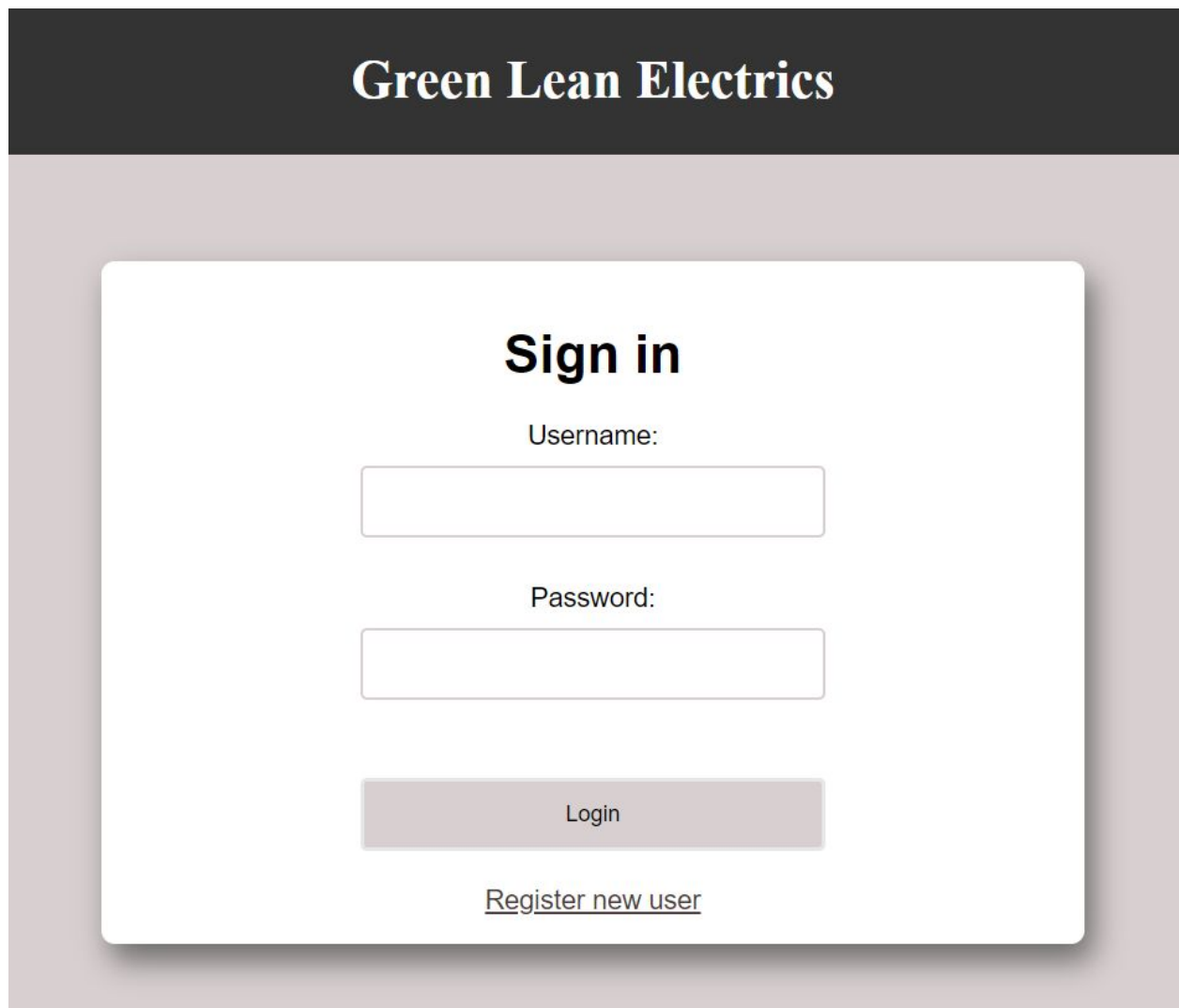


# REPORT

---



The image shows a web page mockup for 'Green Lean Electrics'. It features a dark grey header with the company name in a white serif font. Below the header is a light grey background. In the center is a white rounded rectangle containing the login form. The form has the title 'Sign in' in bold black font. It includes two text input fields labeled 'Username:' and 'Password:'. Below these is a grey 'Login' button. At the bottom of the form is a link that says 'Register new user'.

**Green Lean Electrics**

**Sign in**

Username:

Password:

Login

[Register new user](#)

Josefin Andersson-Sunna, josane-5@student.ltu.se

Filip Öberg, berfil-5@student.ltu.se

---

---

# Table of Content

<b>Table of Content</b>	<b>2</b>
<b>Introduction</b>	<b>3</b>
<b>Method</b>	<b>5</b>
Design choices	5
Front-end	5
Back-end	6
<b>Results</b>	<b>9</b>
Front-end	9
Prosumer page	9
Manager page	10
Sign in page	11
Register page	11
Back-end	11
<b>Architecture</b>	<b>12</b>
Advanced features	14
<b>Discussion</b>	<b>15</b>
Scalability analysis	15
Security analysis	15
Challenges	16
Front-end	16
Back-end	16
Future work	17
Front-end	17
Back-end	17
<b>References</b>	<b>18</b>
<b>Appendix</b>	<b>19</b>
Time analysis	19
Josefin	19
Filip	19
Contribution	19
Grade	19
Github link	20

---

## Introduction

This is a software project created in the course M7011E - Design of Dynamic Web Systems at Luleå University of Technology in Luleå, between November 2019 and January 2020.

