LULEÅ TEKNISKA UNIVERSITET

Green Lean Electrics

Design of Dynamics Web Systems

Frenchs Group - *https://github.com/Green-Lean-Electric*

Emilie Borghino - emibor-9@student.ltu.se - +33 6 12 40 89 72

Maxence Cornaton - maxcor-9@student.ltu.se - +33 6 47 76 01 51

20 december 2019

Teachers part

.

.

.

.

.

.

.

Introduction

Context

We decided to create separately 3 different projects in order to modelize the 3 differents parts of the Green Lean Electrics project :

- Manager
- Simulator
- Prosumer

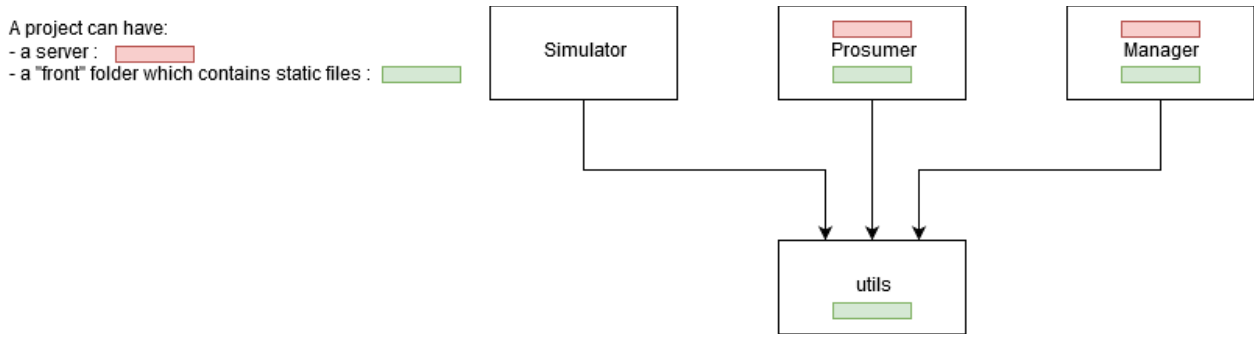
We have also created a common project called *Utils* which contains code used by the others such as the database connection.

Both of *Prosumer* and *Manager* run a different server with an API using Utils in order to connect to the database. They both have a client-side. For easy-to-use reason, we decided to put the server and the client-side in the same project: the server can then easily serve the static files which compose the front part of the application.

The *Simulator* project is in charge of all the calculations, such as the electricity production, consumption, price or the wind speed. It's an application which infinitely runs and computes each second the new values for the power plants, the prosumers and the market.

Architecture

Below is our architecture for this project.



Note that the project *Utils* has a "front" folder. It contains some common code that is used in the *Prosumer* project and in the *Manager* project (e.g. the application's stylesheet).

Design choices

First, we made a single project which contained a single server to work on the *Simulator*. Later on, when we had to start thinking about the *Prosumer* and the *Manager*, we have created these 2 new projects. As both of them had to contain a server, we thought about making a fourth project which would be able to centralize common code: here is how our *Utils* project is born.

Two of our projects (*Prosumer*, *Manager*) contain a "server.js" file which defines the static files that can be requested and the routes that lead to an action on the server. These files are the ones that should be executed to start the servers. They also each have a "service.js" file that is the actual implementation of the actions that can be executed on the server.

Manager and *Prosumer* have a "front" folder wherein the main and unshared static files are located. *Utils* has a "front" folder, too. It contains every shared code files (HTML, CSS, JS) and resources (mainly images).

This structure let us control which files should be accessed or not, which is a great security feature (see the "Security analysis" part below). In addition, our projects share code which makes the maintenance easier and the code better. Having three different servers is a great feature, too: it provides our system a better scalability (see the "Scalability analysis" part below).

The most important drawback is probably the not-so-structured file we finally have. *Simulator, Manager* and *Prosumer* only have two files each. We should have splitted them in more files so the cohesion in each file is greater but we didn't. It makes the code a bit harder to read than if it was well splitted.

Scalability analysis

Our application has not been designed around performance. However, it still gets some points when talking about scalability: we use MongoDB which has itself been designed to be easily scaled. The storage and recovery of our data is then a strength of our application.

Regarding the caching, it's up to the browsers to cache or not the requests. Our servers do not send any information telling whether the request should be cached. We think the browsers are optimized to cache what should be cached so we don't want to mess it up.

