# Final Report

## M7011E - Design of Dynamic Websystems

Green Lean Electric Machine
Tom Hammarkvist - tomham-3@student.ltu.se
Anton Grahn - antgra-6@student.ltu.se

June 8, 2020

# 1 Introduction

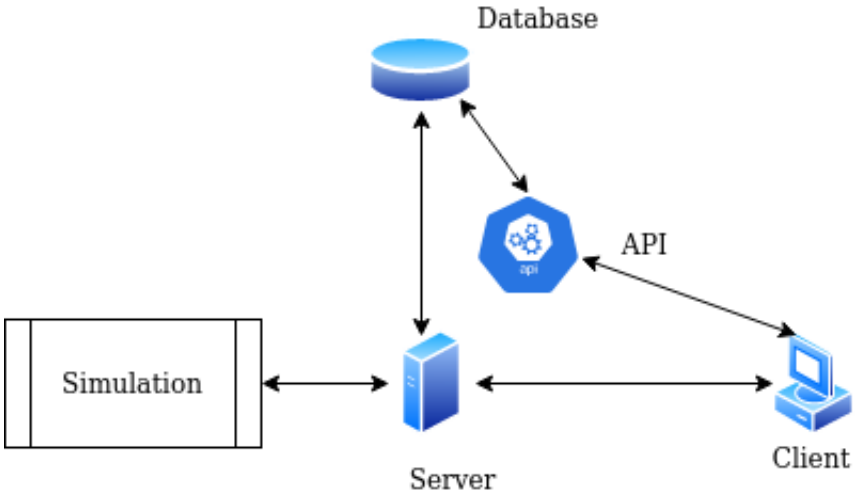


Figure 1: How the data flows in our aplication

Our application starts with a simulation that simulates data. The simulation data is sent to the database but the simulation is located on the server. Then we have a client that either communicates directly to the web server or through an API. The client communicate directly with the web sever when going to different pages on the website. The communication with the API is happening when the client is on the main page where they can see all their data. Because of this the data on the webpage is continually updated and which is done in the background with JavaScript that calls the API to get the updated data.

# 2 Design Choices

We chose to have our project in docker containers. Docker lets you package your application and its dependencies in whats called containers, which then you can run your application in. This means that there isn't any problems with different machines having different versions of dependencies, since the docker-container is the same on each machine its ran on. This meant that the the code written on one members computer ran seamlessly on the other members computer it also helped us to deploy our application on our virtual private server.

For database we chose to use MySQL. MySQL is a relational database which might not have been the optimal database for this application because there are not many relationships. We chose it because there are some relations and because we had prior experience with MySQL. We didn't want to spend too much

time learning and setting up a database but to start coding the simulation and the actual application.

We chose to use restful API because we had no experience with either APIs or grapgQl and it seemed the easiest and the most approachable. So we could learn and get the hang of in a reasonable time-frame and not get stuck on it for too long.

We also decided to send passwords as cleartext using http instead https. We started using http when first starting the development of the site, but thought that changing it to https in the future would be simple. But when doing research on the topic we realized that changing it would be more of a hassle, and since we had time constraints we figured that we'd not manage to swap everything out in time and add the proper support. If we would redo the project from scratch we would have started with designing the site around https from the very beginning instead of starting with http.

# 3 Scalability

The current iteration of the system is not verý well optimized for a high amount of customers. The reason for this is that when a new user is created the info that they put into the registration form is added to the database with a size that is over a gigabyte. This is something that unexpectedly affects the scalability of the system in a bad way.
The believed reason for this bottleneck has to do with the way we store images in the database, since its from the point where we implemented that the problem arose. The solution must therefore be either to change the way we store images in the database or to think of a completely different way to store the images without it.
Another potential bottleneck in the architecture is that the more users that register for the site, the more work the simulator has to do. This will put huge stress on it and most likely make it spit out values at a much slower rate than we originally planned for.