The goal of the project is to develop a dynamic web system that oversees and manages an electric company's electricity production and its customers in the grid, as well as a simulator that simulates real-world parameters such as electricity consumption, wind speed (for electricity production via wind turbines), supply and demand, electricity prices and more.

The system has two main actors. A "prosumer", which is the "customer" to the electricity company. A prosumer is a household which can consume energy from the grid, but can also produce its own energy (Through a wind turbine connected to the household), and either save the energy in a buffer, or sell it to the market, or both. In the case of overproduction, the prosumer can choose how much of the excess energy to sell to the market, or send to the household buffer. In the case of underproduction, the prosumer can choose how much to buy from the market, or take from the household buffer.

The other actor is the "manager", which controls a large part of the system, mainly the power plant and its production, but also the prosumers connected to the grid. The manager can start and stop and control the production of the power plant. The power plant also has a buffer, where a ratio of the produced energy is stored. A ratio controlled by the manager. The manager also controls the price of the electricity. The simulator provides a modeled price, based on parameters like market demand, but the manager can choose to set any price. The manager also has access to the prosumers system, and can block them from selling their own energy to the market.

If not enough energy is produced in the grid, households will start to experience blackouts. Blackouts are visible by the manager.

---

---

---

## Method

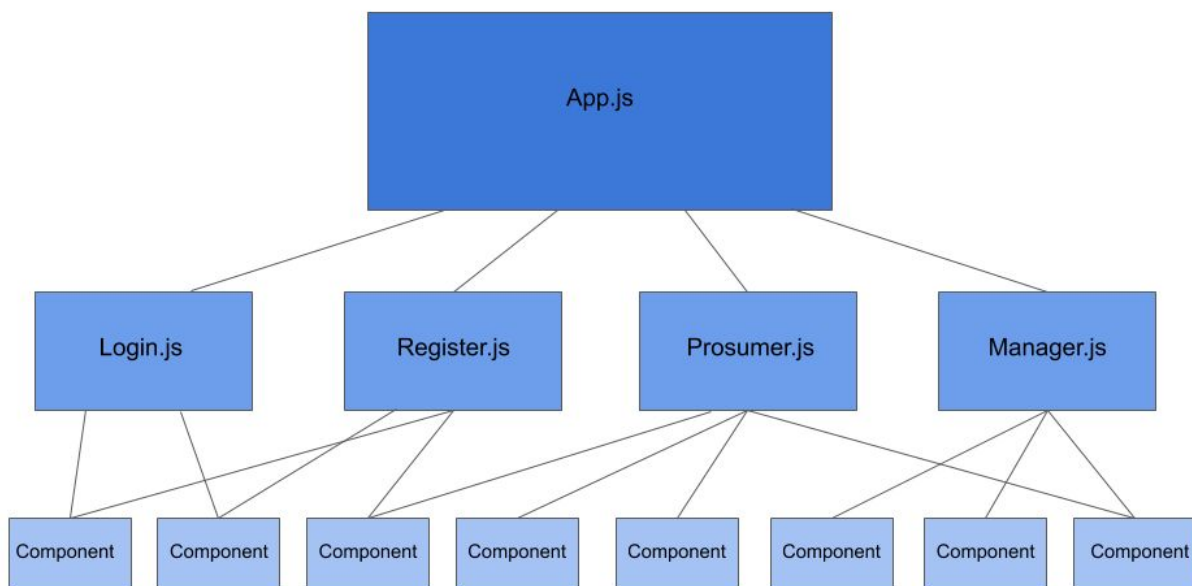
### Design choices

#### Front-end

The front-end is developed with React JS. React JS is a framework for making single page applications with components that can be reused in many parts of the application. The different components can hold states with values that can change dynamically without page reloads.

To make this project a single page application it has a parent component App.js which renders the consistent parts of the application together with the different page components depending on what page the user wants to visit, this is done with react router.

There are four page components, Login, Register, Prosumer and manager and these consists of many smaller components which can be used by several pages. The smaller components can also be used in each other. This is illustrated in figure 1.



*Figure 1: Component structure.*

---

The main benefit with this structure of page components consisting of smaller components is the reusability. Some pages have the same functionality and the smaller components makes it easy to add new functionality to a page without rewriting the same code. It is also easier to find where to make changes in the code if it consists of many smaller components instead of larger ones. If you want to add new functionality to a page you can simply write a new component and add it to the page component. A drawback with using the same components in several pages is that if you want to change or add something to it you have to make it work and look good for every page that are using it.

