Wind production

Group name - FannySara
Fanny Blixt - fanbli-6@student.ltu.se
Sara Andersson - saranj-6@student.ltu.se

Link to website - http://18.212.138.12:3000/
Link to release - https://github.com/saraandersson/saraFanny/releases
Github - https://github.com/saraandersson/saraFanny

Table of contents

Introduction 3

Simulator 3

Prosumer 3

Manager 5

Design 6

Scalability 6

Security 7

Challenges 7

Future work 8

References 8

Appendix 8

Time analysis 8

Contribution of each member 9

Grade for each group member 9

Installation instructions 9

Appendix 1: MySQL database code 11

Introduction

Simulator

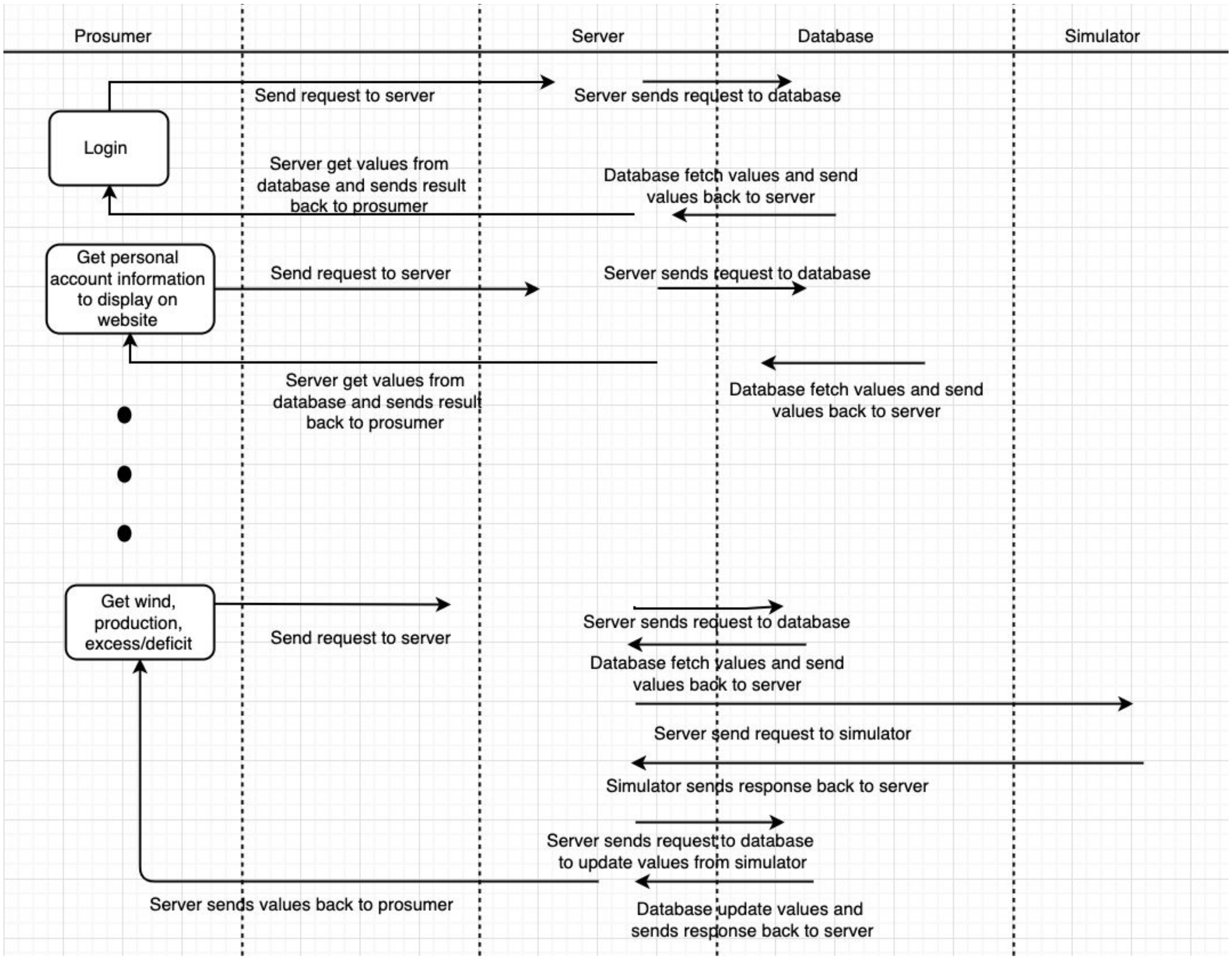
The simulator is a wind simulator that simulates a wind and then calculate a production for the prosumer based on prosumers area and consumption. The simulator also simulates a price that the manager can use to choose an overall electricity price for the market.

