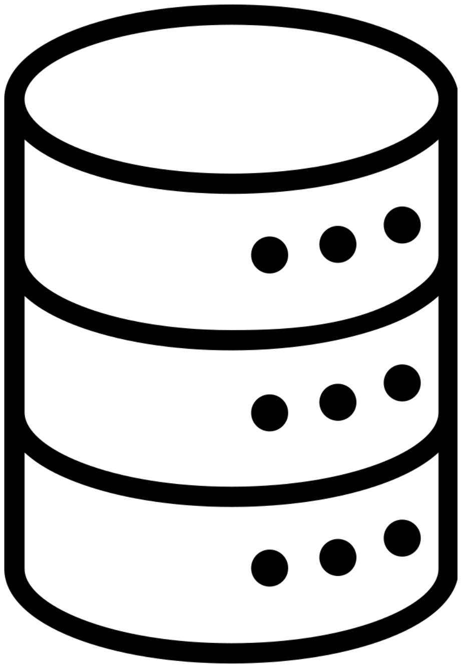
Professor	Peter Parnes	Group	Awesomers
Postdoctoral	Niklas Karvonen	Viktor From, 910227	vikfro-6
Date	January 18, 2020		

Design of Dynamic Web Systems

Luleå tekniska universitet



Contents

1 Introduction **1**

2 System architecture **2**

2.1 Simulation 2

2.2 Server and client 3

3 Scalability **5**

4 Security **6**

5 Advanced features **7**

6 Challenges **8**

7 Future work **9**

8 References **10**

9 Appendix **11**

Professor	Peter Parnes	Group	Awesomers
Postdoctoral	Niklas Karvonen	Viktor From, 910227	vikfro-6
Date	January 18, 2020		

1. Introduction

The report contains specifications of front-end as well as back-end of a web server written with the purpose to simulate and control a small scale electricity market. The project was written in JavaScript during 2019-2020 under guidance of Professor Peter Parnes and Postdoctoral Niklas Karvonen at Luleå tekniska universitet.

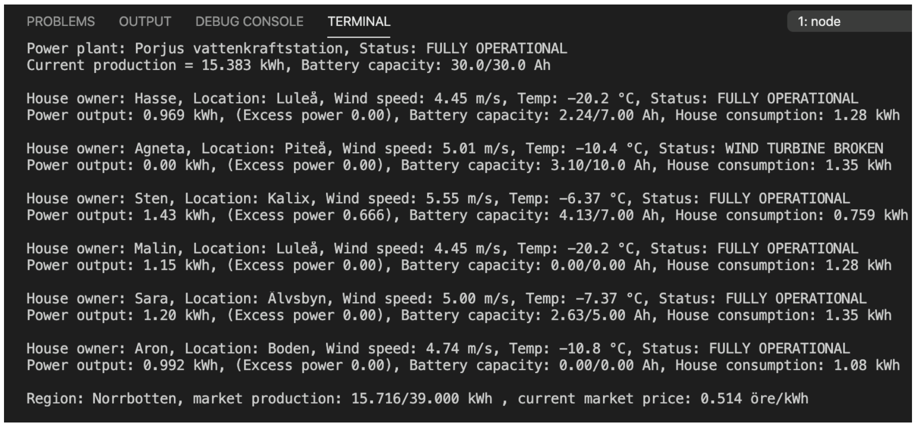
The main purpose was to give an understanding of how to build dynamic web based system using publicly available modules, interface between client and web server, security and ethical problems related to sensitive data.

2. System architecture

2.1 Simulation

The core of the system consists of a simulation with the purpose to simulate and control a small scale electricity market. The simulation is more or less a self-contained engine intended to run on it's own, wrapped in a single setInterval() function. During initialization, four main objects needs to be instantiated in order to run, a market price, a power plant, houses and regions. The simulation was initially written with high modularity in mind, allowing instantiating of multiple counties each with their separate market price and power plant. At the end of the project this feature was disabled in order to increase simplicity and robustness to the system. However, each county still has the possibility to contain multiple instances of houses and regions. Finally, all of the objects variables utilizes normal distribution in order to produce more natural values.