## Back-end

The back-end is built in NodeJS, which was a requirement in the course project. We also use Express as a framework, since we found it easy to use and is the most common web framework when it comes to Node.

The database we use is MongoDB, since we found that it's a very user friendly database that is easy to set up and integrates very well with Node. To make interactions with the database easier and object-oriented we also use the ODM Mongoose, which has built in validations and schemas that does a lot of the work for us when it comes to interacting with our MongoDB database.

The API of the server is structured into three main parts: **Routes**, **Controllers** and **Models**.

**Routes** is mostly self explanatory, it routes the requests coming in from the client to the right **controller**. In the routes step we also verify that the user making the request is allowed to access the data from that endpoint. For example, only a manager should be able to access the API endpoints that control the power plant. If a user is unauthorized, the request stops here. These checks are done through Express middlewares.

It's in the controller where most of the work is being done. It's here that we actually handle the request and send back a response. Depending on the type of request, we can simply fetch and send back data, create, update or delete a document in the database. If something goes wrong here, we send back an error status and a message back to the client.

---

All the access to our database goes through our Mongoose **Models**. A model is essentially an object representation of a collection in our MongoDB database. So if we would like to update some user data, we do that through the methods of a **User** object.

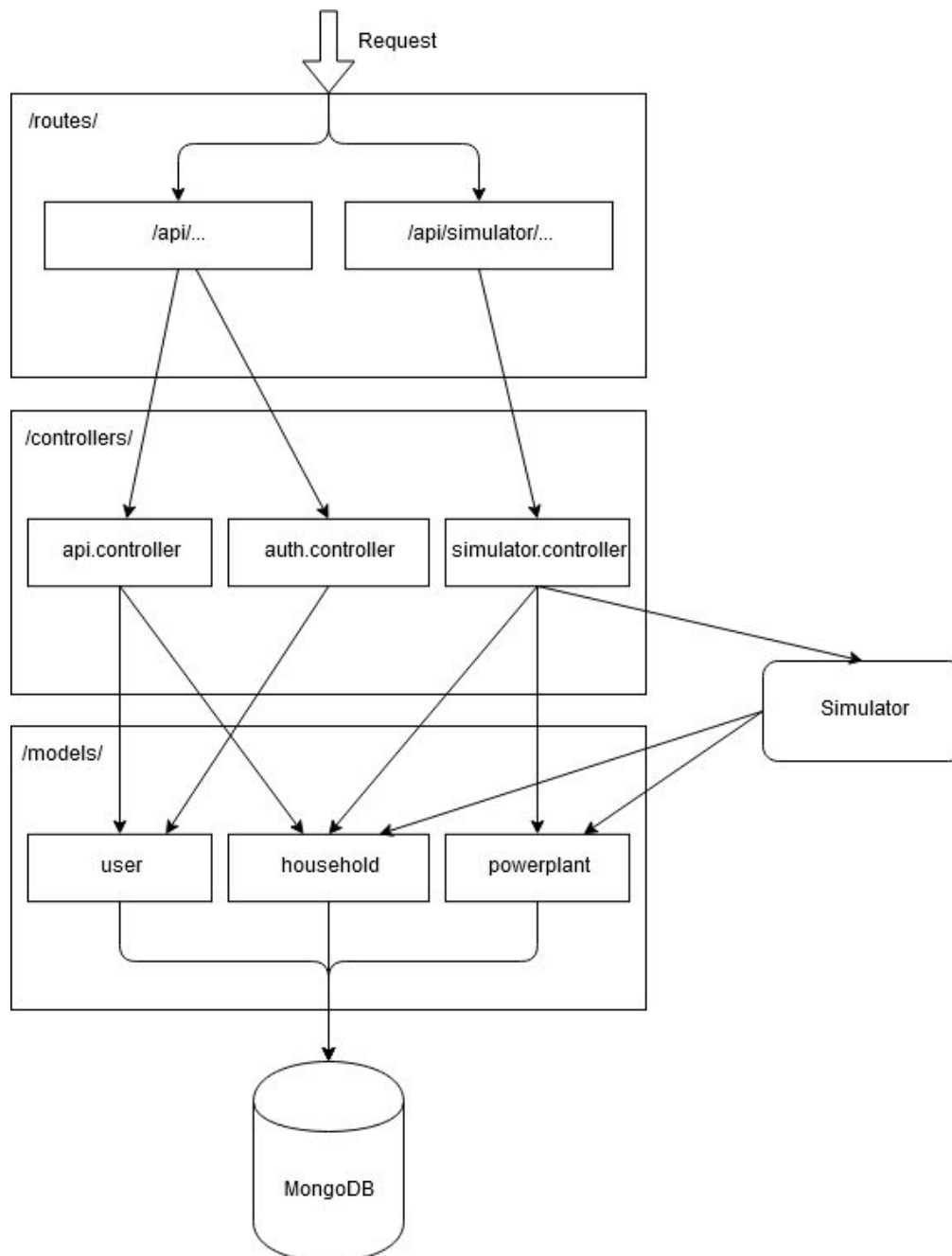


Figure 2: Back-end structure

---

This structure gives each component a clear purpose and makes it easy to understand the path the requests are taking, and where the work is being done. This also makes it easier to add functionality, since it's clear where you should add the new code. For example, if it's simulator related, you know the code is located in the simulator controller.