Prosumer

Every prosumer in the system have a own wind simulator that simulates a wind and calculates the production depending on the power from the wind, the prosumers house area and the prosumers consumption. Each prosumer have a own buffer which contains electricity that the prosumer can use if there would be a deficit result from the production, if there would be a deficit the prosumer can also buy electricity from the market. If the production would result in excess then the prosumer can choose an amount that should be sent to the buffer and an amount that should be sent to the market.

The basic functionality of the prosumers part of the system can be described by the flowchart below.

Prosumer flow chart:


Prosumer
Server
Database
Simulator
Send request to server
Server sends request to database
Login
Server get values from database and sends result back to prosumer
Database fetch values and send values back to server
Get personal account information to display on website
Send request to server
Server sends tequest to database
Server get values from database and sends result back to prosumer
Database fetch values and send values back to server
Get wind, production, excess/deficit
Send request to server
Server sends request to database
Database fetch values and send values back to server
Server send request to simulator
Simulator sends response back to server
Server sends request to database to update values from simulator
Server sends values back to prosumer
Database update values and sends response back to server

Flowchart 1. Prosumers part of the system.

The flowchart above describes the functionality of the prosumers part of the system. All the basic functionality, i.e, login, gets personal information, update personal information and sign up have the same functionality in the system. As the flowchart above shows these basic functionality works as follow, the request from the prosumer is sent to the server, who sends it request to database, the database adds/updates/fetch information depending on the request, then the database sends a response back to server who sends the response to the prosumer.

