M7011E - Design of Dynamic Webb Systems
Group 13 Report

Elsa Jonsson - elajon-7@student.ltu.se
Carl Österberg - acesea-7@student.ltu.se

October 2020 - January 2021

Contents

1 Introduction

The purpose of the course was to develop a dynamic web system for a company, Green Lean Electronics (GLE). The web system should control the production and consumption of electricity on an electrical grid consisting of a power plant operated by a manager and also multiple prosumers which are house owners that also have their own wind turbine that produces electricity depending on the wind speed. The house owners can also have the role of consumers, which is a household that does not have a wind turbine, but only uses energy directly from the power plant.

In addition to the electrical grid, a simulator which simulates real-life values like wind speed, electrical consumption, price aswell as production depending on wind. These values are accessed and manipulated with an API. These values are used in the system to have a realistic simulation of the GLE system.

The three main parts of the electrical grid have a bit of different functionality. The manager should oversee the power plant. This includes controlling its production, aswell as having an overview of the prosumers and consumers in the grid. The manager should be able to change the production aswell as stop it fully depending on different parameters, such as if the wind is high enough for the prosumers to be self-sufficient. The power plant also has a battery which it can fill up with electricity that can later be used when the power plant production is stopped. The manager should be able to control how much energy goes to the battery and how much goes out into the electrical grid to provide the prosumers and consumers in the system. The manager should also be able to set the price of the electricity produced depending on the current market demand. A suggested price should be modelled in the simulator, but the manager should be able to set whichever price they would like. The manager should also be able to change users information, acting like a sort of admin of the web system aswell. They should be able to change details such as the username, email and name of a prosumer or consumer, aswell as delete them completely. In addition to this, they should see when a user is logged in and be able to block the prosumers from selling electricity to the market.

The prosumers are the households, and should have an overview of their current production, consumption aswell as see the current wind speed and price. They should, like the manager, have a battery where they can store electricity when they have produced more than they have consumed, bu also be able to sell this to the market. When the produced electricity is not enough, the additional energy should be taken from the battery or from buying from the power plant. The prosumer should be able to control the ratios of how much should go into/be used of the battery. If there is not enough electricity in the battery or from the power plant to cover a prosumers consumption, a blackout should occur. This is very bad and should be prohibited by warning the users. Just like the manager, prosumers can update their own information.

Consumers are prosumers without the wind turbines, so they do not produce any electricity. They do not have a battery to story any energy either. Otherwise the functionality is the same as for the prosumers. All three types of users, managers, prosumers and consumers should also be able to upload a picture to their profile.

2 Method

2.1 Front-end

The front-end of the system was developed with JavaScript aswell as Embedded JavaScript templates (EJS). EJS was used since it is a good way to define HTML pages where you can easily use dynamic data. AJAX (Asynchronous JavaScript and XML) was used to allow the site to make calls to the server and update dynamic data without reloading the site. Express was used to route the data because of if simplicity and how widely used it is.

To preface when we go further a lot of decisions were weighted with knowledge we already have from prior webbcourses, specifically php. EJS was choosen with this in mind because its similar in use, just add a tag and start typing out js code. Comparativly to something like react it didn't seem as intuitive with the only downside being that ajax had to be implemented separately.

2.2 Back-end

The bare-bones is built with NodeJS, as it was a requirement for the course. Express was used as a webframework for NodeJS beacuse its the, by a margin, most used framework for Node. Using express gives the option for a lot of useful middleware, which is vital for a project like this.

When choosing the database we had a few key points in mind: performance, scalability and non heavy data-relations. MongoDB fits this category very well and is also heavily used in conjunction with NodeJS.

For login authentication we went with a middleware solution called express-session, which uses a redis store to hold sessions. Redis is just a key value store and gives you the session JSON-object of the associated user sending data to the http server. Also here our php background colored our choices, we went with familiarity beacuse it severly lowered the research/understanding phase of the middleware solution.

