

UNIVERSITY OF GRONINGEN

BACHELOR PROJECT

---

**Thesis**

**Visualization of local energy communities**

---

*Authors:*

Klaas TILMAN *s3229807*

Peter ELGAR *s2945584*

*Supervisors:*

M. MEDEMA

prof. dr. A. LAZOVIK

July 11, 2019



rijksuniversiteit  
groningen

# Contents

<b>1</b>	<b>Abstract</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
<b>3</b>	<b>Background</b>	<b>4</b>
3.1	Energy independent community . . . . .	4
3.2	Energy sources . . . . .	5
3.2.1	Solar panels . . . . .	5
3.2.2	Wind turbines . . . . .	5
3.3	Energy storage . . . . .	5
<b>4</b>	<b>Data analysis</b>	<b>5</b>
4.1	Electricity usage . . . . .	5
4.1.1	In a year . . . . .	6
4.1.2	Amount of inhabitants . . . . .	6
4.2	Electricity generation . . . . .	7
4.2.1	Solar photo-voltaic systems . . . . .	7
4.2.2	Wind turbines . . . . .	8
<b>5</b>	<b>Solution</b>	<b>9</b>
5.1	Analysis . . . . .	9
5.2	Architecture . . . . .	9
5.3	Requirements . . . . .	10
<b>6</b>	<b>Use cases</b>	<b>10</b>
6.1	Configuration cases . . . . .	11
6.2	Time cases . . . . .	13
<b>7</b>	<b>Implementation Web Interface</b>	<b>13</b>
7.1	Design choices . . . . .	13
7.2	Project building process . . . . .	15
7.3	Documentation . . . . .	17
7.4	Functionality . . . . .	19
<b>8</b>	<b>Implementation Sketchup</b>	<b>22</b>
8.1	Functionality . . . . .	22
8.2	Design choices . . . . .	24
8.3	Details . . . . .	25
8.4	Project building process . . . . .	26
8.5	Scaling . . . . .	27
8.5.1	visual . . . . .	27
8.5.2	practical . . . . .	28
<b>9</b>	<b>Discussion</b>	<b>28</b>
9.1	Strengths and Weaknesses of the Web interface . . . . .	28
9.2	Strengths and Weaknesses of Sketchup . . . . .	29

9.3	Sketchup visualisation versus Web Interface visualisation . . . . .	29
9.4	Current solutions versus other potential solutions . . . . .	30
<b>10</b>	<b>Improvements and known problems</b>	<b>32</b>
10.1	General . . . . .	32
10.2	Web Interface . . . . .	32
10.3	Sketchup . . . . .	32
<b>11</b>	<b>Conclusion</b>	<b>33</b>

# 1 Abstract

*Purpose:* The thesis finds and implements a solution for displaying decentralised electricity usage and production. The situation at hand is an increasing amount of decentralised produced electricity. Currently this energy is stored and excesses are being sold back to the grid.

The problem is that a household would profit more if it could sell this electricity on its own terms or store it in some device. The thesis describes and elaborates this method of handling the flow of electricity and tries to find a good model for displaying this information.

Identified energy sources are solar panels and wind turbines. Devices that can store electricity are cars and electric batteries.

*Methods:* The model thus consists of the production of electricity, followed by trading electricity and lastly storing excesses in an external device. The thesis describes two implementations of this model: One visualising through a 2D interface and one visualising through a Sketchup plugin in 3D. The models can run simulations for an average day in December and July.

*Results:* The 2D interface displays information through a web interface. The interface uses the google maps API to place icons on a map and uses polylines to display trading of electricity. Some more information is presented in the form of text on a side-window.

The 3D Sketchup plugin models households, energy sources and energy devices as 3D objects. Through colour coding it is able to show how electricity exchange would be able to work. Finally, both the implementations can run simulations for two or eight houses.

*Conclusion:* Lastly the thesis justifies the use of these type of visualisations by explaining that the quality of displaying information is more important than the quantity. Moreover, the implementations are superior to other possible methods, such as live graphs and a static web page.

# 2 Introduction

A house that understands your exact energy wishes. An electric car that decides when the best time is to charge a car with respect to electricity. Trading electricity to your neighbour for an amount of money. These are all things that we may expect in the future of electricity networks.

In such a community, energy will be collected from renewable resources. Mainly generated through solar panels and possibly windmills, each household will thus generate a certain amount of electricity.

In case of overproduction (producing more electricity than is needed for the household) the household can either store the electricity, trade it to one of the other households, who is dealing with underproduction, or sell it back to the grid.

However, currently electricity that is produced beyond the needed amount is simply sold back to an external provider. This provider sells the electricity with profit to some other household. The problem

is that the producer of electricity does not really make profit in this situation. A better situation would be that a household could sell electricity on its own terms or store it in some storage device. The problem and goal of this thesis is finding the best way to display this information in a way that it is clear to a potential user.