In addition to the API, we also run a simulator in the back-end. This simulator updates its state once every 10 seconds (By default, is configurable through a config file), and uses gaussian distributions to simulate fluctuations in wind speed during the day, as well as the consumption of each household.



---

# Results

## Front-end

The front-end is made up of four pages, a login page, a register page and a home page for each of the roles (prosumer and manager).

### Prosumer page

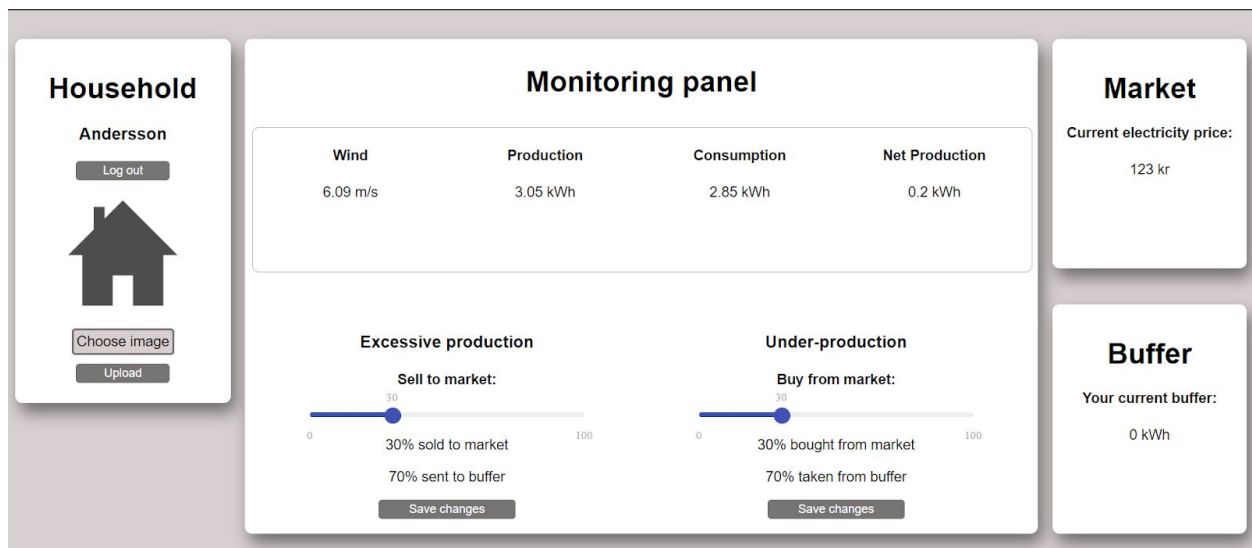


Figure 3: Prosumer page.

This is the homepage for a prosumer. Here, all the information about the prosumers household system can be found. The current wind speed, the production and the consumption of the household, the two ratios that control what to do with the energy in the case of under-/overproduction, the amount of energy stored in the buffer and lastly, the electricity price set by the manager.

The prosumer can also upload a picture of their household here.

## Manager page

The screenshot displays the Manager page interface. At the top, there is a navigation bar with links for Home, Prosumers, and Log out. The main content area is divided into two columns. The left column contains the 'Coal Power plant' section, which shows the status as 'Running' with a green dot, a 'Stop production' button, and production metrics: Producing (1000 kWh), Market demand (3.22 kWh), and Buffer (516.48 kWh). Below this is the 'Electricity price' section, showing a modelled price of 6.63 kr and a current price of 123 kr, with a 'Set market price' input field currently set to 0. The right column features the 'Prosumers' section, which indicates 'No black-outs' and displays a table of active prosumers.