The simulator generates all the relevant data for all the different kinds of users. This is what the API layer utilize with whenever a http request is received. The structure of the simulator split into a few different parts, one each for the different kinds of data we want to simulate, wind, consumption, production, price a separate part that hold all of these and finally a API layer that communicates with the simulator. The simulator only generates values and based on these generates a price model. The values generated are based of a Gaussian distribution to try and emulate some real-world elements in the simulator. All of the other calculations are done in another part. This part communicates with the API, performs calculations and puts them into the database.

Lastly for the API layer we had prior knowledge of graphQL and knew its great benefits compared to a RESTful API. With graphQL there is no need for multiple endpoints all the data gets sent in one request.

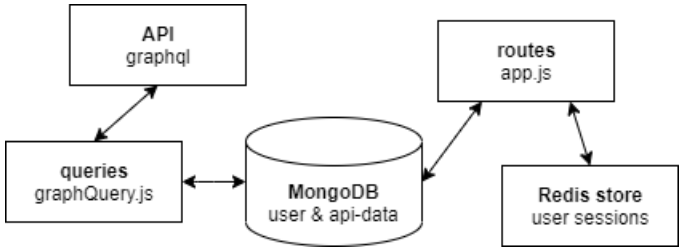

Figure 1: Architecture overview

3 Result

3.1 Front-end

The website consists of a home page, a personal page aswell as a user overview for the managers.

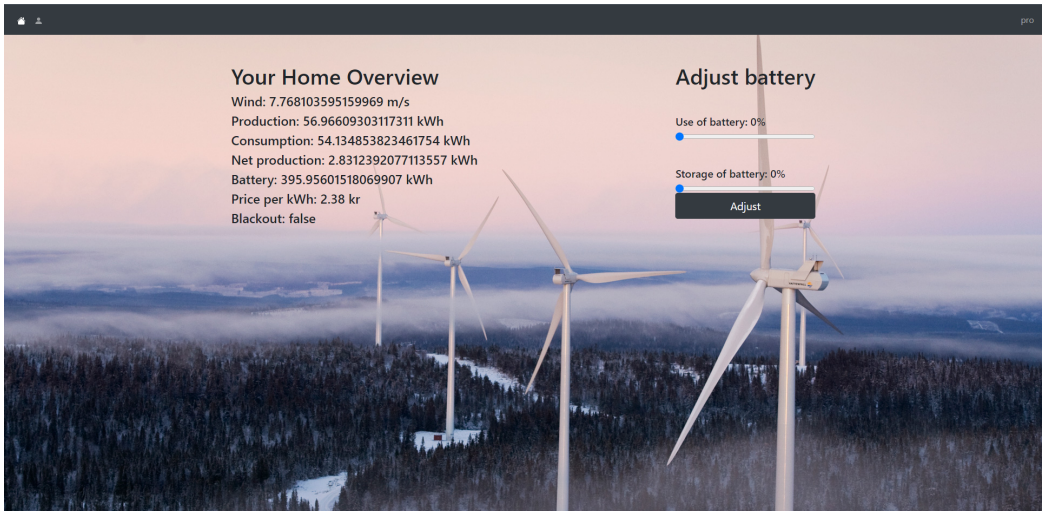

Figure 2: Prosumers home page

The home page for the prosumers shows the current state of the house and allows the user to change how much should go into the battery when the net production becomes positive aswell as how much to take out of the battery when its negative. You also see the price set by the managers. When a user is blocked from selling to the market or a blackout might happen because the power plants battery is low warning messages also appear on the home page.