### **3 Background**

In recent years we have seen a great shift towards households being able to produce and store their own electricity. A reason for this shift is that technologies such as solar panels, wind turbines and electric cars have become relatively cheap and can produce or store a significant amount of electricity. Along with the fact that people desire to be more independent from the grid, and like to be more green in order to combat global warming. Below we go into more detail.

#### **3.1 Energy independent community**

Energy independent communities involve groups of citizens, social entrepreneurs, public authorities and community organisations who participate in the energy transition, which is done by collectively producing, selling, distributing and investing in renewable energy [1].

Benefits of such communities are:

- Economic development
- Creation of new jobs
- Cheaper energy
- Self-sufficiency
- Community cohesion

In this project we will lay the focus on groups of citizens, in the form of households. A household consists of a number of occupants who consume energy throughout the day, which is defined to be the amount of energy the household needs. To fulfil this need, there needs to be an equal amount of energy available, which can either be obtained by renewable sources, received from other households or lastly acquired from the grid.

Each household has the following attributes:

- Consumption of electricity per time period
- Production of electricity per time period
- One or no solar photovoltaic system
- Connected to one or no wind turbine
- One or no electric car
- One or no battery

A group of households will form a neighbourhood, in which we will model electricity transitions between the different households.

## **3.2 Energy sources**

Energy collected from renewable resources is defined to be renewable energy. Examples of these type of naturally replenishing energy sources are sunlight, wind, rain, tides, waves, and geothermal heat [2]. In this project we lay the focus on electricity generation by sunlight and wind. The reason for only using solar panels and wind turbines is because they are the easiest to get hold of and easiest for an individual to install.

### **3.2.1 Solar panels**

Sunlight can be absorbed by photovoltaic solar panels as a source of energy to generate electricity [3]. Electricity generation from these systems occurs only during daylight hours and peaks around noon. An important factor is the seasonal variation, which effects the amount of generation. In this project we assume a 3kWp solar PV system offering between 550-1300W of power during noon peak, we will go more in depth into this data in section 4 [4].

### **3.2.2 Wind turbines**

Wind turbines are able to generate electricity, not dependent on the solar position. They are dependent on the wind resource. We assume a 2.5kWp offering approximately 400W of electricity generation on a typical day, which will be discussed in more depth in section 4 [4].

## **3.3 Energy storage**

We can capture electricity produced at one time for use at a later time. In this project we will use electric cars and batteries as a way to store electricity.

## **4 Data analysis**

We decided to look for real world data. Firstly, because we wanted the models to reflect real world situations. Secondly, to save time from having to estimate the exact energy consumption of several different types of houses.

The data that we acquired was from the British government's website called the Household Electricity Survey. The survey followed 250 households for a year, to look at the average energy consumption of British households. Additionally, it took into account different types of houses (detached, semi-detached, apartments etc.), the amount of people who were living there, and also in what stage of their life the inhabitants were in e.g. young professionals, family with children, retired. Lastly, it also contained many other aspects such as holidays/weekends and workdays [5].

### **4.1 Electricity usage**

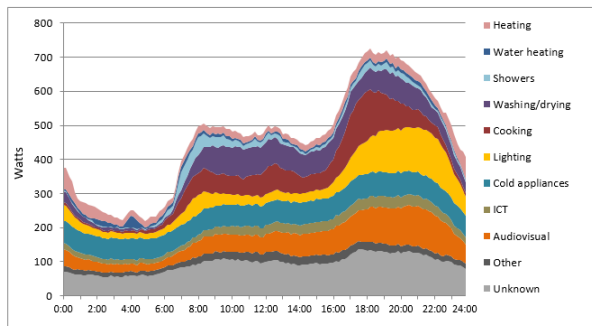
There are two main factors for the difference in electricity use. The first factor is the time of the year, because of the amount of sunlight meaning that more lights will be on during the day in winter as opposed to the summer. Furthermore, it is more likely to be bad weather during the winter, therefore indoor activities are more likely to be popular e.g. watching TV or playing board or video games. The second factor, is the amount of people living in a house. Because, this means that more people will be using electrical resources inside a house.

### 4.1.1 In a year

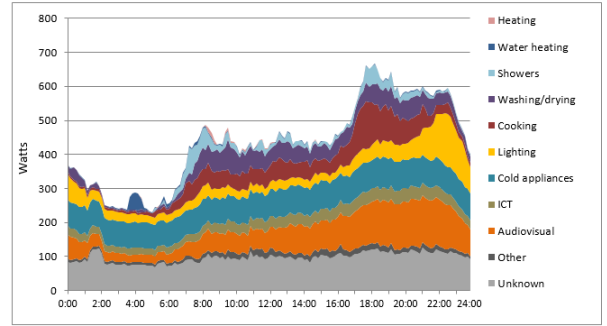
Looking at the figures below, which we obtained from the excel sheet, we can see that there is a significant difference in the amount of Watts of energy that gets used during the day. Furthermore, we can see a significant difference in the quantity between December and July.

