Feature request - Group DROP TABLES

Gustav Rixon, gusrix-8@student.ltu.se
Edison Widell Delgado, edidel-8@student.ltu.se

Github for project - https://github.com/Gustav-Rixon/M7011E

Contents

1 Introduction

This report is done in correlation to completing the project in course M7011E. The goal for this course is to implement a dynamic web system and an API that is based on a simulation data generator. The project task was to solve a list of predefined specifications that requires the implementation and usage of different components and a well defined API. This challenge includes creating a web application for a electricity system network, with a simulation of several privately owned wind turbines with their own household and a power plant, which all are connected to a community market. The project task was to be done in teams of two.

The system has three types. The first type is the prosumer. A prosumer has a private household with their own production. They can overview their production, consumption and information about their households like wind speed and temperature. If the prosumer has a positive net-production, then they can choose how much of it is sent to the market buffer.

The consumer can only consume electricity. The admin controls the system. Admins can start and stop a power plant set its production and change how much of its power should be sent to the market. They can also block the prosumer for selling for number of cycles and view their details such as if they are blocked or not. The admin interface is only available to the admins.

2 Methods

2.1 Frontend

For the front end, the thought was to use something that's made for the specific task. Therefore the decision we landed on was to use a programming framework React. React can be written using Javascript and allowing the usage of prebuilt packages. As a security measure we have an intermediate API that acts as a front-end server communicates with the API in the back-end where a that handles the hashing of passwords and sees keeps record of who's online and all the calls to our back-end API. Our application has 6 different views:

- /login where a user can log in.
- /register where you can register.
- /admin where admins log in.
- /adminpage dashboard view for an admin.
- /loginpage dashboard view for users.

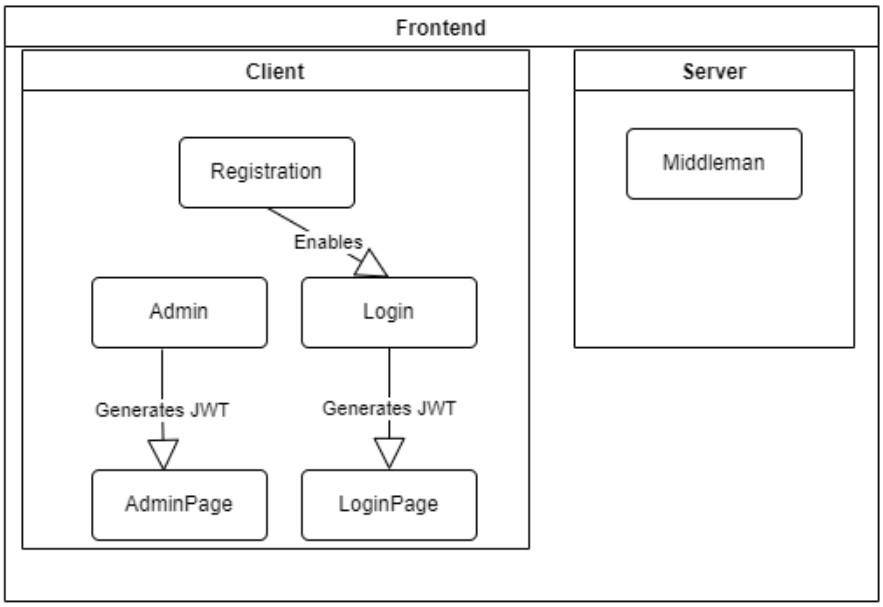

Figure 1: Module diagram of the front-end

2.2 Backend

The backend is build using Python and uses real temperature data and wind data from SMHI's API.
It also uses Nominatim for geocoding so that it can find where it should take its information from.

The backend consists of

- The simulator
- API endpoints
- Database communication

The simulator part is the heart of this program. Here the simulation of the power grid is created and controlled. The simulation logic can be seen in figure 3
The simulation is can be accessed through the API that is written with Werkzeug which is an WSGI web application library.

