M7011E

André Christofferson - andchr-6@student.ltu.se
Wilma Krutrök - wilkru-7@student.ltu.se

January 14, 2022

Contents

1 System Specification

Our system is built on seven services and a website which are all running an express server using Node.js. All services utilizes the same MongoDB database with one collection for the market and another collection for the users. The different services are described in more detail below.

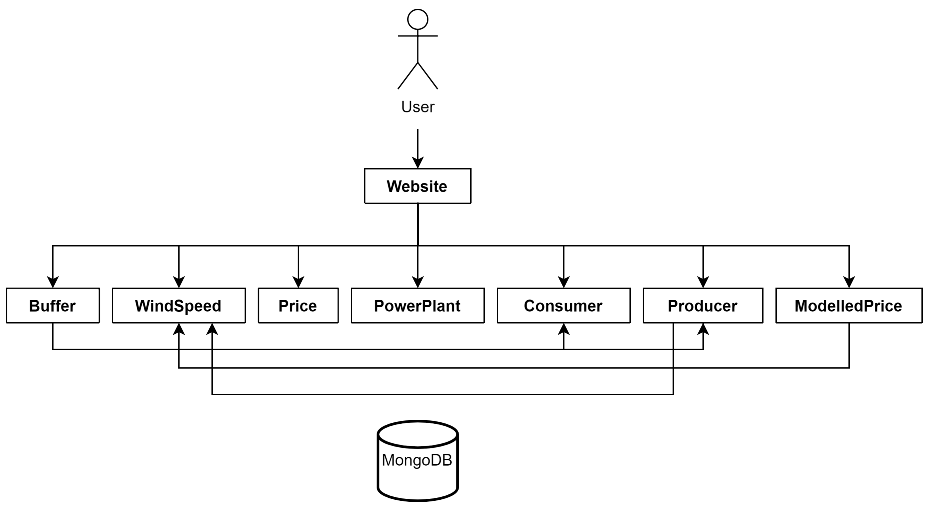


Figure 1: System Architecture

1.1 Website

The website is responsible for presenting the user with the user interface and authenticating the user. When a user has registered and logged in to the website requests are made to each microservice to collect the simulated data that should be presented, that means that no simulation or heavy computational simulation is done in the website. The front-end is built using the view engine Embedded JavaScript templating (ejs) together with the express server to provide the user interface. The data is refreshed on the client using ajax and Jquery which creates calls to the server to update the data without the need to refresh the page.

1.2 Producer

The Producer service is responsible for creating simulated values for producing energy for each user. It is possible to start a new user through the API which makes the producer service simulate values for that user with a rate of 1 Hz and then stored in the database. The Producer service is also responsible for retrieving the production value for the user and is accessible through the API.

1.3 Consumer

Similar to the producer service does the consumer service create simulations of the consumption for each user. This service retrieves the consumption values and it is possible to start a new user through the API.

1.4 Buffer

The buffer service works just like the producer- and consumer service but produces simulations for the buffer instead. The values put into the users buffers is calculated using the net production. The ratio of how much of the net production that should be sent to the buffer/market is set by the user on the website which communicates this to the buffer service through its API.

1.5 Powerplant

This service is used to manage the status of the power plant, to sell and buy from the managers buffer and to set the ratio of how much of the power plant production should go to the buffer and to the market.

1.6 Modelled Price

To help the manager set the electricity price is a modelled price calculated using the market demand and the current wind speed.

1.7 Price

The current electricity price is set using this service to later be retrived and displayed on the website.

1.8 Windspeed

This service calculates values for wind speed using two gaussians distributions. The first distribution is used to set a mean value for every given day and then the second distributions uses this mean to update the windspeed every hour.

1.9 API

The tables 1 and 2 give a better understanding over the communication between the services represented in figure 1 by showing all API routes implemented.