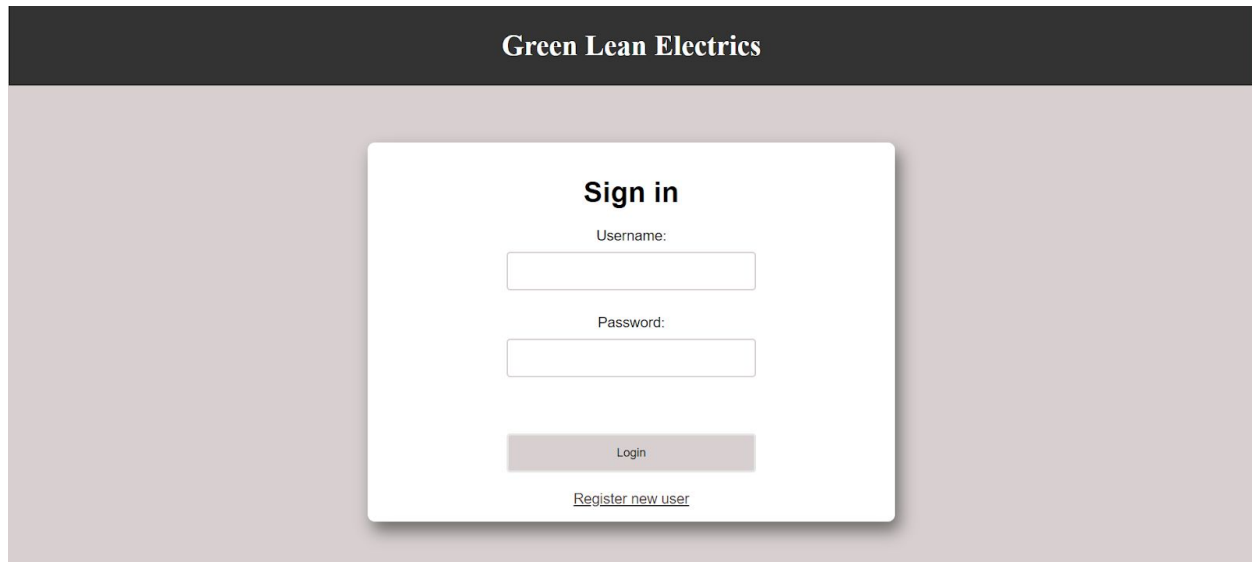
Username	Lastname	Status	Profile	System
jossan	andersson	online ●	Profile	View
anders	Andersson	offline ●	Profile	View
johan	johansson	offline ●	Profile	View

Figure 4: Manager page.

This is the home page for the manager. This is where the manager can control the coal power plant and the electricity price. The manager can also see all the prosumers, their profile, their system and their status. If a blackout occurs, the manager can see which prosumers that are affected. He can also update the prosumers credentials and delete their accounts. The manager has a profile page where an image can be uploaded and shown on the page and the manager can also update his own credentials and delete his account.

---

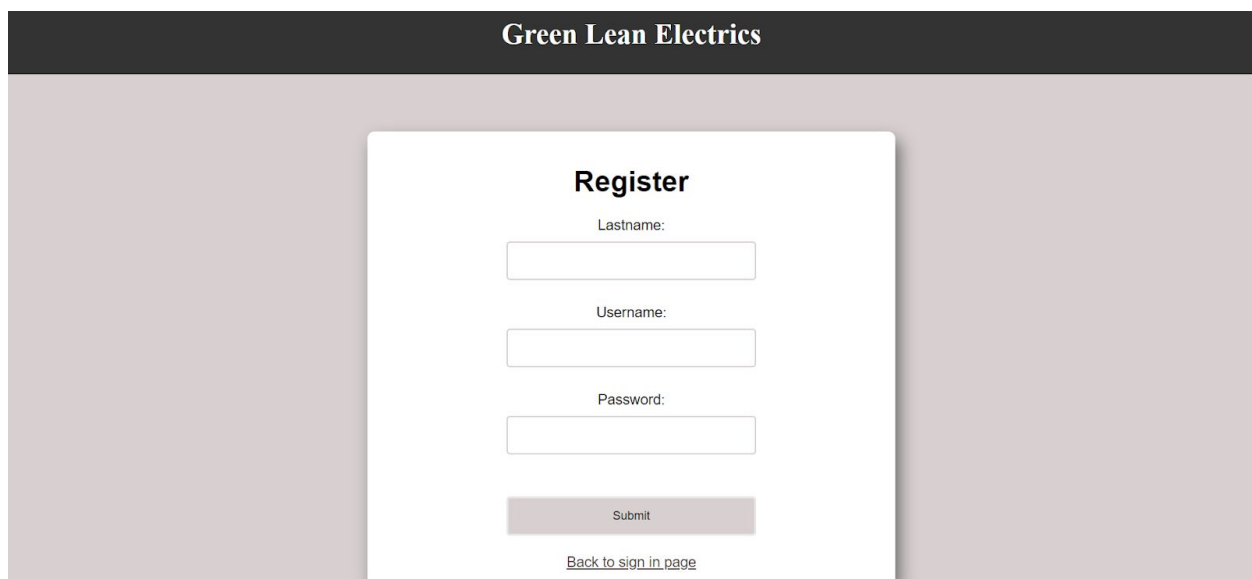
## Sign in page



The sign in page features a dark header with the text "Green Lean Electrics". Below the header is a light gray background. In the center is a white card with a shadow. The card has the title "Sign in" in bold. Below the title are two input fields: "Username:" and "Password:". Below the password field is a "Login" button. At the bottom of the card is a link that says "Register new user".