Once a user registers on the web client, a new house is instantiated and a new region, unless said region is already registered in the database. The reason for this is to allow regions to share wind speed and temperature, which affects the house electricity consumption, see Fig. 1. The house itself is by default instantiated with a wind turbine and a local battery, for easier testing all houses are outfitted with both modules. However, the house object could be set to run without either of them.



```
Power plant: Porjus vattenkraftstation, Status: FULLY OPERATIONAL
Current production = 15.383 kWh, Battery capacity: 30.0/30.0 Ah

House owner: Hasse, Location: Luleå, Wind speed: 4.45 m/s, Temp: −20.2 °C, Status: FULLY OPERATIONAL
Power output: 0.969 kWh, (Excess power 0.00), Battery capacity: 2.24/7.00 Ah, House consumption: 1.28 kWh

House owner: Agneta, Location: Piteå, Wind speed: 5.01 m/s, Temp: −10.4 °C, Status: WIND TURBINE BROKEN
Power output: 0.00 kWh, (Excess power 0.00), Battery capacity: 3.10/10.0 Ah, House consumption: 1.35 kWh

House owner: Sten, Location: Kalix, Wind speed: 5.55 m/s, Temp: −6.37 °C, Status: FULLY OPERATIONAL
Power output: 1.43 kWh, (Excess power 0.666), Battery capacity: 4.13/7.00 Ah, House consumption: 0.759 kWh

House owner: Malin, Location: Luleå, Wind speed: 4.45 m/s, Temp: −20.2 °C, Status: FULLY OPERATIONAL
Power output: 1.15 kWh, (Excess power 0.00), Battery capacity: 0.00/0.00 Ah, House consumption: 1.28 kWh

House owner: Sara, Location: Älvsbyn, Wind speed: 5.00 m/s, Temp: −7.37 °C, Status: FULLY OPERATIONAL
Power output: 1.20 kWh, (Excess power 0.00), Battery capacity: 2.63/5.00 Ah, House consumption: 1.35 kWh

House owner: Aron, Location: Boden, Wind speed: 4.74 m/s, Temp: −10.8 °C, Status: FULLY OPERATIONAL
Power output: 0.992 kWh, (Excess power 0.00), Battery capacity: 0.00/0.00 Ah, House consumption: 1.08 kWh

Region: Norrbotten, market production: 15.716/39.000 kWh , current market price: 0.514 öre/kWh
```

Figure 1: Console.log() output of the simulation.

The power plant and house objects share some similarities in their structures, where the difference in production of the power plant is, except for the obvious such as higher production etc., more advanced. The power plant has a "start up" sequence and a "powering down" sequence, which occurs during server boot up and blackouts. During these events the power plants production will accelerate/decelerate.

In terms of electricity consumption, the users house will prioritize consuming the electricity generated from their personal wind turbine. If net production is negative, due to underproduction or the wind turbine breaking. The house will turn to the power plant and consume the power plants production instead. Since the power plant has the risk of going into "blackout mode" and subsequently powering down. All houses connected to the power plant will then start to consume the battery of the power plant. Further, if the power plant battery is depleted, every house will then start to consume their local battery. If the local battery is or eventually is depleted as well, the house will also go into "blackout mode". If the net production is positive the house will by default store half of the production in their local battery and send the other half to the power plant battery.

In order to get accurate price and production readings, the market price need to run last during every cycle of the simulation. Since production and consumption from the power plant and all houses are combined using simple arithmetic addition. The market price is determined by,

$$current\ \ price = \frac{1}{\left(10 \cdot \frac{p_c}{p_{tot}} \cdot p_b\right)}$$

where p_c is the current production, p_{tot} is the total production and p_b is the base price of 0.4825 SEK/kWh. Due to the inverse functionality the price will automatically increase when the production goes down and vice-versa. However, it will never go below the base price.