API Specification		
Path	Method	Responses
buffer/	get	value
buffer/addToBuffer/username/amount	get	”ok”
buffer/getBuffer/username	get	value
buffer/getNetProduction/username	get	value
buffer/getFromBuffer/username/amount	get	”ok”
buffer/getRatio/number/username	get	value
buffer/setRatio/number/username/value	get	”ok”
producer/	get	”Production service”
producer/startUser/user	get	”ok”
producer/getUser/user	get	value
consumer/	get	”Consumption service”
consumer/startUser/user	get	”ok”
consumer/getUser/user	get	value
modelledPrice/	get	value
windSpeed/	get	value
price/	get	value
price/setPrice/price	get	
powerplant/	get	value
powerplant/status	get	string
powerplant/start	get	
powerplant/stop	get	
powerplant/getBuffer	get	value
powerplant/sellToMarket/amount	get	”ok”/”not ok”
powerplant/buyFromMarket/amount	get	”empty”/”0”/”not ok”
powerplant/setRatio/value	get	”ok”

Table 1: API specification for all services

API Specification		
Path	Method	Responses
app/	get	
app/getWindspeed	get	value
app/getModelledPrice	get	value
app/getConsumption	get	value
app/getProduction	get	value
app/getNetProduction	get	value
app/getNetProduction/username	get	value
app/getBuffer	get	value
app/getBufferManager	get	value
app/getUsers	get	value
app/getRatio	get	value
app/getRatio/number	get	value
app/getMarketDemand	get	value
app/getPowerplant	get	value
app/getPrice	get	value
app/getStatus	get	value
app/checkUpdateCredentials	post	
app/login	post	
app/redirectregister	post	
app/redirectlogin	post	
app/logout	get	
app/login	get	
app/register	get	
app/updateCredentials	post	
app/admin	get	
app/delete	post	
app/block	post	
app/register	post	
app/imageupload	post	
app/getImg	get	path
app/sendToBuffer	post	
app/sendRatiomanager	post	
app/useFromBuffer	post	
app/setPrice	post	
app/switch	post	

Table 2: API specifications för app service

2 Authentication

In order to authenticate users accessing the website they have to make an user account before given access to the simulated data. When signing up the user is presented with a register page (figure 2) where the user has to fill in their desired username and password. The username has to be unique and the password will be hashed with the help of a secret key and stored in the database. In addition to the hashing a random "salt" is added to password before the hashing takes place, this makes it so that identical passwords gets unique hashes. This is great because if identical password would have the same hashes and an attacker would compromise all the hashed passwords it would be easy to identify the hashes of the most common used passwords, but with salting this case is eliminated. Then the user can login with their username and password via the login page (figure 3). If the username exists in the database the stored password will be checked against the given password and if they match the user will be given access to the website and a "session" variable will be stored on the server with the information about the user and that the user has been authenticated.