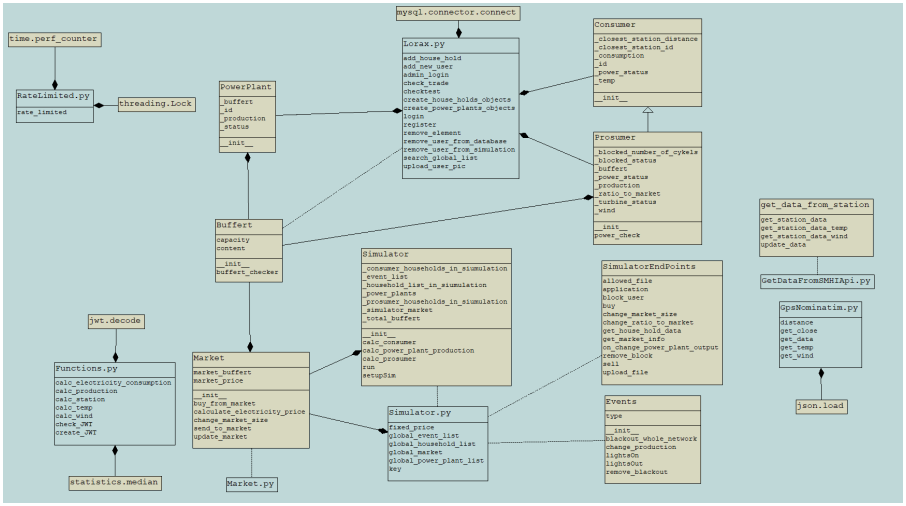

Figure 2: UML diagram of backend

In the SimulatorEndPoints class we create our API endpoints by creating a map structure that binds a URL to an method.
The request is then handled by the werkzeug controller and served to that method.
Example request:

- User id 70 is requesting to change how much of its net production is sent to the market through the endpoint change_buffert_to_market
- werkzeug receives the request and dispatches it to the corresponding method.
- The method handles the request accordingly and then sends a response to the requester.

For does endpoints that contain user data or is admin based have JWT authentication to make sure that only authorised users can view that data. Authorised in that user 1's information can be accesd with user 1's JWT or the admins JWT. The JWT from the created in the and stored in the users browser local storage. And because the backend knows the secret key it can decrypt it and check if its a unauthorised or authorised user that is trying to request the data. Our JWT is signed using the HS256 method.

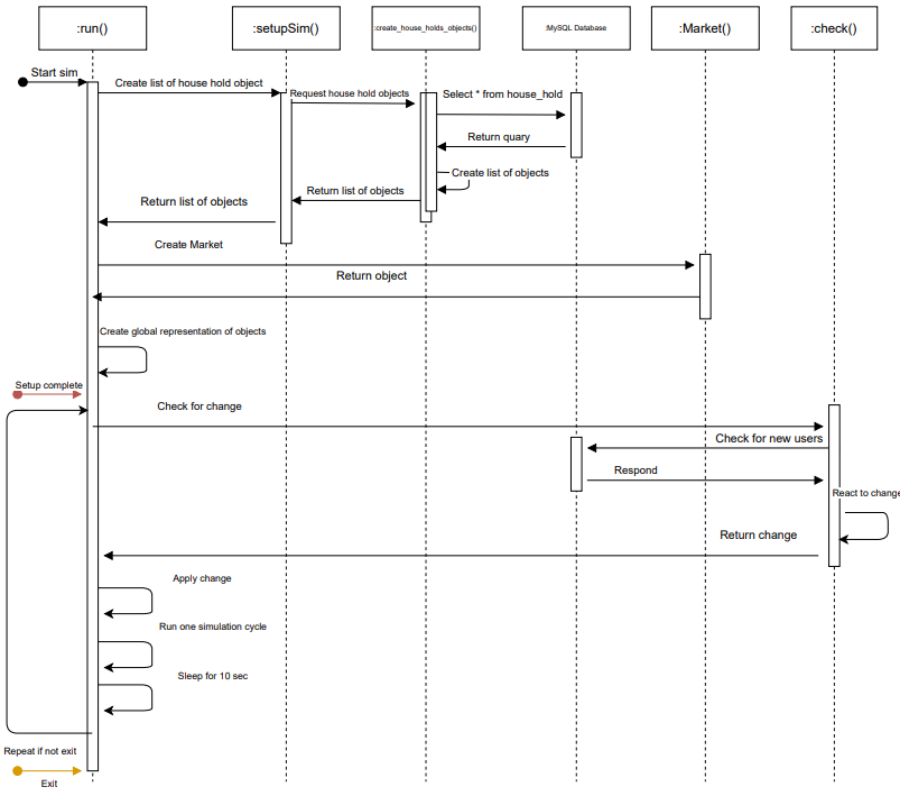

Figure 3: Sequence diagram of simulator

3 Results

3.1 Frontend

3.1.1 User registration and login

The front-end starts with a login page where you can either use your existing account or create a new account by going to the register page. When creating an account there are some requirements:

- Unique username that doesn't contain any special characters and isn't an empty string
- Email must contain an @ and a . to be accepted.
- Password that isn't an empty string.
- Address must match a real life address and there must be exact one match to the nominatim public api there must be a matching zip code (nominative guesses zip code by geographical calculations so it doesn't always match real life zip code).
- Prosumer is defaulted to not activated when you register but there is a checkbox that activates it.