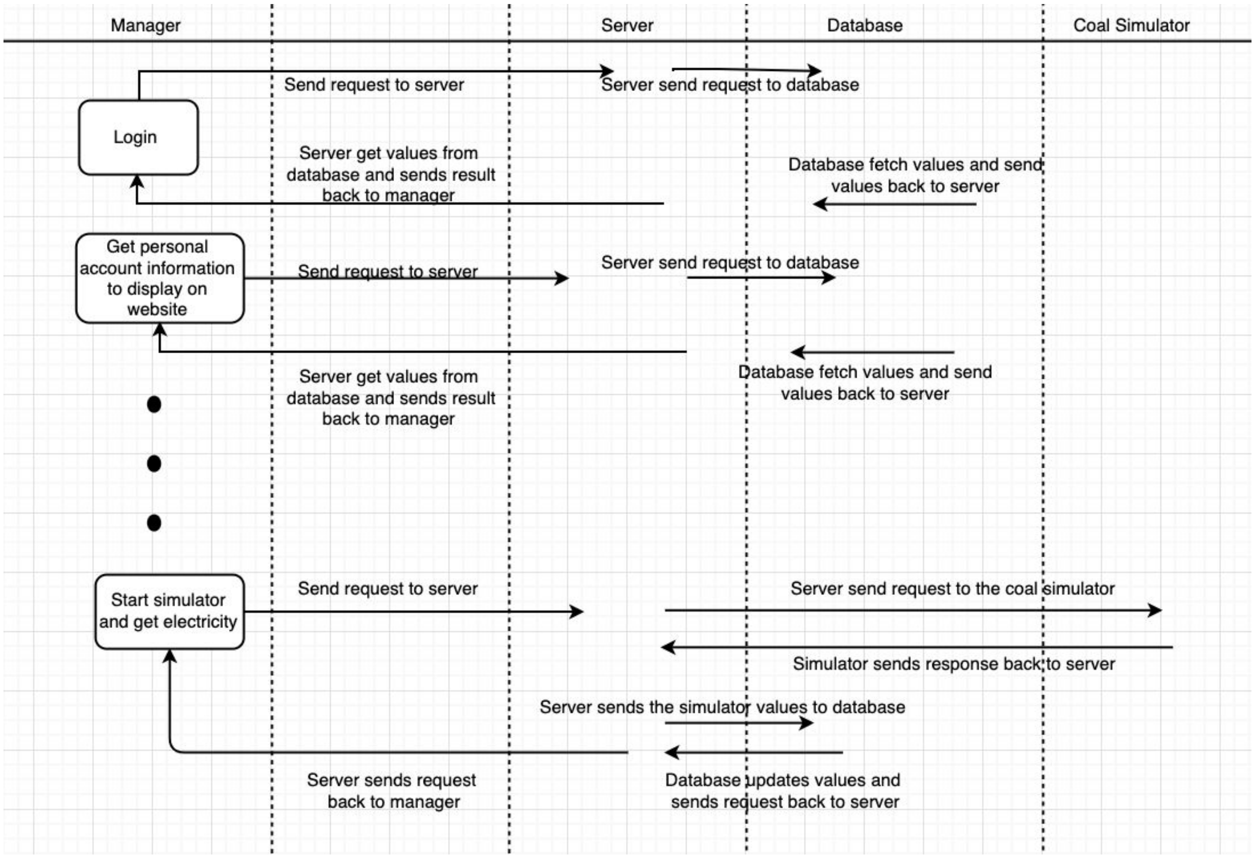
The functionality that differs in the prosumers system is when the prosumer wants to get information about the wind production. Then the request is sent to the server, the server sends the request to the database and the database fetches values as area and consumption and sends the response back to the server who sends the request to the simulator, the simulator simulates the wind and calculates the production and sends the result back to the server who sends the response to database that updates values as the buffer and the market. The database then sends the result to the server who sends the response back to the prosumer. This request is updated automatically every ten seconds.

Manager

The manager controls the market, the overall electricity price and also contains information of each prosumer. The manager can set the electricity price based on the prosumers simulated price or choose one of its own. The manager also has an own coal simulator which can be used if the market would need more electricity. By using this coal simulator the manager can (as prosumers) choose to send an amount to the managers own buffer or to the market. The manager can also block a prosumer from selling to the market.

The basic functionality of the managers part of the system is described in the flowchart below.

Manager flowchart:


Manager
Server
Database
Coal Simulator
Login
Send request to server
Server send request to database
Server get values from database and sends result back to manager
Database fetch values and send values back to server
Get personal account information to display on website
Send request to server
Server send request to database
Server get values from database and sends result back to manager
Database fetch values and send values back to server
Start simulator and get electricity
Send request to server
Server send request to the coal simulator
Simulator sends response back to server
Server sends the simulator values to database
Server sends request back to manager
Database updates values and sends request back to server

Flowchart 2. Managers part of the system.

The flowchart above describes the functionality of the managers part of the system. The basic functionality for the manager, i.e, login, gets personal information, update personal information, delete user, see prosumers production and set electricity prices all have the same system functionality. As described in the flowchart above these functionalities works as follows, the manager sends a request to the server, the server sends request to the database who adds/updates/fetch information depending on the request and then sends the response back to the server who updates the manager.

The functionality that differs in the managers system is the coal simulator functionality. As shown in the flowchart above the manager sends a request to the server to start a coal simulator, the server sends the request to the simulator, who handle the request and sends back the response to the server, the server sends the response to the database that updates values and then the database sends the response back to the server who sends the request back to the manager.

To see the manager pages on our site, use username:**admin** and password:**admin**.

Design

We choose to structure our system in three major parts, prosumer, manager and wind simulator. And we also chose to divide the website, server, simulator and database functions in different modules, this was a benefit since it made it easier for us to work and also to locate different functionalities. We also made the decision to use an intermediary between the database and the server and that also benefits our project since the requests became synchronized.