Figure 5: Sign in page.

## Register page



The register page features a dark header with the text "Green Lean Electrics". Below the header is a light gray background. In the center is a white card with a shadow. The card has the title "Register" in bold. Below the title are three input fields: "Lastname:", "Username:", and "Password:". Below the password field is a "Submit" button. At the bottom of the card is a link that says "Back to sign in page".

Figure 6: Register page.

## Back-end

The finished back-end contains an express server that runs a REST-API as well as a simulator. The REST-API fetches data both from the database and the running simulator.

---

The data is retrieved and saved on a MongoDB database. We use the ODM Mongoose to access the database in the Javascript code.

## Architecture

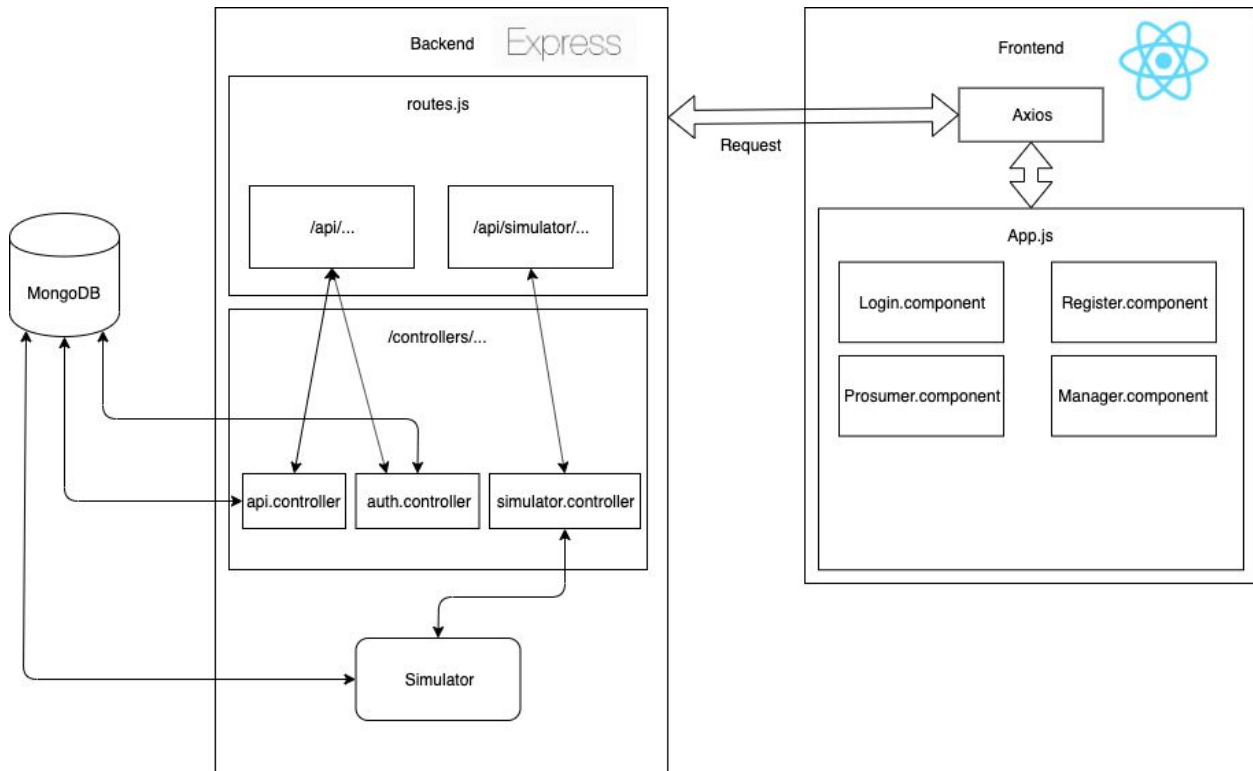


Figure 7: Overview of the system.

This graph shows an overview of how the system communicates between the front-end and the back-end. The front-end make requests using Axios. This request is then retrieved by the express server, which routes it to the correct controller. The `api.controller` handles requests not related to the simulator, i.e user data, profile pictures, etc. The `auth.controller` handles authentication when logging in and out of the system. And lastly, the

simulator.controller handles all the requests related to setting or retrieving data from the simulator.

Most of the data in the simulator are not saved to the database (wind speed, for example), so if you restart the server, these values will change. Some of the data are saved continuously though, for example the buffers of the households. This is because the buffers change every time the simulator updates, and the data needs to be consistent between server restarts.