If all of these requirements are fulfilled registration should be done without issue and would there be an issue text will appear that specifies where the error is located.

Same with login if something goes wrong there will be text saying that the user doesn't exist or that there is a mismatch with the username and password.

3.1.2 Prosumer and Consumer

Both can upload/change their profile picture and logout.

On the **prosumer's** page the user can view their household information:

- Production - Displays the amount of production in kWh.
- Consumption - Displays the amount of consumption in kWh.

- Net Production – Displays the difference between the production and consumption in kWh.
- Wind Speed - Displays the current wind at that location in m/s.
- Temperature - Displays the current temperature in Celsius.
- Buffer – Displays the current amount of electricity in the buffer.
- Current market price - Displays the current market price of 1 kWh
- Sell - lets you sell from your personal buffer and send it to the market.

On the **consumer**'s page the user can view their household information:

- Consumption - Displays the amount of consumption in kWh.
- Current market price - Displays the current market price of 1 kWh

3.1.3 Admin

On the admin page, admins can inspect parameters such as:

- See a list of the users in the simulation displaying all the data of all users same as a producer and if they have a blackout.
- See a list of power plants id, containing their ratio, production, status and id.
- See a list of market info such as recommended electricity price, current electricity price, market capacity and current market content.
- See currently logged in users.

The admin can make different choices during the simulation, such as blocking a user from selling to the market for a chosen number of cycles. They can also choose how much of the power plant electricity that should be sent to the market and the buffer.

Admin can delete users, change their registration data such as prosumerstatus, address/zipcode, password and username. They can set the amount to be produced by the factory, upload a picture and send power from the power plant buffer to the market.

3.2 Backend

The backend is divided by into three function calls. The first calls create threads one thread that the simulation runs on and on thread that runs the API communication to SMHI and updates our real-life data. The last call is to the werkzeug API that runs as the main thread.

The simulator begins with a database request about the users and power plant. It then creates three lists of objects, one for the prosumers, one for consumer and one for power plants. Currently the simulator needs one power plant to be present for a simulation to begin. It also only supports one power plant to be present to, but functionality as been designed around the possibility to support more power plant.

The simulator then saves this object to global representation and updates their state and data each cycle depending on input from the user and admin but also the state of the simulation.

A market is represented as an object that handles all the transactions in the simulation. You can sell, buy and view what the current and recommended price is.

The simulator does not save any data of the simulator.

The API was first written with flask but was then rewritten using only werkzeug. This resulted in a light weight API structure that can easily be build on in future releases.

3.2.1 Additional implementations

The exercise had some requirements and below are some things we chose to implement to make it more advanced:

- Temperature are used in the calculation of consumption
- Use of real whether data from SMHI
- Use of real GPS data from Nominatim