However, we can endure performance issues when dealing with a lot of simultaneous request. We have not done any test to know our current limit and this limit will probably depend on the computer we use as the physical server. As we currently use an OVH's VPS, it should be easy to

vertical scale it up if needed. Horizontal scaling would be a bit tougher: we can easily split the *Prosumer* server, the *Manager* server and the database on different machines but we will then need to open our database to external requests (please see the following part to discover our database security).

Security analysis

We have tried to build our application in a secure way. The main security features are:

- A user can only access his/her homepage while being connected. Otherwise he/she will be redirected to the login page.
- A new prosumer or manager should confirm his/her email address before his/her first login.
- The passwords are never sent in plain text. They are always encrypted before being sent and stored encrypted in the database.
- No user input is directly fed to the database. This prevents our application from an injection.
- A file sent to one of our server is only accessible for the user who sent it. Moreover, it is never executed on the server. This prevents Remote File Inclusions.
- A list of authorized files to be served is initialized when the servers start so they can only serve authorized files. This prevents Local File Inclusions.
- Our database is only accessible from localhost.

However, we know that our security is not as perfect as we would it to be. As an example, we use two external modules (*mongodb* & *formidable*). It's hard to tell whether they are 100% secure. The same goes for Node.js (we know this last one is not 100% secure). Moreover, there is no strict protection on the login: an attacker can try as many times as he/she wants when logging in. Neither do we have a monitoring system so we are unable to spot and fix security breaches.

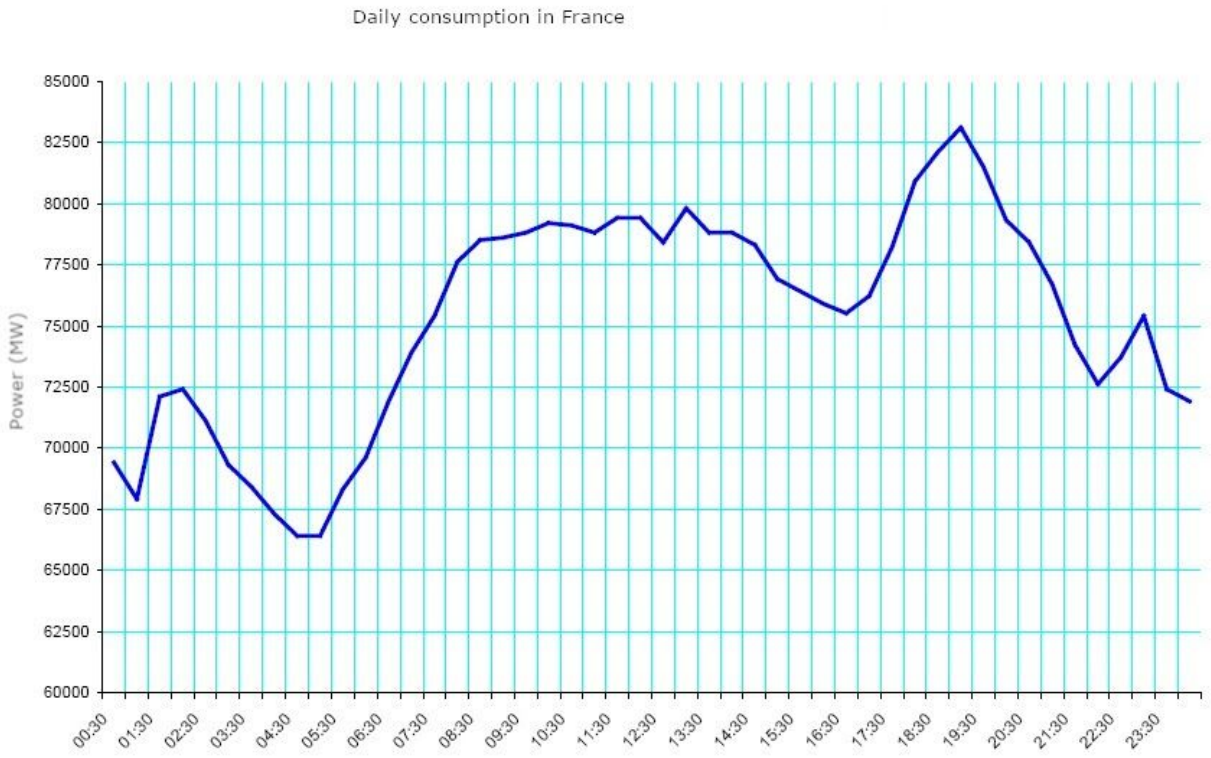
So, can we mitigate these possible attacks? When it comes to external modules or Node.js, we have to keep them up-to-date. Security updates are regularly published to prevent vulnerabilities to be exploited. Regarding our code, we could add a few security features to improve our application. These security features can be:

- Setting up a two-factor authentication;
- Using a secure protocol (HTTPS);
- ...

As we explain later in the *Future work* part, the manager registration is a security breach, too. However, it was designed in this way so the academic evaluation could be done.

Challenges

The first challenge we think of is the simulator: it was hard to get what exactly was expected. Should it be something realistic (using some well-known values) or completely made up? We chose to use some realistic values. After some research we found the following chart:



Following the line, we can assume that there are two peaks in the daily consumption: a first one at 11 A.M. which is stretched out and a second one at 7 P.M. which is more spiky. This means that we can represent the consumption as the sum of two gaussian functions. It took us some time to figure it out.

Another challenge was to find how to serve our static files while keeping some security. An easy way to let the client access to an application is to let him/her enter an URL and resolve this URL whatever it contains. This can lead to security issues (e.g. entering “domain.com/../../../../etc/shadow” can let a potential attacker read a sensitive file). So we add to define a whitelist of files that can be accessed by the client. This whitelist also let us access to files in other directory than the working directory: it can be used to serve shared files such as the stylesheet.

The third hard point was to understand what the goals and the feedback of each teacher were. It has sometime confused us when we were working on the debriefing of the meetings. The importance of this point has been increased by the online meetings: we think it's hard for our teacher to see what has actually been done and it's harder for us to understand someone through an online meeting. This last point is due to our not-that-great level of English but it was a real challenge!

Future work

First of all, the manager registration should be reworked. As of now, managers can register themselves. That's a security breach as a manager can access to prosumers' data. It was set up this way so it's easy for a teacher to try our application. However, it should not be done in this way in a real system. We should be the only ones to be able to add managers.

An important point to improve is probably our error management. They are currently logged to the console so they can only be accessed from the console. Writing these errors to a permanent file would help us detect and fix bugs.

A third thing we could add is the simulation of many cities. Our system currently simulate only a city (= a power plant), so a single manager is needed and prosumers are all linked to this unique manager. In a real system, it would be a good thing to be able to manage different cities, each one with its prosumers and managers.

On another note, we could think about the geolocation of each prosumer: the current calculation is the same for all so they all produce the same amount of energy. This is not realistic. Taking into account each prosumer location would let them having different wind speeds so they could produce different amounts of energy.

The last important thing we could work on is the buffer management. A prosumer's buffer is used only when the prosumer asks for it: when a blackout occurs, the prosumer should change its consumption parameter to adapt to the situation. If he/she has enough power in its buffer, he/she should explicitly set its consumption ratio so that he/she consumes his/her buffer. Otherwise, no electricity will consume at all. The same goes when it comes to consumes from the market: if he/she does not produces enough electricity and its parameters are set so he/she should consumes from his/her buffer, he/she won't buy electricity from the market if his/her buffer is not filled with enough electricity. This feature lets the prosumer control what he/she produces and buys but it is not what we could call "smart". Instead, the ratio could update itself so the prosumer does not have to worry about a potential blackout when there is enough power.

Appendix

Time reports analysis

Emilie

The beginning of the project was quite catastrophic, we were lost in the middle of all this information and we did not know where to start. You must have feel it through our many emails and questions on the subject.

Time for each goal was not indicative of the expected difficulty. We finished setting up the server very quickly, so fast that we were afraid of missing something and tried for hours to check that we had not missed a step.

So we spend a lot of time on “reflexions”, trying to understand the goals and how to achieve them, trying to anticipate further changes in order to have a modifiable architecture but this was really complicated and finally quite everything in our architecture have changed.

The other thing we spend a lot time on is the calculations, we are really bad at mathematics and the world of electricity is unknown for us so it’s was really hard to imagine how much we can produce/consume and how to simulate it.