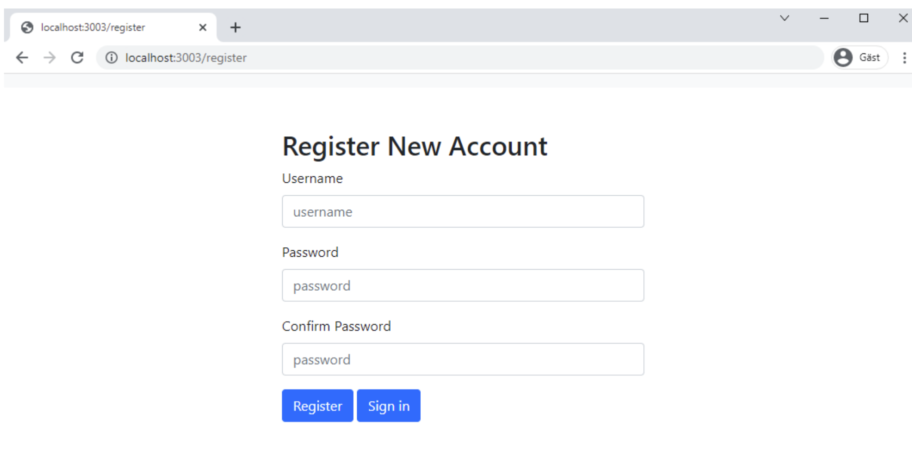


Figure 2: Register Page

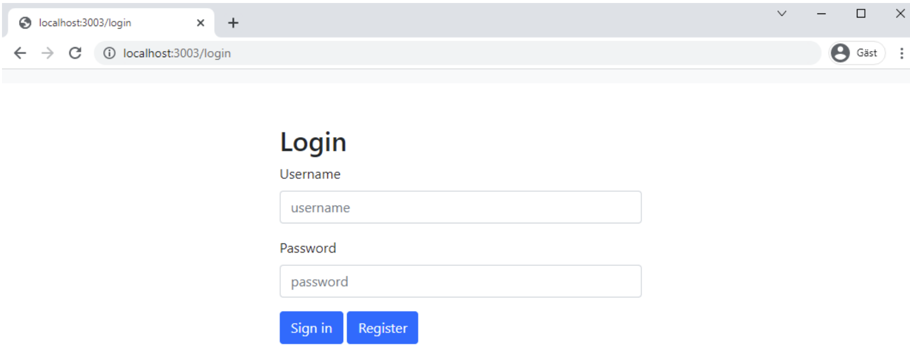


Figure 3: Login Page

3 Data

The simulated data is generated through several services which are presented in the System Specification. All data is stored in the same MongoDB database and uses two different collections for the market and the users. The Powerplant service is responsible for creating the market where users can buy their energy from. The admin can control the electricity production in the powerplant service through the API and also control the status and the ratio of what should be sent to the market and the powerplant buffer.

Market Collection	
Field	Type
Market	Double
status	String
ratio	Double
buffer	Double
MarketDemand	Double

Table 3: Market Collection in MongoDB.

A user can both produce energy and consume it. Therefore a user is connected to both the producer service and the consumer service. These services creates simulated production and consumption values for the users and stores these values in the database. Each user is also connected to the buffer service where the user is able to set a ratio of how much of the produced energy that should go the users buffer or be sold to the market.

User Collection	
Field	Type
username	String
password	String
buffer	Double
ratio1	Double
ratio2	Double
blocked	Boolean
role	String
consumption	Double
production	Double
blackedOut	Boolean
market	Double

Table 4: User Collection in MongoDB.

The values from the database is then loaded into the different pages on the website as seen in figure 4 and 5.