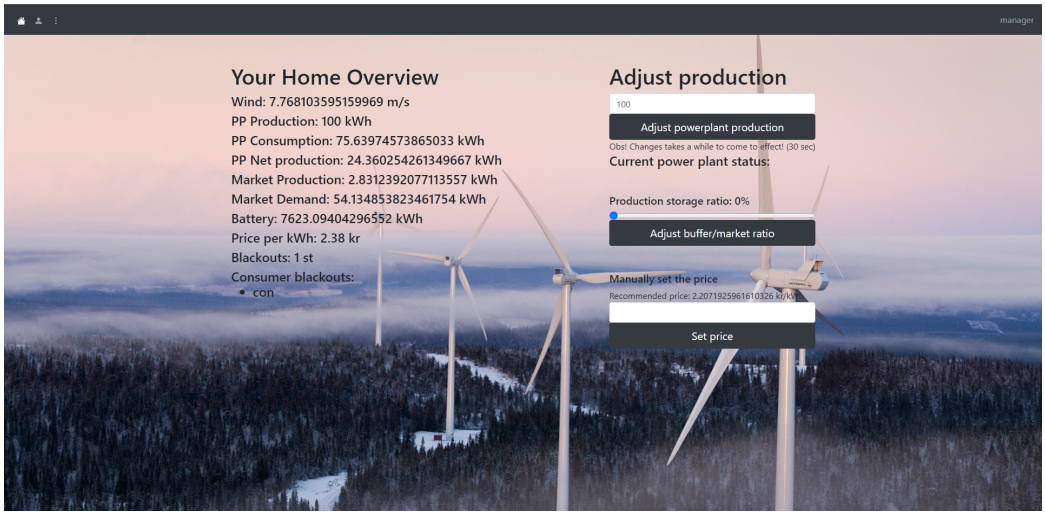

Figure 3: Managers home page

The home page for the manager shows the current state of the power plant aswell as the market. You can see and change the power plants production aswell as price, and also decide how much produced should be sent to the battery and how much should go out to the grid. You can see the market demand aswell as the current blackouts.

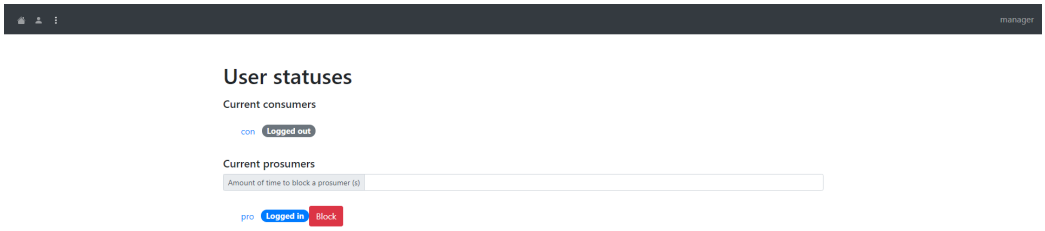

Figure 4: Managers users list

The manager also have a list where they can see all the prosumers and consumers in the system. They can see which users are online, and also block the prosumers from selling to the power plant. They can also click in on a user which redirects them to the following page:

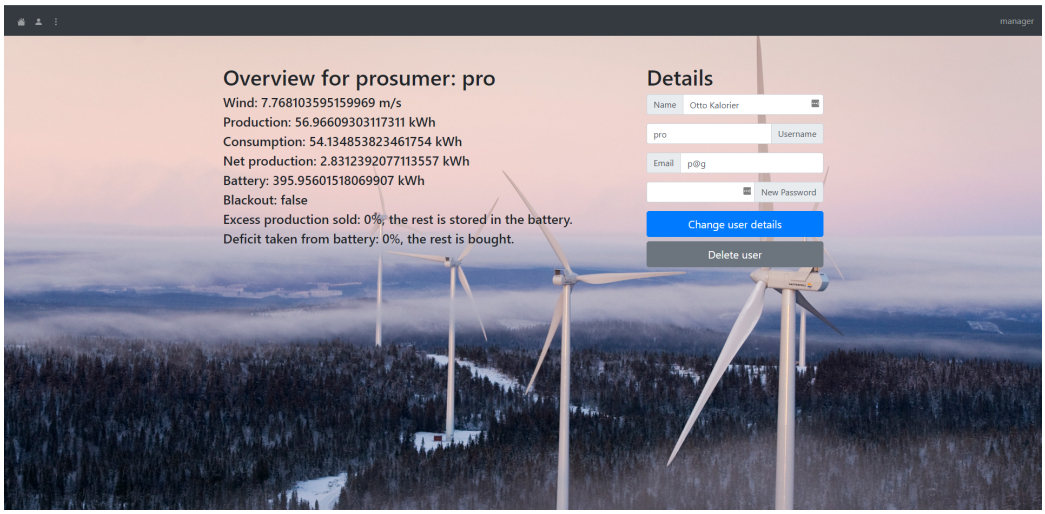

Figure 5: Managers users overview

Here, the manager sees an overview of the prosumer or consumer, and can see their current simulation values, aswell as update their information. Here, the users current password is not needed to change information, since the manager is acting as the admin and does not need as many safety precautions. The manager can also delete users.

