GROUP PROJECT

M7011E
Design of Dynamic Web Systems

LULEÅ TEKNISKA UNIVERSITET

David Wass
davwas-7@student.ltu.se
Philip Eriksson
phieri-5@student.ltu.se
Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

March 27, 2022

Contents

Contents

1 Introduction

Power production today is in the total control of the big power companies and countries that control the resources. With the technology today anyone with the resources can either build or contract a company to build a windmill on privately owned property. This gives the power control back to the people themselves, in order to coordinate this a system where private consumers can register and monitor their windmills is a necessity. This project is about the people or "prosumers" if you will, so they can control the power produced by their own windmills. If the prosumers produce excess power they can decide to sell it. All that's needed is a central API that anyone can download and run themselves. To get things started a simulator has developed by the creators of the API to give an idea of how it will look like when enough people have connected. Metrics such as wind speed, temperature and power consumption has been taken into account.

2 The project

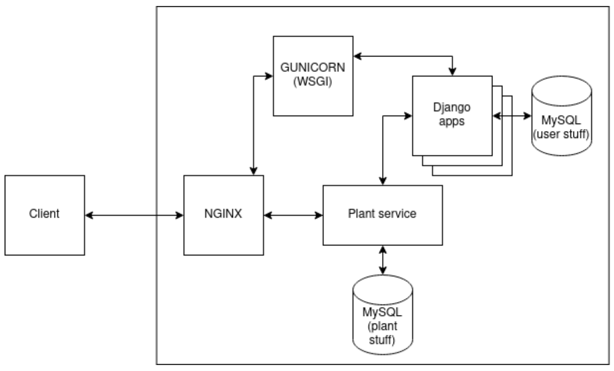


Figure 1: System design

For our project we decided on going for a micro service architecture - one service for dealing with plant information and running the engine of our events. The other service acts as an authenticate service and website for our users where they can graphically see their plants and decide on what they want to do with them.

2.1 Framework

Instead of dividing the framework into two parts front-end and back-end we chose to use a framework that work seamlessly between both worlds, Django. Using Django we can configure our desired database (MySQL in our case) and define the back-end using "views" and "models", then rendering them on the front-end in the browser using templates that are built on HTML files. Django simplifies a lot of things, to create tables in databases you simply write models and then make migrations that automatically adds them with all desired properties in the database. To access and view them views are created, all in python files. Then to render it in the browser templates are written in HTML files, to route everything correctly Django utilizes URL files. Signals can be used to create chain reaction, the included user model in Django is used in the project for signing up to use the service. These users only have predetermined data such as name, username and email; but we also want to be able to have a profile picture and a bio description. Using signals when a user is created a custom profile model is automatically created and linked to the newly created user. To make the service easier on the eyes, already created CSS style sheets are imported using a base file that the rest of the HTML files inherits.

User profile Every user has a profile associated with it. The profile consist of additional info (other than that of personal information upon creation) like a biography and an avatar. The profile can be viewed by the authorized users.

Admin dashboard As an admin you have extra permissions, you can view users, their profiles and make changes to them. In the navigation bar there are sections to view all power plants and their status, list all users and their login status.

2.2 Plant service

The asynchronous plant service is written with the python library asyncio. It parses incoming requests and determines if the request is authorized/supported. The service also runs a separate engine/simulator which is responsible for triggering random blackouts on the producing plants and updating the prosumers power storage. The database client is responsible for keeping track of incoming queries to the database and to process them accordingly (check for sql injections). By having a client for the database we avoid overloading the database by separate connections. To gain access to the plant server one has to first get a ticket by authenticating themselves with the authentication service. This ticket is time limited so if the two services are hosted on different sites then if the authentication service goes down the client still has a time limited ticket to manage their producing/consuming plants through the api.

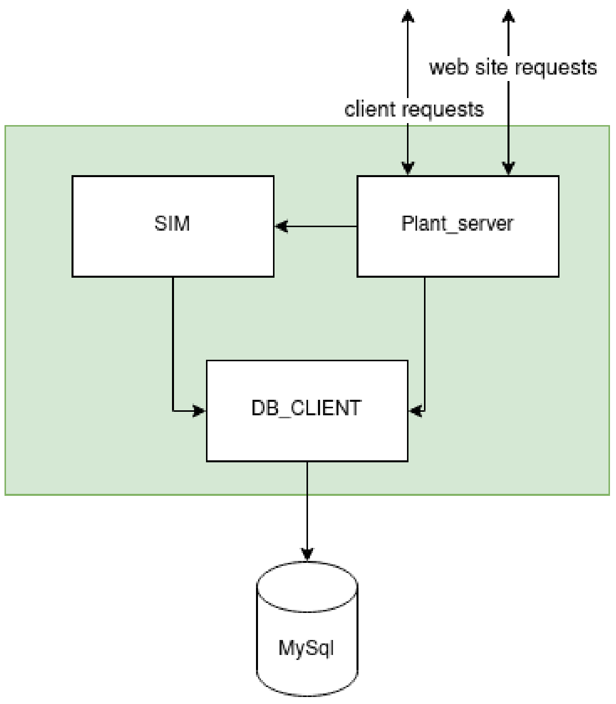


Figure 2: Module diagram over the plant service. Responds to authenticated clients

By having the client authenticate through the authentication service and receiving confirmation code from the authentication service that the client is trusted with token to access the plants we successfully allow connections to the customers power plants if the website were to ever go offline (limited time on ticket).