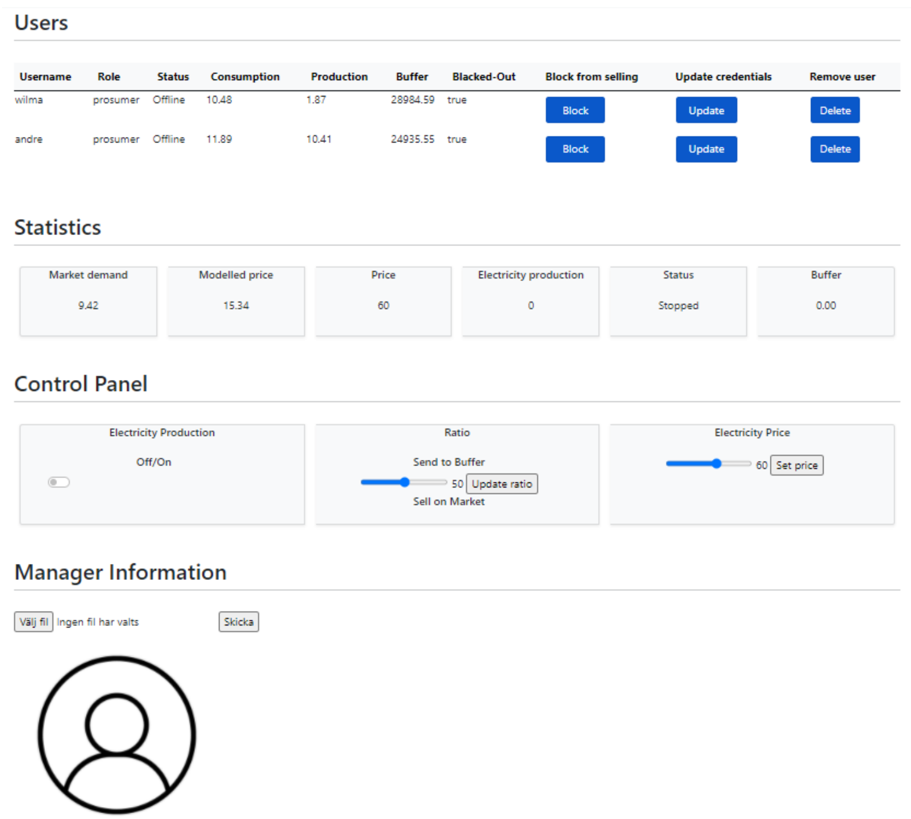


Figure 4: Admin Page

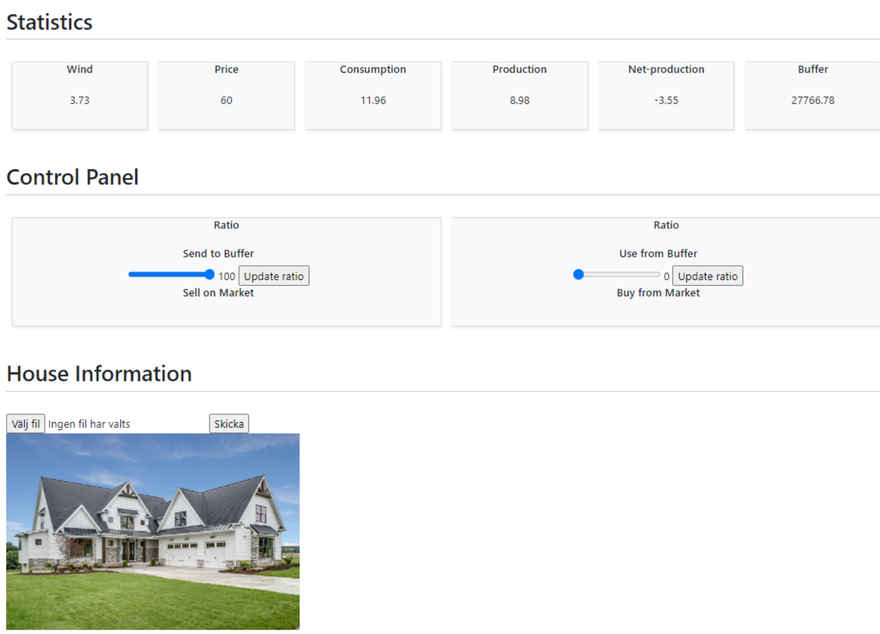


Figure 5: User Page

4 Documentation

4.1 Instructions to build and run application

The link to the github respository is: github.com/wilkru-7/M7011E. The instructions for how to setup the applications are given below.

4.1.1 Steps to build and run the project

1. Switch to branch named "docker"

2. Build all the containers by running the following command in the respective folder:
   docker build -t buffer .
   docker build -t consumption .
   docker build -t modelledprice .
   docker build -t price .
   docker build -t windspeed .
   docker build -t powerplant .
   docker build -t producer .
   docker build -t hemsida .