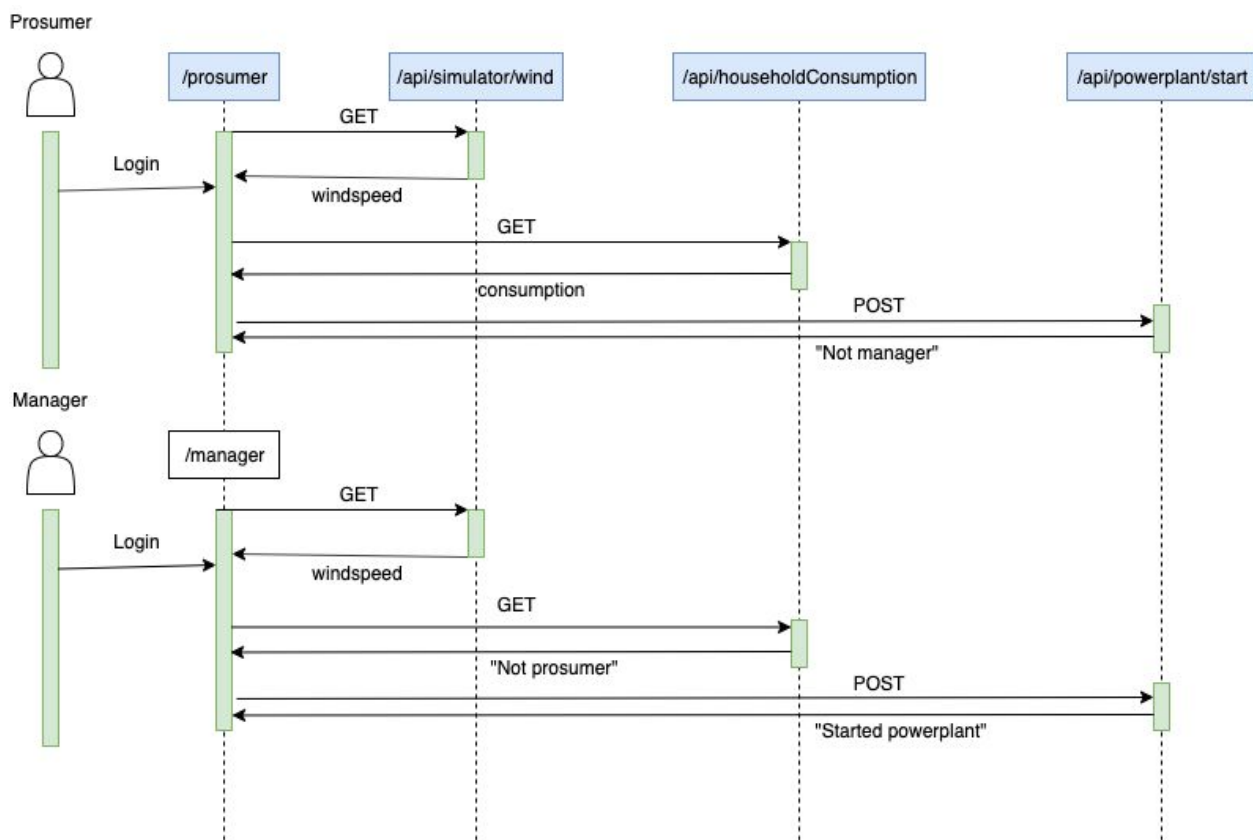


Figure 8: Sequence diagram of API requests.

This sequence diagram briefly explains how our API only allows users of the correct role to access or post data to our different endpoints. In this case, both a manager and prosumer can access the wind speed of the simulator. Only a prosumer can retrieve the consumption of the household (The manager has no household in this context). Only a manager can start the powerplant.

---

## Advanced features

The course assignment provided a number of advanced features to implement if a higher grade was to be obtained. These are the advanced features we feel we have implemented.

- There is a visual representation (e.g. Gauges) of the different values and graphical tools (e.g. sliders etc) for controlling the plant
  - We have sliders for controlling the plant and its ratios
- Should be able to use from multiple devices e.g. desktop and mobile (responsive)
  - All our pages in the frontend are responsive and can be used on mobile devices as well as desktops.

---

## Discussion

### Scalability analysis

Our API contains lots of small endpoints that send just one piece of data each. For example, one endpoint that sends back the household consumption, one endpoint that sends back the household buffer, one that sends back one of the household ratios, etc. These requests are made once every 10 seconds. This means that every logged in user queries the server a lot of times. Instead, we could group together these requests to one big endpoint, that sends back all of this data in an object instead. This would minimize the amount of times the server is queried. So if we merge 5 endpoints into 1, we can have 5 times the amount of users with the same amount of server queries.

Currently, we're pinging the API every 10 seconds to update our frontend. We have no way of knowing if something has changed beforehand. If we were to use websockets instead, the server could tell the client every time something has changed, and we wouldn't have to ask the server quite as much as before. For example, we now ask the API to send us the current electricity price every 10 seconds, but the price might not change for days or weeks. So if we would use websockets, the server could tell the client when the price changes instead, and only update it when it's required.

So currently, this system would not scale very well with the amount of users. The back-end server would probably be the bottleneck when it eventually won't be able to keep up with the large amount of queries that comes in every 10 seconds.

### Security analysis

Passwords are saved encrypted in the database.