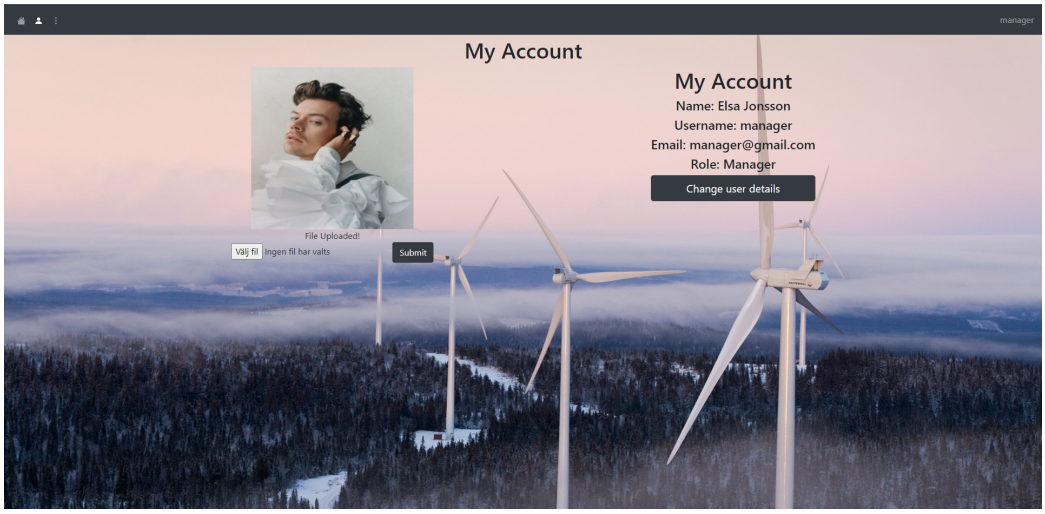

Figure 6: Personal page

The personal page looks the same for all users, and shows the uploaded picture aswell as the information. To change the picture, you simply choose a new file and then submit it. To change the other information you click the button and it will redirect you to the following page

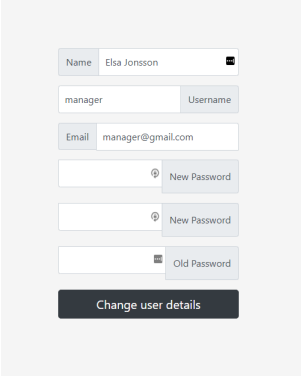

Figure 7: Change information

where you can update your information. To update the information, you have to write in your current password, otherwise it wont work. To log in or create user, you get directed to files almost identical as this, but instead of changing details you either sign up or sign in.

Other than that, there are multiple error sites that you get if something goes wrong, but for the manager they were not implemented as nicely as for the prosumers and users because the group prioritised other tasks instead.