We also chose to have an own wind.js file where the wind and the electricity price was simulated, this also also benefited our project since it made is easier for us to be able to change the calculations of the wind and also made it easier to create a new wind for each prosumer.

A drawback with our system design is that we could have divided our system functionalities into smaller files and modules, this would improve our design since it would make it easier to locate different functionalities and also make it easier for future developers to understand and develop our system.

Another design choice that we made was that we chose to use object oriented design when created the coal simulator and the wind simulator, this benefited our project since it made it easier for each prosumer to have its own wind production in the system and also made it easier for the manager to have several coal simulators.

Scalability

To be able to handle a very high amount of users we chose to store every image as a URL in the database instead of storing the picture in the database. Storing the picture in the database would take a great amount of storage and might compromise our database. This solution also gives the system a better performance.

To improve our scalability we could choose to use more servers and also have the database located on another server. This would improve our scalability since out would prevent the systems performance issues and also network trafficking, it would also improve our solution with handling a very high amount of users.

Security

From a security standpoint, we chose to use hashed passwords when a new user is created in the system and also store the hashed password in the database. To use hashed passwords helps us to protect the user's data on the website.

We also chose to use cookies and session id:s instead of storing the id in the URL. This also made our users data more secure since it prevents users in the system to access other users sites and data by typing an other id in the URL. To also prevent that users do not access sites that they do not have privileges too, we chose to have different roles in the system and use a block that for example prevents a prosumer to access the managers sites.

Due to lack of knowledge one flaw in our system could be the configuration of security groups on our server. We configured the security groups so only a few ports are open and accessible to other users, but if we missed something when we configured these groups there could be possible attacks to our server.

Challenges

One of our major challenges with our project was the configuration of the server and the database. At first we created a server with linux environment and also a PostGreSql database, but since the database did not work together with the server we got help with setting up a new server with an ubuntu environment and setting up a mysql database instead. This took us a great amount of time and also caused that we did not have time to focus on the higher-level functionalities in our project.

Another challenge that we experienced in this project was how we were suppose to handle the different time intervals. For example when a prosumer is blocked for a certain amount of time, both the prosumer and the manager needs to have knowledge about for how long the prosumer is blocked. At first we thought it was hard to know where we should locate the timers in our code and also to make sure that the timers would not collide with each other, but after a lot of testing we came up with a solution that worked for both the manager and the prosumer.

Future work

One suggestion of a future work of our project could be security work on our system where a security check would be needed. This is important since our project is supposed to store prosumers data and also data related to manager, if this data is not secure there might be a risk that our system fails. Since our system also handles a market business it is also important that the market is handled correctly and if our system would have any security flaws then the market functionality might be compromised.

Another suggestion of future work of our project could be scalability, since our project and the database is located on one single server the project could improve its scalability. This is important since if many users would use the system this would improve the systems standard.

References

References to material you have used as a base for your work

Setting up nodejs server- https://www.guru99.com/node-js-tutorial.html

Use hashed passwords in nodejs - https://www.npmjs.com/package/bcrypt

Use multer in nodejs - https://www.youtube.com/watch?v=9Qzmri1WaaE

Setting up database credentials - https://www.w3schools.com/nodejs/nodejs_mysql.asp

Appendix

Time analysis

As can be seen in our time reports we have put about 20 hours a week on this project to be able to finish in time. Most of our time on this project was spent on setting up the server and the database, and also the connection between the server and the database. After the server and the database were working and connected we started dividing the project and work on our own. When we experienced a bigger issue or bug in our system we planned a meeting and solved the issue together. We also kept a backlog where we wrote down the tasks that needed to be done and sorted them by priority, when the task was done, we marked it green in our backlog and this helped us keep track of what was needed to be done in our system.

When a task was done, we also made sure that each member of the group tested the task, so it would work properly and if a bug where detected one member of the group solved it and continued with the testing.

Contribution of each member

Contribution of each member have been equally distributed where at the beginning of the project we worked together to set up the project on the server and the database. Later on we divided the work between each member. For more details see our time reports on github.

