



Green Lean Electrics

Made by:

Rickard Frykgård, ricfry-6@student.ltu.se

Hampus Lindberg, hamlin-6@student.ltu.se

Table of contents

Introduction	2
Design choices	3
Scalability	3
Security	4
Advanced features	4
Challenges	4
Future work	5
References	5
Appendix	7

Introduction

Our applications runs on an EC2 instance which is a virtual machine hosted on Amazon's cloud computing platform and we chose to use Windows as our operating system on the machine. The applications that the instance run are two node servers. Server number one provides an API which is used for simulating data, handles all the request sent from the user and stores everything important on a database. The database is provided by MongoDB Atlas which is also hosted on Amazon's cloud. Server number two runs the web applications for both the prosumer and manager. The architecture can be seen in figure 1.

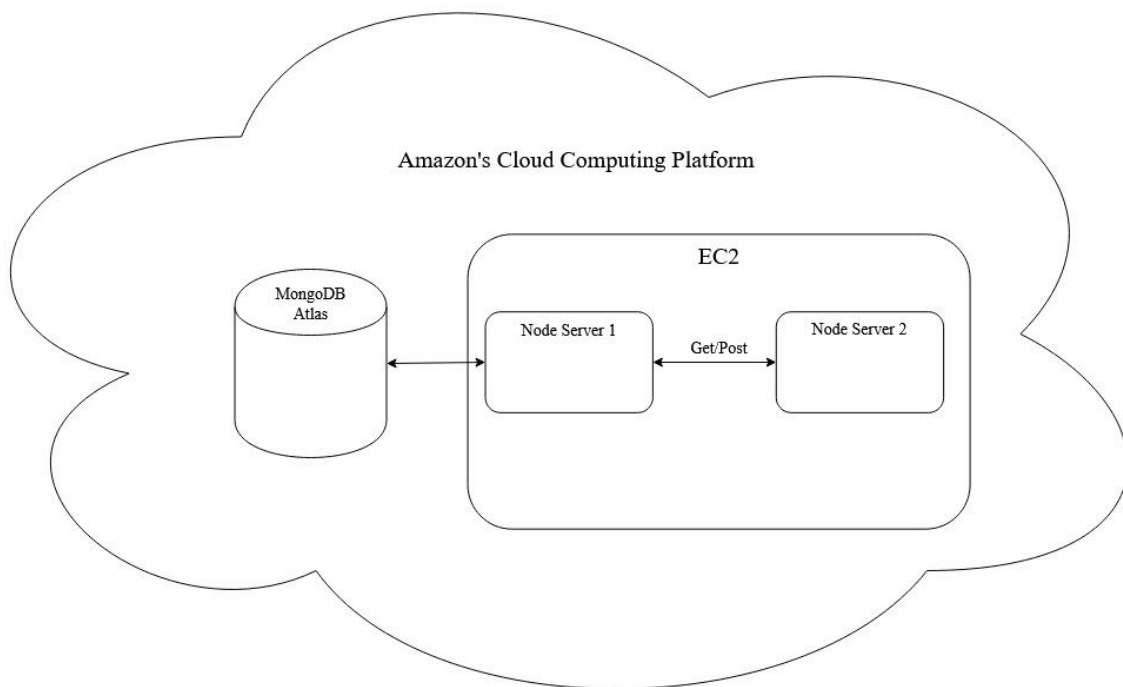


Figure 1: The architecture of the system.

Design choices

When creating the simulator we choose to use a restful api because of the advantages such as separation between the client and the server to make it easier to develop parts independently. Therefore, the simulator and the prosumer/manager file structures heavily differ since they do not have much to do with each other similarity wise. This was great because it was easy to keep the code where it belonged instead of having to mix it all together. We also used different files to set structures like the database schemas and used routes to redirect and separate the data flows, in order to not cluster the files with too much code.

The api is reachable by making a connection to a node.js server which is located on a virtual machine hosted by Amazon's cloud computing platform. We were told to use node.js in the project and we found it easy to use. The virtual machine, however, was an option but we chose it since it was a free option made available to us.

We also chose to use MongoDB Atlas for our database since we didn't have too much data to save, which meant that we could use the free cloud storage and thus simplify the system. We researched some different DBs and found that MongoDB was a popular choice and we were also looking to explore one we hadn't tried before. We really enjoyed the way it worked and how simple it was to learn, so we will definitely use it in further projects.

The prosumer and manager web applications are reachable by making a connection to a different node.js server, than the simulator, through a web browser and they are located on the same virtual machine as the simulator.