Maxence

I have worked between 90 and 100 hours on this project during these 7 weeks. One of the longest tasks (between 20 and 30 hours)) was to design the simulator in such a way it would be realistic. Having a realistic model was not the main goal but I think it was crucial so we can have some interesting values and behaviors (a prosumer consumes more than what he/she produces, he/she produces more than what he/she consumes or he/she has a blackout).

I have spent a lot of time to help Émilie on her parts, too (about 15 hours). My JavaScript knowledge is bigger than hers so I had some tools to help her going forward.

The rest of my time has been spent on developing content and fixing bugs.

Project contribution

Emilie

I worked a lot on the “development part”, Maxence has better knowledges and experiences in architecture design and implementation. So my part of the job was to develop features on the project, i did all the front end on both the manager and prosumer side. On the client side i worked a lot on the web services too and database requests.

Maxence

While Emilie has worked on the front-end part, I have worked on the back-end part: setting up a good project architecture, introducing the database access, normalizing some methods (such as server creation, static files access, ...), providing a live server and fixing a lot of bugs.

In addition, I have written most of this report after having talked about it with Émilie.

Grade

Emilie

I think our work is quite good, all the “basic functionalities” are implemented, technical requirements are also well respected. We prefered to focus on the main functionalities, security and quality attributes instead of trying to do the most advanced functionalities we could. I think our work could have been better if the subject wasn’t as complicated. It was really hard to understand the goals of this project it was really confused at the beginning with a lot of things to do but we didn’t know how. We also had a lot of difficulties on the calculations, first because our mathematics skills are not really good and then because of the complex world of electricity (with units / conversions etc...), which is completely different than our expertise field. So we finally forgave to respect these “real world similarities” to concentrate on the rest of the project and main functionalities. But we lost a lot of time on it and because of all these misunderstanding we didn’t well design our system at the beginning and we had to change a lot of things during the project.

Finally as all the main requirements are respected and working, i think we should have a grade 4.

Maxence

I think a fair grade would be 4. It has been explicitly told that we had to do as much work as the other groups even if we had to leave three weeks earlier. I think - and I hope - we achieved to do so in 7 weeks instead of 10. Even if we have not done a lot of advanced functionalities, I think our solution is still great and I hope you will feel the same!

Deployment and Installation instructions

Documentation

For the Manager API : https://documenter.getpostman.com/view/3311500/SWECVugc

For the Prosumer API : https://documenter.getpostman.com/view/3311500/SW7Z3oT5

Deployment

Below are the instructions regarding the deployment of our application.

Download and install Node.js version 12 or higher. Lower versions may not work due to advanced JavaScript features. Download and install on localhost MongoDB version 4 or higher. Default port should be 27017.
Our projects should be put in the same root folder. The directory tree should look like:

```
root folder
|
|---- manager
        |---- package.json
        |---- src
                |---- server.js
                |---- services.js
                |---- front
                        |---- * Bunch of files*
|---- prosumer
        |---- package.json
        |---- src
                |---- server.js
                |---- services.js
                |---- front
                        |---- * Bunch of files*
|---- simulator
        |---- package.json
        |---- src
                |---- server.js
                |---- services.js
|---- utils
        |---- package.json
        |---- src
                |---- configuration.js
```

```
|---- mongo.js
|---- server.js
|---- front
        |---- * Bunch of files*
```

Once every project is downloaded, go inside each folder (manager, prosumer, simulator and utils) and run `npm install`. It should install Node.js dependencies we have used throughout our application.

You can now start the three main projects. From the root folder, launch the following commands:

1. `node simulator/src/server.js`
2. `node manager/src/server.js`
3. `node prosumer/src/server.js`

The servers should now be up and running and you should be able to access to your web interfaces at "localhost:8081" (prosumer interface) and "localhost:8082" (manager interface).

Please note that if you test our application on a Windows computer, the registration mails can't be send. We recommend you to test it on a Unix computer. However, if you only have a Windows computer, you need a few more steps to activate an account (manager & prosumer) after its registration:

1. open your localhost MongoDB "greenleanelectrics" database.
2. look for the prosumer (collection "prosumer") or the manager (collection "manager") you want to activate.
3. delete the "registrationToken" field.

The application is currently running on the following addresses: http://145.239.75.80:8081/ for the prosumer interface and http://145.239.75.80:8082/ for the manager interface.