Grade for each group member

Fanny - Grade 4
Sara - Grade 4

Installation instructions

The github link to our project: https://github.com/saraandersson/saraFanny
The github link to our release: https://github.com/saraandersson/saraFanny/releases

1. To be able to deploy and run our project you first need to install nodejs and all nodejs packets that is used on your local machine
    a. To install nodejs: https://nodejs.org/en/download/
    b. After installing nodejs you need to install following packets:
        i. npm init -y (Creates package.json)
        ii. npm install express --save (Creates API endpoints)
        iii. npm install cookie-parser --save (Use cookies)
        iv. npm install body-parser --save (Parse incoming request bodies, returns middleware that only parses json)
        v. npm install bcrypt -- save (To use hash passwords)
        vi. npm install multer --save (To store images)
2. To be able to deploy and run our project you then need to download the project from github.
3. To be able to run the project you also need to be able to have a mysql database on your machine, instructions to install mysql database can be seen here: https://www.w3schools.com/nodejs/nodejs_mysql.asp
    a. When you have installed mysql you create a new database and run the code written in Appendix 1.
    b. When your database is complete you need to change the database credentials in the file located at db/db_users.js, at the top of the file you need to change to your database credentials, for example password and database.

4. After the installation is done you need to enter the saraFanny folder you downloaded from github and type the command:

   ```
   node server
   ```

5. The project can then be seen in a url (depending on your server url) at port 3000.

Appendix 1: MySQL database code

```
CREATE TABLE users (
  id INT NOT NULL AUTO_INCREMENT,
  role_id INT NOT NULL,
  online INT NOT NULL,
  firstname VARCHAR(50) NOT NULL,
  lastname VARCHAR(50) NOT NULL,
  email VARCHAR(50) NOT NULL,
 password VARCHAR(250) NOT NULL,
  img VARCHAR(50),
  area INT NOT NULL,
  buffert INT NOT NULL,
  consumption INT NOT NULL,
has_power INT NOT NULL
  PRIMARY KEY (id)
) ENGINE=INNODB;


CREATE TABLE blocked (
users_id INT NOT NULL,
blocked INT NOT NULL,
time DOUBLE NOT NULL,
INDEX par_ind (users_id),
CONSTRAINT fk_users FOREIGN KEY (users_id)
REFERENCES users(id)
ON DELETE CASCADE
ON UPDATE CASCADE
)ENGINE=INNODB;


CREATE TABLE user_production (
  user_id INT NOT NULL,
  total_production DOUBLE NOT NULL,
 total_excess DOUBLE NOT NULL,
latest_production DOUBLE NOT NULL,
latest_wind DOUBLE NOT NULL,
latest_excess DOUBLE NOT NULL,
sim_price DOUBLE NOT NULL
  INDEX par_ind (user_id),
  CONSTRAINT fk_user FOREIGN KEY (user_id)
  REFERENCES users(id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE user_sell_buy (
  user_id INT NOT NULL,
  sell INT NOT NULL,
  buy INT NOT NULL,
  INDEX par_ind (user_id),
  CONSTRAINT fk_usersellbuy FOREIGN KEY (user_id)
  REFERENCES users(id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
) ENGINE=INNODB;

CREATE TABLE market (
price DOUBLE NOT NULL,
amount DOUBLE NOT NULL,
price_sim DOUBLE NOT NULL
) ENGINE=INNODB;

CREATE TABLE coal (
  id INT NOT NULL AUTO_INCREMENT,
  user_id INT NOT NULL,
  status INT NOT NULL,
  INDEX par_ind (user_id),
  PRIMARY KEY (id),
  CONSTRAINT fk_us FOREIGN KEY (user_id)
  REFERENCES users(id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
) ENGINE=INNODB;
```

```
CREATE TABLE coal_power (
coal_id INT NOT NULL,
  date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  time DOUBLE NOT NULL,
  production INT NOT NULL,
  status INT NOT NULL,
  INDEX par_ind (coal_id),
  CONSTRAINT fk_coal FOREIGN KEY (coal_id)
  REFERENCES coal(id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
) ENGINE=INNODB;
```