It also reduces the load on the main website as users who already have authenticated themselves now do not have to go through the authentication service after a log in has taken place.

2.3 REST API

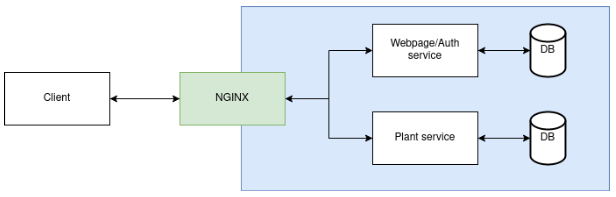
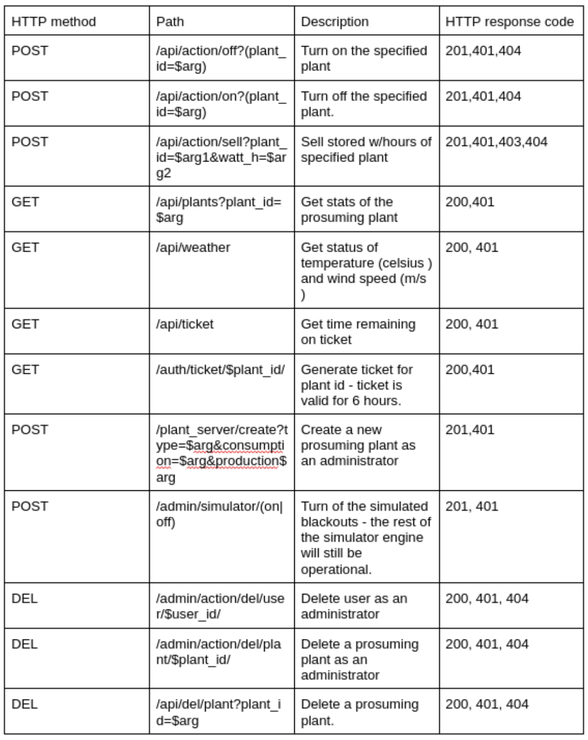


Figure 3: REST API Structure

As mentioned before, the client has the opportunity to manage their plants via the website which is hosted on the authentication server, or via the plant server if the authentication server would be offline (given they have a valid ticket)

2.4 API documentation



HTTP method	Path	Description	HTTP response code
POST	/api/action/off?(plant_id=$arg)	Turn on the specified plant	201,401,404
POST	/api/action/on?(plant_id=$arg)	Turn off the specified plant.	201,401,404
POST	/api/action/sell?plant_id=$arg1&watt_h=$arg2	Sell stored w/hours of specified plant	201,401,403,404
GET	/api/plants?plant_id=$arg	Get stats of the prosuming plant	200,401
GET	/api/weather	Get status of temperature (celsius) and wind speed (m/s)	200, 401
GET	/api/ticket	Get time remaining on ticket	200, 401
GET	/auth/ticket/$plant_id/	Generate ticket for plant id - ticket is valid for 6 hours.	200,401
POST	/plant_server/create?type=$arg&consumption=$arg&production$arg	Create a new prosuming plant as an administrator	201,401
POST	/admin/simulator/(on|off)	Turn of the simulated blackouts - the rest of the simulator engine will still be operational.	201, 401
DEL	/admin/action/del/user/$user_id/	Delete user as an administrator	200, 401, 404
DEL	/admin/action/del/plant/$plant_id/	Delete a prosuming plant as an administrator	200, 401, 404
DEL	/api/del/plant?plant_id=$arg	Delete a prosuming plant.	200, 401, 404

Figure 4: API documentation. Contains endpoints for both servers.

All the API calls to the plant service will return a 400 status code if formated incorrectly. POST calls will return a description body of the requested endpoint.

3 Work Distribution

As a group of two that previously had worked together we held weekly meetings determining what should be done for each week. Some times we sat together in voice-chat for the day working on on our own thing but helping each other out, other days we sat alone and synced up at the end of the day. Mainly three major areas was worked at; the simulator, the server and the front-end rendering.

3.1 Self-assessment and grading

David Wass:
I've mainly worked with the Django framework, learning it from scratch and creating the application that lets you create a user, login and update your profile. This is what is used to show the work done by the simulator. I've put a lot of time into this project but in hindsight the time could have been better spent since a lot of it didn't "bear any fruit" to the project result. Adding new pages or "views" to the project was quite straight forward after doing it a couple of times, but changing the built in admin features that Django provides to suit our needs was rather complex and time consuming. I do not think I deserve more than a passing grade as of right now.
Philip Eriksson:
I've mainly focused on the plant service, I've created an asynchronous http server from scratch which is used to respond to the API calls. Setup NGINX and the WSGI gunicorn server. The simulator engine runs separately from the Django apps so if it's any network downtime it won't effect the simulator as long as the VM (hosted by LUDD) is up and running. I do think I deserve a passing grade as I have developed a functional micro service architecture basically from scratch with many different local services communicating asynchronously.