# 4 Security

From a security standpoint the system adheres to a majority of the goals that we set for it. Passwords are not stored in the database as plain text, all database information such as its password is stored in a *gitignore* file so it cannot be accessed by anyone but us. Our system is also secure against SQL injections, which we solved with the *db.escape()* function. This is still not enough as the security side certainly needs more work since we are nowhere close to fulfilling every requirement in the **OWASP top ten project**. There is definitely a way for an experienced hacker to get in, we have just limited the options he/she has.

# 5 Advanced Features

The only advanced functionality we have implemented is that the wind turbine can brake.

# 6 Future Work

In its current state, the system has many things that could be improved in order to make it more attractive to customers. One area that is in due need of improvement is the actual manager side of the system, since not all of the basic functionalities have been completed.
Next thing to work on is definitely to upgrade the security, since eliminating the possibility for an unauthorized user to get access to use the system in a malicious way would be appropriate. One example that would be easy to add is that we remove *http*, and add *https* instead.
The look of both the prosumer and the manager system is also something that could use some more work. The reason for this is that the sites does not really have a good base layout in our opinion. A way to fix this would be to just use a template that fits the site the best, and then modifying it to put our own spin on it.

# 7 Challenges

One of the biggest challenges we faced in this course was something we brought on ourselves. We decided that docker was going to be a part of the architecture, which in hindsight might have been a mistake considering none of us had ever worked with docker before. Issues arose at every turn it felt like, and as soon as we fixed the problem, a new one came forward. This slowed down our progress by a lot and has haunted us every step of the way.

Another challenge we had was understanding how everything was supposed to connect together. We had no prior experience with simulating data to a database or how to write an API. When we had written our simulation we didn't know whether the simulation was supposed to always run on the server or if it was supposed to be done on the client side when the user logged in. We also didn't really know how to write or use an API so we are calling the database both from the API and the routes of the application. All database calls should be done from the API.

We also ran in to a problem where we couldn't connect our server to our database and spent multiple days trying to solve that problem. The solution was extremely simple we only had to enter this command (ALTER USER 'root' IDENTIFIED WITH mysql_native_password BY 'password';) to the mysql terminal where 'password' is our root password.

# 8 Appendix

Anton ended up with 182 hours worked on the project. The majority of that time was spent on functionality for the prosumer and setting up everything around our application like docker and connecting our database to our server.

Tom ended up with 175 hours worked on the project. The time was spent mostly on getting everything setup, since we had never been working with docker or anything like it before, which made it take more time than we originally anticipated since everything in the project is dependent on docker working in a correct manner. Another area where i spent a lot of time on was the RESTful API, since id never worked with and used an API before, and i was not really sure what its purpose was. This took a lot of time to get understand and implement correctly.

When dividing the work between the two of us we just discussed briefly what each one of us wanted to do, and then let that stick. If anyone had any problems we'd ask each other for help if we felt that it was needed.

## 8.1 Contributions

The first few weeks were all about setting the lab environment up and getting a basic simulation working. All these things we did together since everything was new to the both of us. Eventually the group split up, while Anton continued to build out the simulation, Tom setup a MySQL server and started to research how REST works. After we managed to get a connection between the database and the simulation we had to start building the API, which is something that was continuously expanded upon during the entire course. Then when it comes to the webpages we both worked side by side to make sure that we were both satisfied with the look and feel.

## 8.2 About Grade

We think that we both deserve a 3 in this course. Because even though we didn't finish all the basic functionality we still used all the essential elements of this course we designed a webpage, we used an API and we used a database. The things that are missing are pretty much rehashes of parts that already have been done in the prosumer part of the website. Also a big part of this course was completely new to us so we spent a lot of time not making progress on the task but just trying to set our application up and doing research to grasp what needed to be done.

### 8.3 Links to Release

Github release: https://github.com/Hammarkvast/M7011E/releases/tag/v1.0

deployment: 54.85.244.46:80

installation guide: https://github.com/Hammarkvast/M7011E/blob/master/README.MD