3.2 Back-end & Simulator

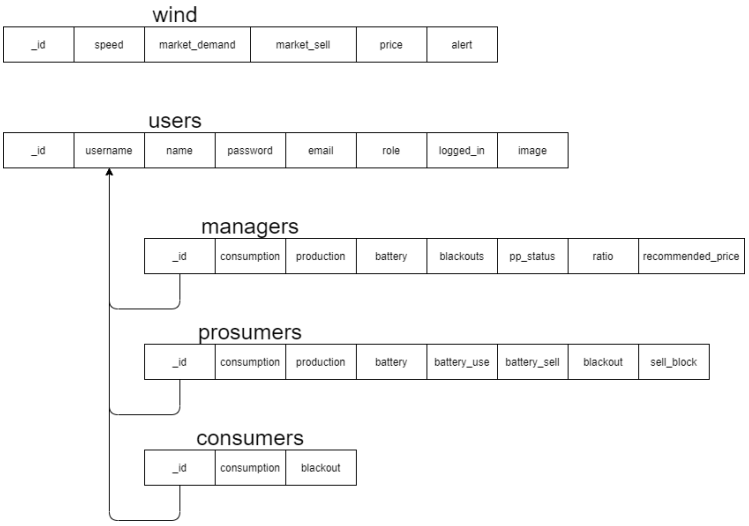

Figure 8: Database collections

The MongoDB database contains five different collections. One is the 'wind' collection which holds all the global variables that all users should be able to read, such as price and wind speed. One is 'users' which holds all the users registered on the site and their personal information, such as their email, password and image path. Then, there are 'managers', 'prosumers' and 'consumers' which holds all the respective users depending on their role. There, they have information about the simulation, such as their current consumption, production etc.

The simulator consists of different classes - a wind, price, consumption and production class. These are all connected by a class sim which hols objects of each class. There is an API implemented which makes it possible to communicate with the simulator. The API is queried by the GraphQL file, which also connects to the database to update values of all the users.

4 Discussion

4.1 Scalability Analysis

Right now the the part that would bottleneck our project would be the database. Luckily for wide scale deployment MongoDB supports sharding which tackles this issue.

On the API there is only one request every 10 seconds, no matter how many users, no scalability issues here.

The final thing to consider is the redis store which holds all of the sessions. These sessions are only small JSON objects which are not demanding.

4.2 Security Analysis

The security of the system is not horrible but could have been improved aswell. Some measures were taken to improve security. The passwords were hashed using bcrypt before they were stored in the database. They are hashed using 10 salt rounds, which basically means that they are impossible to guess or decode.

Sessions were also used once a user is logged in which used cookies to make sure that the users only saw the content they should. A prosumer cannot go into the manager home page since they are not a manager. Sessions were used for authentication of the users.

One problem regarding security in the sense of not overwhelming the website that was fixed was the implementation of images. They were uploaded to a folder on the server, which at first just uploaded an image every time a user would upload a picture. This was fixed, so each user can only upload one picture, and if they upload a new one, the old one will get deleted. This was the server cant get DDoS by a user uploading pictures.

On the other hand, there is no check that the email that the users give have to be real. This way, a DDoS attack can occur if someone creates a lot of users and overwhelm the database.

Ejs is rendered on the server and not on the client side. This improves security since no user could use their client-side console to see what's going on with the server.

An improvement regarding security would be to use HTTPS instead of HTTP. This should be done before the actual deployment of the website. This is important as the information that flows from server to browser is not encrypted, which means it can be easily stolen. This is not good since we are dealing with personal data which should be protected.

4.3 Advanced Features

The group did not fully implement any of the suggested advanced features, and did not make up any other ones themselves either. The one thing that was mentioned in the prosumer advanced functionality that was implemented was the user friendly sliders to control the ratio from/to the market and battery. Other than that, the website has the basic functionality that was needed. Instead of implementing another feature, the remaining focus was to try to fix as many bugs as possible, implement some security and a user-friendly interface that appeals to the average user.

4.4 Challenges

Starting out with the project we had issues getting a good overview, this lead to some us feeling a bit overwhelmed and a lot of stumbling around. So we started out small and worked our way out. This lead to some questionable choices which had to be reviewed and altered. An example of this is a collection in MongoDB called wind. It was named this because we want every user to access to it, but this collection

grew into all the data that every user should be able to access. That name really doesn't make sense anymore, but its spread all around the code so fixing it wasn't really worth because of time constraints.

Secondly we had some issues with JavaScript itself and getting used to its quirks and features. Going from strongly typed languages like Rust and Java to a loosely typed language like JavaScript is very annoying. Type errors like when suddenly a number turns into a string can have be disastrous effects, because you grow accustom to the compiler telling you when this happens. JavaScript just accepts everything so long as it doesn't crash. In the future we will definitely use typescript.

Another feature of JavaScript, it being single threaded, also led to some issues. Specifically how promises work. Looking around, the old method of combating this was using callbacks, but this leads to some really ugly indentation nestling. Also realizing what this all meant was a bunch of trial and error.

