the database as a model, i.e. it stores the state of the simulation and user data. The simulator then loops over this data and updates with the incremented values. The view, then, is the web based interface which retrieves and shows data from the database to the user. This component also contains the most significant deviation from the standard MVC design pattern, as it can also modify some data in the model, DB. Specifically it is from the web interface that users create houses, batteries, and wind turbines etc, as such it functions both as a view and a controller in some ways. This however is necessary for the simulation to be interactive. In short, the controller will be split into two with the simulator being responsible for the automatic data changes occurring over time. While the web interface can only modify preferences, such as buying and selling ratios, and add or remove "objects" from the simulation. It can not change, for instance, the amount of electricity stored in a battery.

2.2 Database

We chose to use mongoDB as our database and use the graphql API to communicate with it. Graphql is a relatively new technology that was originally designed by Facebook to use internally, and later released to the public as open source in 2015. Since then the API is overseen by the GraphQL foundation. We chose to work with this API as we perceived it as being the new bleeding edge technology in the industry with a focus on speed and as such would be useful to learn.

2.3 Front end

For the front end we chose to use react which is also developed by Meta, previously Facebook, and quite bleeding edge. Additionally it is designed to be modular with a heavy focus on objects being reusable and to behave differently based on context. As both react and graphql were both developed by Facebook we also assumed that they would interact and work well with each other.

2.4 Server

We implemented the server using nodeJS and express as we both had some limited previous experience working with those as well as them being compatible with both graphql and react. JavaScript is also the dominant language on for web development and is therefore useful to be more experienced in.

2.4.1 Simulator

For the simulator we chose to use nodeJS to make integration with the server more seamless.

2.5 Security

The main focus regarding security from the outset in our project was to use JSON-webtokens, JWT, and input parsing, to prevent injection attack. As JWTs are hashed on creation and therefore can not be altered maliciously after being issued, this also makes authorization simple as you simply need to submit the JWT along every user interaction. The second part of our focus, injection attacks, is due to them being both the most common and dangerous attack vectors. As an attacker can run arbitrary code if the input is not being parsed.

3 Result

At this time the project is not yet complete.

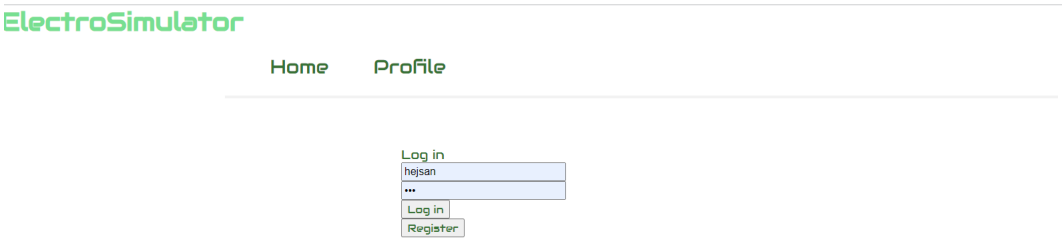

Figure 1: Shows the login page.

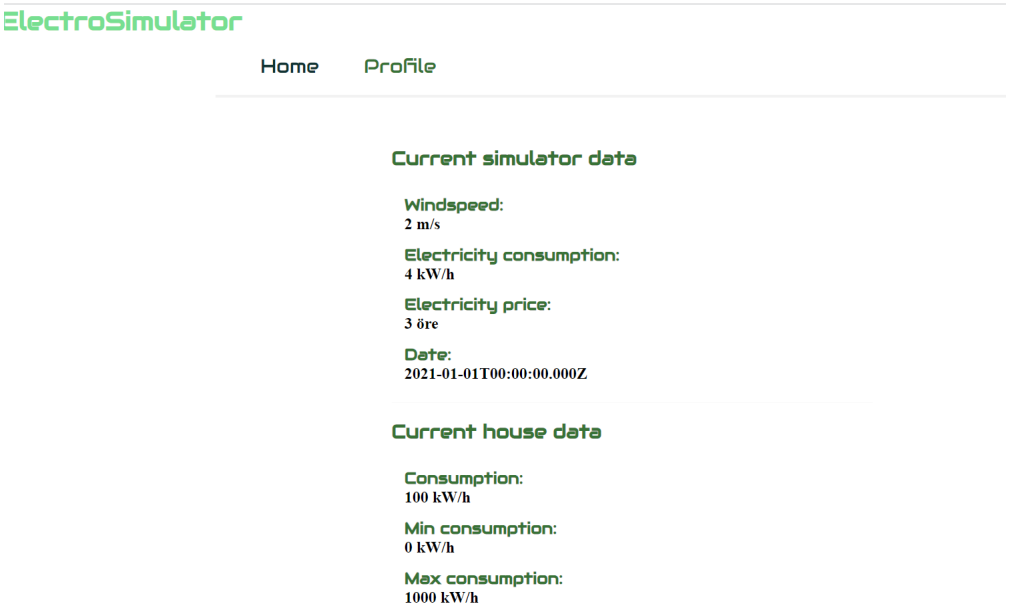

Figure 2: Shows the page where the user can see all its data. All data is not shown in the picture due to not fitting in the picture.