3. Deploy the application by running the following command in the deployment folder:
   docker-compose up

4.1.2 Steps to update services

1. Create the changes you want to deploy.

2. Rebuild the docker container by running for example docker build -t hemsida . (note that the name "hemsida "should be replaced with the name of the service, see step 1 in the build instructions for all the names)

3. Update the container that is used by docker-compose. This can be done either by stopping the service in docker hub and then starting it again (the container that was build the latest will be used when starting it again). The alternative is to stop the deployment and then restart it with the command docker-compose up in the temp folder

4.1.3 Extra notes

The docker-compose deployment is currently exposing the services through the same ports as defined in the express apps. If one would like the make a lot of changes to a specific service it might be better to not use that service in the docker-compose file (comment out the field concerning the service) and instead start the service with npm start.

4.2 Timelog

Below is a table over the documented time put on this course.

Timelog		
Week	André (h)	Wilma (h)
1	6.5	6.5
2	4	2
3	3.5	5.5
4	18	18
5	22	20.5
6	20	17
7	19	20
8	10	10
9	3	2
10	22.5	22.5
11	23	23
Total	151,5 h	147 h

Table 5: Documented time put on this course

4.3 Tests

In order to test that the simulated values are correct have two test files been created. The first test tests the wind speed values and works by generating 100 values and then checks that the mean value is within the gaussian distribution that has been defined. The test also checks that two consecutive generated values are not the same.

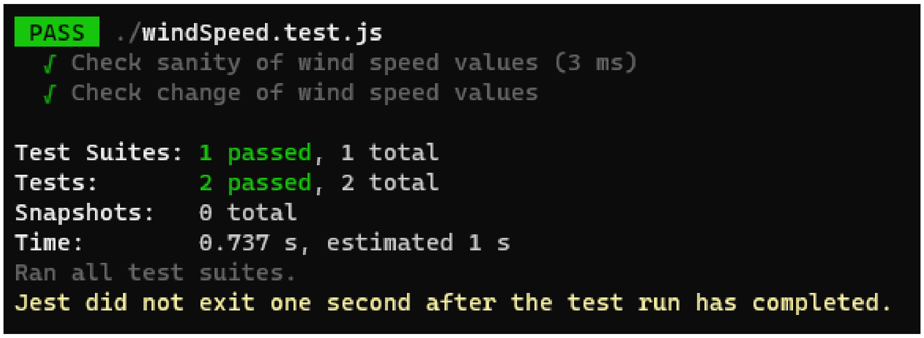
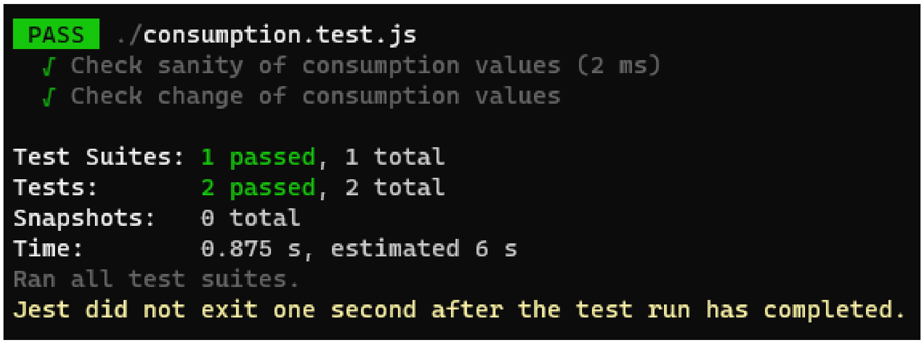


Figure 6: Wind speed test

The test for the consumption values works the same as the one for wind speed. First 100 values are generated and then checked that the mean value is within the gaussian distribution and also that two consecutive values are not the same.



```
PASS ./consumption.test.js
  √ Check sanity of consumption values (2 ms)
  √ Check change of consumption values

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.875 s, estimated 6 s
Ran all test suites.
Jest did not exit one second after the test run has completed.
```

Figure 7: Consumption test

5 Deployment

The entry point for the VPS is: 130.240.200.67:3003.
To login as admin use username: "admin" and password "hej". To login as ordinary user please register a new user and then login with the credentials chosen.