4.5 Future Work

For an actual release a first would be to implement https for encrypted traffic for authorization.

Because of procrastination we didn't make use of good coding practices. Starting out we did follow this and tried to structure things into separate files and have a object oriented approach (seen in simulator). But afterwards all of this got abandoned when we started working on the express routes in app.js. This had a lot to do with us not understanding js promises and using callback chains instead. Given time this would be one of the first things to fix.

One other thing that needs to be improved further is the user interface, which at the moment is fine but has some flaws. It is not always clear how to control certain things such as that in order to update your personal information, you have to write your old password. There is no indication at the moment. The error pages are also not fully done, since most of them are barely a page, but rather just a sentence of what went wrong and nothing more. If an error occurs when changing your own information, you go to a standard error page that's tells you that something went wrong, but not what. You don't know if you wrote the wrong password, if the username was already taken or what.

Another thing to do is to implement sharding in MongoDB to increase scalability. This is not relevant at the moment, but is needed before the web system is released on a larger scale.

There is also a known bug that has to do with the session. Right now whenever a user logs in their status is set to logged-in in the database. This only changes when the user presses the log out button. So if the user would just destroy the session by closing the browser the user is still seen as logged in.

5 References

Tutorial of how to implement image uploads with Multer:
https://www.youtube.com/watch?v=9Qzmri1WaaE&t=1357s&ab_channel=TraversyMedia

Bootstrap tutorial:
https://www.w3schools.com/bootstrap4/

GraphQL:
https://graphql.org/

Node.js tutorial:
https://www.guru99.com/node-js-tutorial.html

JavaScript tutorial:
https://javascript.info/intro

Express tutorial:
https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

MongoDB with node.js tutorial:
https://www.w3schools.com/nodejs/nodejs_mongodb.asp

jQuery Ajax get/post tutorial:
https://www.w3schools.com/jquery/jquery_ref_ajax.asp

Gaussian bell curve:
//https://stackoverflow.com/questions/25582882/javascript-math-random-normal-distribution-gaussian-bel

Express sessions with Redis:
https://codeforgeek.com/manage-session-using-node-js-express-4/

Hashing and verification of passwords:
https://www.npmjs.com/package/bcrypt

6 Appendix

6.1 Time Report Analysis

The Time Report kept by the group was not very good, since we did not provide much further information than which part of the assignment we were implementing. You can see that the group had a slow start to the project, and spent a lot of time during the Christmas break to implement the manager aswell as fix all the bugs. You can see that the group did not spend as much time as expected to implement the web system, but rather a little over half of the estimated time. This can depend on having a good workflow, clear instructions aswell as not implementing many advanced features.

6.2 Contributions

Both members of the group have worked on all parts of the code, with some small exceptions. For example, while Carl implemented AJAX, Elsa implemented image uploads with Multer, and while Carl implemented the managers list of users, Elsa implemented the update of information of users. In the beginning, the group worked on everything together, by coding together. This was good since both members understood the entire web system, but since the group did not work enough in the beginning and procrastinated implementation of the manager by underestimating the time required to implement aswell as the time needed to fix bugs, at the end both members took the remaining tasks and solved them separately to be more efficient. All these parts were explained to each other during meeting though to still keep the understanding between both members. The group have had good contact and discussed the implementation before actually coding which made the implementation easier.

6.3 Grades

Elsa Jonsson: 3
Carl Österberg: 3
We think that since we have developed a program that has all the basic functionality requested aswell as developed the web system with security in mind, we should get a passing grade. No advanced functionality was implemented, but the group made sure to make the web site user friendly aswell as somewhat reactive. There are some implementation decisions that the group in hindsight thinks were poor, but have learned from them and are now better programmers because of it. Therefore we would give ourselves the grade of 3, possibly 4 if we were being generous.

6.4 GitHub Link

`https://github.com/CarlOsterberg/M7011E`

Deployment instructions are in the ReadMe file in the GitHub.

6.5 URL to website

`http://130.240.200.36:3000/`