2.2 Server and client

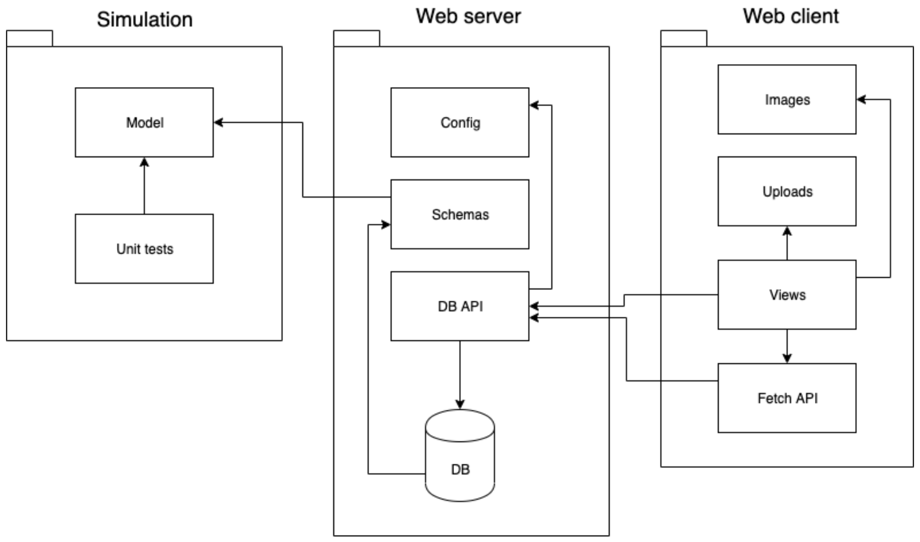


Figure 2: Architecture of application.

In terms of system architecture, see Fig. 2, the simulation was wrapped in Express Framework for node, see (Holowaychuk, 2019), allowing for robust routing and a view system, http redirection, caching and more. Thus, the simulation and back-end was encapsulated from the web client and limiting the communication to the web server using built in JavaScript Fetch API and front-end views. Data from the simulation and user input is stored in MongoDB using schemas and REST API-calls, see (Inc., 2019).

Authentication and encryption of user credentials was handled using Passport middleware, see (Hanson, 2019), and Bcrypt, see (C. et al., 2019). Further security was contained in the config-file in order to lock down the REST API. Only signed in users are able to access the functionality and in case of extended functionality only a manager/admin has the proper rights.

Once an image is uploaded, it's stored in /public/uploads/ on the web client and a REST API-call is made to the database changing the path of the users picture. The other option would have been to store the image directly in the database as a blob. However, the first method was selected due to performance and easier implementation. Since the images are stored in a folder, fewer calls to the database is executed. With a large-scale user base this would have reduced the load on the database, but is less optimal when e.g. storing sensitive images such as profile pictures.

All images used on the front-end are provided by Pixabay, see (et al., 2020a), labeled for "non-commercial reuse and modification". Front-end views and HTML templates was provided using Bootstrap, see (team, 2019). Further, graphs were provided using Chartjs library, see (et al., 2020b), and unit testing was implemented using Mocha, see (et al., 2019b) and Chai, see (et al., 2019a).

3. Scalability

Since the server is stateful a user session can be revoked at anytime, increasing flexibility in user management. Such as keeping track of user statistics, who is online etc. It could also be used to implement soft deletes of user accounts, as the access is restricted but the account is still stored in the database. The disadvantage in terms of scalability would be increased complexity and server hardware. As the sessions require relatively large memory usage on the web server. In the case of database horizontal scaling such as sharding, an algorithm is required in order to keep track of the users session across multiple servers.

The REST API separates the user interface from the data storage and the server. Subsequently, this leads to traffic being directed to multiple end-points as opposed to other methods such as GraphQL API. However, it could be argued that the simplicity of a GraphQL API end-point, allows or increased scalability once the project reaches a tipping point in terms of size. REST APIs also has the advantage of further development on multiple components independently.

Since the project was not written in a docker container, deployment does not become as smooth as it could have been. Each instance of the project needs to be cloned from the Github repository and required dependencies needs to be installed manually in order to run. This obviously would not be feasible in large scale production and multiple servers.

The simulation data does not require complex relational data to be stored, which is the reason why MongoDB was selected for this project. NoSQL-databases are superior at handling large sets of historical data and is very easy to use in terms of scaling.

4. Security

Passport and Bcrypt ensures a high standard of security regarding handling of user credentials by storing passwords in the database using hashing and salt. With the wide range of encryption tactics there are certainly more secure methods of encryption. Bcrypt was chosen due to the PBKDF protocol (Password Based Key Derivation Function) which performs key strengthening. Essentially, this means a brute force attacker has to spend more time with each successive attempt at cracking the password by slowing down the calculations.

As mentioned in section 3, the application is stateful and utilizes sessions to keep track of the user. Where the drawback of this, since https is not implemented, is user credentials such as passwords can be displayed in plain text on the website once the user has logged in. This would not occur with https since the packages would have instead been encrypted.

With MongoDB being a NoSQL database, it inherently prevents the database from attacks such as SQL injections. The reason being that MongoDB simply does not parse input as instructions from the user and hence SQL injections cannot be performed.