We use server side sessions when logging in, and a cookie is saved on the client as a key to access the correct session on the server. The key stored in the cookie as a long random string and it would be close to impossible to guess someone else's key.

---

When a client tries to access a page, we ask the server if the client is logged in, and which role the client has logged in as (prosumer or manager). Based on the reply, the client will either get redirected from the page or get access to it. The same logic applies to when a client accesses the API, as seen in figure 7 above. If the user has the correct role for the endpoint, the client retrieves the data, otherwise the client is denied access.

The back-end uses Mongoose to access the database. The Mongoose models has built in validations that prevents incorrect types of data to be saved. For example, a string cannot be saved in a field which requires a number. You can also configure models with maximum and minimum values.

## **Challenges**

### **Front-end**

The biggest challenge for the front-end part of the project except from using a new framework (react) was to connect the front-end to the back-end. The manager component for example, has a lot of data and functionality that is dependent on requests to the server. The whole component was done before connecting any part to the back-end so when we connected the parts there was a lot of work to test the code and change it to make it all work as expected. It would have been easier to complete one part at a time.

### **Back-end**

The most challenging thing when developing the back-end was that we had no good and reliable way of testing that things worked the way it was supposed to. For example, if we rewrote a part of an API endpoint, we would have to manually test all combinations of ways that endpoint would be accessed or used. We mostly did this by using Postman to construct requests, but it's easy to miss something and it could take a long time to test everything. If we would've written tests for the API, we could've verified right away if something had broken during development, just by running the tests, saving lots of time.

Another challenging thing was to try and construct a relatively safe authentication system for the first time, and to understand how sessions in Express actually worked.



---

## **Future work**

### **Front-end**

Minimize the amount of requests by grouping them together and using websockets as described in the scalability analysis.

Build a better authentication system. When managing personal data, it is important to have a secure authentication system. As the system looks right now, a request is made to the server every time someone tries to reach a page to see if the person is logged in and if so what role it has, then it is sent to the correct page depending on the response from the server. This is a secure way because you can not reach a page if your not authenticated, but it is unnecessary to request the server on every page load. One way to resolve this could be to use private routing and authentication tokens, which would mean that only one request would be made to set these tokens when logging in depending on the role of the user. You can then remove these tokens when the user logs out or after the session time expires. The private routes would then only be available with an approved token.

There are always improvements that can be made to the GUI to make it cooler and more user friendly.

### **Back-end**

There's always a lot that can be improved. Websockets, as mentioned above in the front-end part as well as the scalability analysis, seems to be the most obvious thing that would improve performance. Grouping together API endpoints to fewer, larger ones, (also mentioned in the scalability analysis) would also be something that would improve performance. Also, writing tests for the API would probably be one of the most important improvements, just to verify that everything actually works the way we wanted to, and to prevent bugs appearing when adding more functionality to the system.

---

## References

- Traversy media, React JS Crash Course - 2019, [Video, github]. Available: <https://www.youtube.com/watch?v=sBws8MSXN7A&t=4873s> , [https://github.com/bradtraversy/react\\_crash\\_todo](https://github.com/bradtraversy/react_crash_todo)
- React training, React training / React Router, [web page]. Available: <https://reacttraining.com/react-router/web/guides/quick-start>
- React js, [web page]. Available: <https://reactjs.org/>
- w3schools.com, HTML, CSS, Javascript, [web page]. Available: <https://www.w3schools.com/>
- Corey Cleary, Project structure for an Express REST-API when there is no “standard way”, [web page]. Available: <https://www.coreycleary.me/project-structure-for-an-express-rest-api-when-there-is-no-standard-way/>
- Codementor, How to build a simple session-based authentication system with nodeJS from scratch, [web page], Available: <https://www.codementor.io/@mayowa.a/how-to-build-a-simple-session-based-authentication-system-with-nodejs-from-scratch-6vn67mcy3>

---

# Appendix

## Time analysis

### Josefin

At the beginning of the project I spent a lot of time reading links, setting up and installing everything needed to get started with the project. We made the base of the simulator together which made it quite quick to get ready.

Then I spent my time on the front-end, creating page by page and all its associated components which is what has taken up most of my time. Out of all the components, the manager took the most time to create because of all the features it contains.

### Filip

We started the project by building a good base and the simulator together. After that we split up the work between us. I've put in a pretty even amount of hours each day/week, mostly I've just been doing one small task at a time from our task backlog we kept in an excel sheet. Most of these tasks were small and didn't take very long on their own. Looking at the time sheet, nothing really stands out and my experience has been pretty smooth, not a lot of time has been spent being stuck on something.

## Contribution

Josefin has worked on the frontend and Filip has worked on the backend.

We've met at least once a week to synchronize our work and merge the front-end and back-end, as well as plan the work ahead with a backlog of tasks in excel.

## Grade

Josefin: 4

Filip: 4

---

## **Github link**

<https://github.com/Aundron/M7011E>