Scalability

Since the systems for the manager and prosumer updates every second, there is a lot of gets and posts being made all the time. This puts a lot of pressure on the server. Since we have limited funds and have to have the server up for a long time, we chose the free version, that unfortunately isn't powerful. This sometimes results in some lags on the server, and is something that would get increasingly worse on scaling, although a decent server would handle a lot more than right now.

Since we do not save too much data in our DB, the scaling there should not be a problem. We also tried to limit the amount of times we accessed the DB and the amount of http methods, but since we did the systems one step at a time, this didn't really work too well. In the future, one could try to optimize this process to use as few resources as possible.

In order to increase the scalability of the project, some areas that can be improved are memory, CPU, network I/O and disk I/O. More specifically this might be done by caching your database queries to find which ones run the most frequently and cache them in order to retrieve them faster. The database could also be improved by adding indexes which would reduce the time it takes to retrieve information of a particular request. To increase the scalability of the application, more servers and a load balancer could be implemented in order to handle more users being active at the same time and use the balancer to distribute the load evenly across the servers.

Security

We made use of JWT tokens in our project to protect the data from attackers that are not authorized to the specific data, such as name and password of the user. This means that to be able to see a site of a prosumer or manager, or even access a route of data flow, you will need to have a token, which you can only receive if you go through the login. In addition, a token will only be approved for 1 hour, then you will have to get a new one.

The password will not be stored in clear text anywhere, so even if you get access to the database and try to get a users password, you will not be able to since it will be hashed before it is stored.

The web application uses https with a self signed SSL certificate in order to encrypt the data being sent and received to prevent intruders from sniffing out important data such as passwords and names. Although self signed is safer than having nothing, there are disadvantages with it such as the certificate not being completely trusted by other applications and operating systems. The level of security with a self signed SSL is also lower than a trusted certificate.

Advanced features

We have not implemented any of the functionality apart from the basic requirements.

Challenges

Most of the challenges we faced occurred in the beginning of the project when we had to set everything up since we found it difficult to get everything working with each other. When, for example, we tried to access our node servers and failed we thought that the problem was located in the servers themselves, but instead it was the EC2 instance that stopped our traffic with security settings. So to only locate where the problem originates was difficult in the beginning.

Another challenge we faced appeared when we tried to encrypt the data sent back and forth between the node servers. We started with setting up the internet information service on our instance to be able to add a trusted certificate to the web application. However, to add a trusted certificate we needed to have a domain name and that's where we hit a brick wall. Our accounts on Amazon Web Services did not have enough privileges to add a domain name and to register a domain we needed to pay for it. To solve this we later added a self signed certificate instead of a trusted which was free and didn't require any outside services.

Future work

Some of the most important things that should be implemented in the future would be to improve the security of the system. It is not the worst, but as always it could definitely be improved, and you cannot get too much security.

Another important thing is the data flow, because right now there are probably some html functions that could be avoided by optimizing the flow. One could also research how different storages of the data could be implemented so that the data wouldn't have to be retrieved from the DB all the time. This could be done by saving to a file on the server or not use as many html functions.

The scalability could also be improved, so that the system can handle more users at the same time. The struggle with this now is a weak server and suboptimal handling of the data, but this could be improved and is an important part of the system.

The simulator is now quite unrealistic, and improvements could be done to have the wind change more realistically, by calculating the integral for example.

References

Web server: <https://aws.amazon.com/education/awseducate/>

Database: <https://www.mongodb.com/cloud/atlas>

The gaussian function we used:

<https://gist.github.com/supereggbert/fe5fb7b1fc30609e983b0207ae136707>

Youtube series of creating a rest API:

https://www.youtube.com/watch?v=0oXYLzuucwE&list=PL55RiY5tL51q4D-B63KBnygU6opNPFk_q

Links from the project description:

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

<https://restapitutorial.com/>

<https://www.guru99.com/postman-tutorial.html>

<https://www.w3schools.com/jquery/default.asp>

JWT tokens: <https://jwt.io/>

Scalability: <https://www.romexsoft.com/blog/improve-scalability/>

Appendix

Time report

Altogether we spent around 180 hours on this project. We spent the time pretty evenly on each part of the project but the prosumer took the longest to create and the manager the shortest, since we re-used a lot of the code from the prosumer and had learned a lot more at that time. We did not divide anything, instead we worked together on every part.

Contribution

Both of us always worked together and worked on one computer, so we both contributed an equal amount and both contributed to everything in the project.

Grading

We have only focused on the basic functionality for each part, which is why we think that we both deserve grade 3.

Github

<https://github.com/Narmista/M7011E-HR>

Please see readme.md file for instructions how to deploy and install.