5. Advanced features

The following advanced functionalities was implemented in the project,

- House consumption is dependant on temperature in the region. Houses placed in the same region share the same temperature.
- Breakdowns occasionally occurs for the wind turbines.
- Power production from the wind turbine is dependant on the wind speed inside the region. Houses places in the same region share the same wind speed.
- The simulator stores historical data in the database which is then displayed on the users dashboard.
- Alarms regarding wind turbines and the power plants current status is displayed on the dashboard.
- Visual representation of data, Graphjs allows multiple datasets to be displayed on the dashboard.
- User-friendly dashboard, displaying consumption, production, wind speed, temperature and market price.
- Responsive dashboard view, bootstrap allows the interface to run on different devices and screen sizes.
- The system is able to handle multiple logins from the same user on different devices, mobile and desktop.
- The data stream from the simulator can be viewed in real time.

6. Challenges

By far, the most extensive challenge of this project was the fact that I had to involuntarily complete the project by myself. The group fell apart due to difference in workload and after a discussion with Niklas Karvonen the group was split. Attempting to complete all of the basic requirements and keep effective time management proved to be the toughest part. The following basic requirements where therefore not completed in time,

- Through a profile page update credentials and delete the users account (not the manager).
- Should a black-out occur in the system (failing to provide one or more houses with electricity) the Manager should see which Prosumers that are affected.
- Have a list of a Prosumers from which the Manager can:
  - Be able to see who is online (logged in).
  - Be able to view this Prosumer's system.
  - Block the Prosumer from selling to the market for (10-100 seconds).

However, most of the functionality is implemented on the web server in form of REST API calls and simply needs to be brought forward to the web client. During my discussion with Niklas it was mention that I could eventually get extended time to complete the assignment if needed.

Authentication and encryption of user credentials proved to be surprisingly difficult to implement even though third party libraries was used. This could be due to the fact that I've never implemented some form of authentication method before.

7. Future work

In addition to the remaining functionalities mentioned in section 6, high priority would be placed on implementing https. Further, dashboard controls would be replaced with sliders instead of regular input fields, since such input methods are more intuitive.

Due to no prior experience using MongoDB, the detail of built in functionality such as schemas was initially missed before it was too late. The decision was therefore made to simply transfer the object attributes from the objects in the simulator to schema objects before posting the data to the database. Accordingly, this introducing unnecessary redundancy into the system, although not a critical feature to re-factorize but ultimately a something to revisit.

Finally, if time would have allowed it I would have definitely spent more time trying to find potentially unknown security holes and implemened the needed security to prevent them.

8. References

et al. (2019a). *Bdd / tdd assertion library for node.* Retrieved from `https://www.npmjs.com/package/chai` (2019-11-25)

et al. (2019b). *Mocha, simple, flexible, fun javascript test framework for node.js the browser.* Retrieved from `https://www.npmjs.com/package/mocha` (2019-11-25)

et al. (2020a). *Pixabay.* Retrieved from `https://pixabay.com/` (2020-01-10)

et al. (2020b). *Simple yet flexible javascript charting for designers developers.* Retrieved from `https://www.chartjs.org/` (2020-01-03)

et al., C. (2019). *Bcrypt.* Retrieved from `https://www.npmjs.com/package/bcrypt` (2019-12-26)

Hanson, J. (2019). *Simple, unobtrusive authentication for node.js.* Retrieved from `http://www.passportjs.org/docs/` (2019-12-26)

Holowaychuk, T. (2019). *Express framework.* Retrieved from `https://www.npmjs.com/package/express` (2019-12-20)

Inc., M. (2019). *The database for modern applications.* Retrieved from `https://www.mongodb.com/` (2019-12-16)

team, B. (2019). *Bootstrap is an open source toolkit for developing with html, css, and js.* Retrieved from `https://getbootstrap.com/` (2019-11-20)

9. Appendix

1. Appendix 1, A relatively conserving estimate of the time spent on the project resulted in 260 hours.
2. Appendix 2, Anders Dahl, naddah-5, spent time setting up initial database schemas and routing. However, some of the work had to be scrapped or reworked in-order better fit the final server model. Remaining work was completed and written by Viktor From, vikfro-6.
3. Appendix 3, Considering situation and the amount of time spent working on this project, I would give myself a grade of at least 4 or higher. Of course some of the basic functionality is still missing, if I would have had a partner who'd actually written something. This functionality would, with without a doubt, have been implemented and probably more time could have been spent on further developing more advanced functionalities. If this would have been the case I would have definitely considered myself getting a grade of 5.
4. Appendix 4, In hindsight the project should have been forked but alas it wasn't, therefore there's two different repositories.
   Initial repository: https://github.com/naddah-5/M7011E
   Finalized project: https://github.com/viktorfrom/m7011e