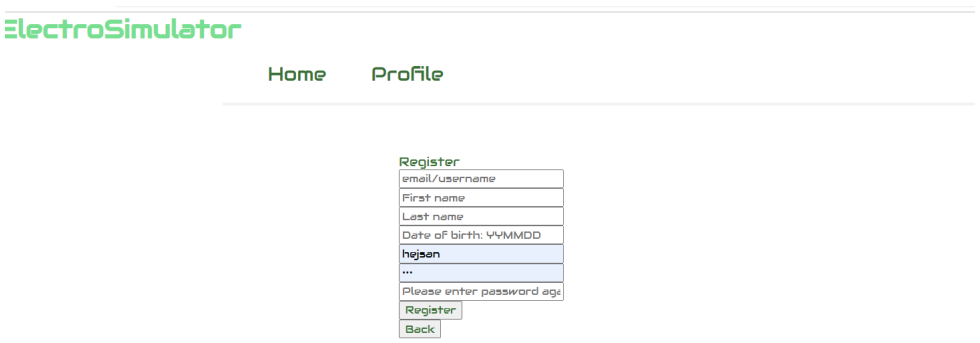

Figure 3: Shows the registration page

4 Discussion

4.1 Security

4.2 Scalability

4.3 Difficulties

5 Github

https://github.com/naddah-5/M7011E/tree/anders-sprint2

6 Time-report

6.1 Simon Pontin

Task	Hours	Description
simulator	34	creating functions for generating windspeed, price, consumption
solve cors for api	0,5	implemented cors so api can be used
simulatorEvent	5	scheme, database and resolvers
house (old prosumer)	32	creating scheme, resolvers and database. Fix relation to other Database objects
useAPI class	15	a class with api calls. Contains mutations and graphQL queries
navbar	18	Setting up a navbar with links. Css fix.
homepage	65	The page displaying data from simulator and house. Css fix. Handle different api data in react.
helper class for resolvers	15	A class that supports resolvers with binding return values and relations

6.2 Anders Dahl

Date	Hours	Description
01-11-21	4	Reading about different API types and their properties.
02-11-21	3	Reading about node.js
02-11-21	5	Installing ubuntu and setting up coding environment.
03-11-21	4	Setting up project structure and debugging software.

04-11-21	4	Working with graphQL and graphiQL.
04-11-21	2	Seminar.
05-11-21	4	Experimenting with mongoDB.
08-11-21	4	Trying to set up the front end.
08-11-21	3	Workshop and lecture.
09-11-21	2	Design meeting and minor code work.
09-11-21	3	React tutorial.
10-11-21	3	React tutorial.
10-11-21	4	Building website with react but having "some" difficulties with versions of different supporting packages not working with each other.
11-11-21	4	Working on the log in page.
15-11-21	5	Working on frontend.
15-11-21	1	Lecture.
15-11-21	3	Working on frontend.
16-11-21	3	Meeting and planning.
16-11-21	4	Reading about react hooks and fixing tech problems.
17-11-21	5	Connecting login to graphql and debugging.
18-11-21	4	Lecture
19-11-21	4	Working on json parsing.
22-11-21	2	Lecture.
22-11-21	3	Json object handling.
24-11-21	8	Refactoring the code, splitting up login page components and reworking the code to be function based instead of class based.
25-11-21	4	Refactoring the individual components to also be function based.

26-11-21	4	Lecture and version control.
01-12-21	5	Refactoring code to use hooks instead of observer.
01-12-21	4	Working on registration page.
02-12-21	7	Working on registration page, ran into problems with CORS.
06-12-21	2	Lecture
06-12-21	4	Fixed CORS issue.
07-12-21	7	Working on registration page query, unclear error.
08-12-21	5	Finished registration page.
09-12-21	3	Working on expanding db
10-12-21	2	Lecture
10-12-21	3	Working on the db
14-12-21	8	Working on db mostly trying to fix a bug that doesn't throw an error
03-01-22	5	Getting back into the code base.
04-01-22	4	Still getting back into the groove
05-01-22	6	Working on database and resolvers.
07-01-22	4	Bug hunting.
10-01-22	4	Working on database and graphql.
11-01-22	7	Working on graphql resolvers.
12-01-22	6	Continued working on the database, had to do some minor refactoring.
13-01-22	4	Continued working on graphql resolvers.
14-01-22	5	Writing the report.