4 Architecture

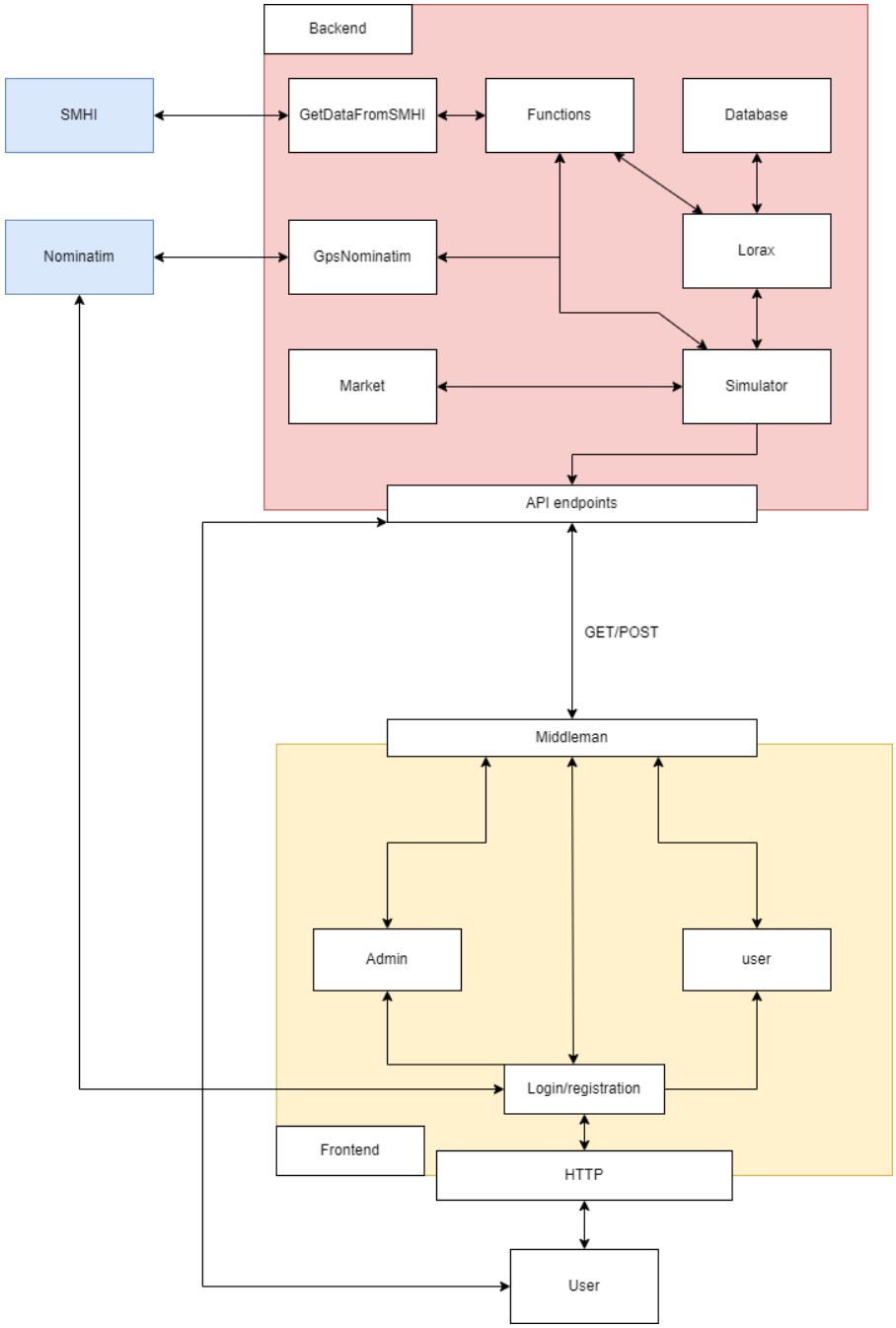

Figure 4: Program architecture

Since the goal was creating a web application React was a great choice and it has possibilities to reuse components in a good way to save time.

React also has a large amount of different packages that you can implement within your application

The frontend has its own api called middleman that encapsulates the backend api and protects it from harm full requests. It also generates the JWT key and hands it to the users/admin on login. Last but not least it handles the list that keeps track of who is currently online and logs them out once their JWT is expired.

The backend consists of python and a MySQL database

When it comes to the database we chose MySQL because of how you can create relations between objects in different tables.

For the security we are using JWT with HS256. Which allow us too add user identification to the data streams of information between parties.

This means you can somewhat ensure that the senders are who they say they are.

5 Discussion

5.1 Difficulties

5.1.1 Frontend

Showing the picture from an external file to the frontend folder was nearly impossible using react but by converting it to a base 64 value it got shown without issue. But sending the image via the middleman seemed to be impossible as using hooks to send a formdata while saving its contents was quite difficult and we were unable to solve it.

5.1.2 Backend

Werkzeug was hard to understand and get to work at the beginning. There documentation is very bare bones and does not give a lot of pointers. And a lot of the things online about it is about flask. So a way to do it was to do it In flask first and then convert it to pure werkzeug.

5.2 Security

Passwords are encrypted using a hashing method md5, then a static salt value is added to be hashed again and saved inside the database. To ensure that the right person is logged in we use a JWT session token in our cookie header. This JWT token is given to a prosumer/manager when logged in with a determined time to live, this JWT token is unique to that person's account providing some security for that user and API endpoints.

5.3 Future work

5.3.1 Frontend

Code refactoring. As there is an issue with having multiple users active in one browser. So if one logs out all of them get logged out due to clearing the browsers local storage. A quick fix to this would be to have a auto log in if there is a valid JWT. As for aesthetics there is still a lot to work with.

5.3.2 Backend

There's still work to be done in the backend, things work as intended now but can still be improved. The structure of the code has suffered and need to be looked over.

5.4 Grade

Both of us are looking for a higher grade then 3.

Appendices

A Link to projekt code

- https://github.com/Gustav-Rixon/M7011E

B Website hosted on LUDD Distributed Systems and Technologies (DUST)

- 130.240.200.31
- Admin credentials for program - user:admin, password:admin
- root credentials for server - user:robi, password:SEE_COMMENT_IN_CANVAS_ROBERT