For example, for an average day the peak is around 18:00 till 20:00 where 700 Watts gets used, whereas in July it hovers around 600 Watts. However, in December it starts an hour earlier at 17:00 and peaks at 1100 Watts. Therefore there is a significant difference in electricity usage during the year.

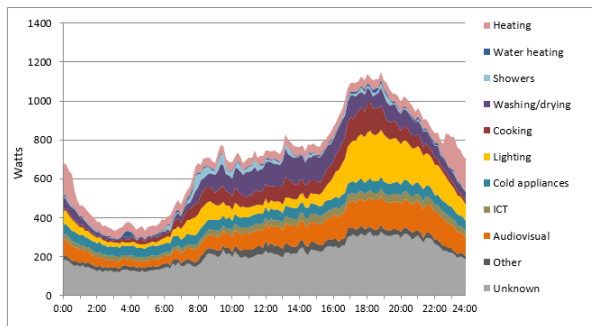
After further examination of the graphs, we can see that there are three stages of electricity usage. The first one being from 01:00 till around 07:00 where little energy gets used. The second phase from around 07:00 or 08:00 till around 17:00, where a medium amount of electricity gets used. Finally a peak from 17:00 or 18:00 till around 20:00 or 21:00, where after the energy usage trails off significantly.



Whole year 250 Households matched the conditions



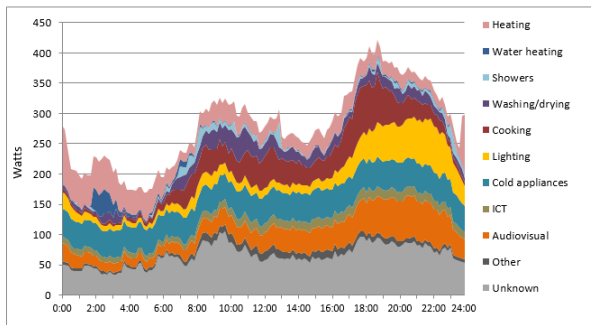
July 64 Households matched the conditions



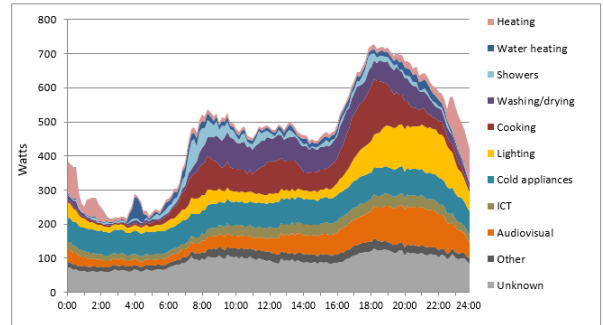
December 61 Households matched the conditions

### 4.1.2 Amount of inhabitants

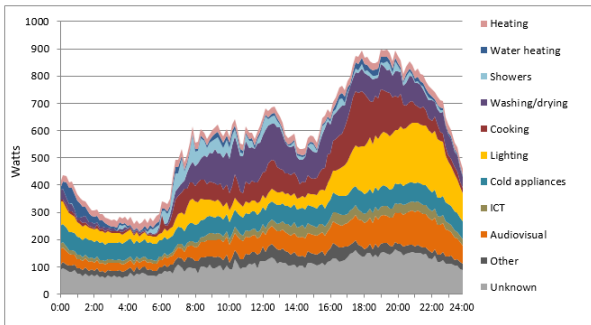
Looking at the data of different sized households. Firstly, we can see that there is a considerable difference in the amount of electricity used during the day between the households. A household with one inhabitant has a peak use of 400 when one with four has a peak use of 1000. Furthermore, there is a slight difference in structure of energy usage. Where a household of four people seems to have a more defined structure of 3 plateaus, when a household made up of one person seems more erratic.



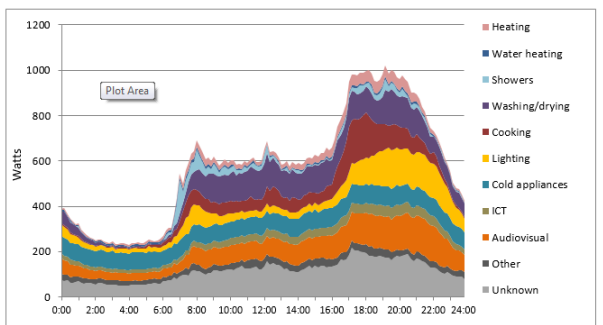
One person/Whole year 70 Households matched the conditions



Two people/Whole year 86 Households matched the conditions



Three people/Whole year 30 Households matched the conditions



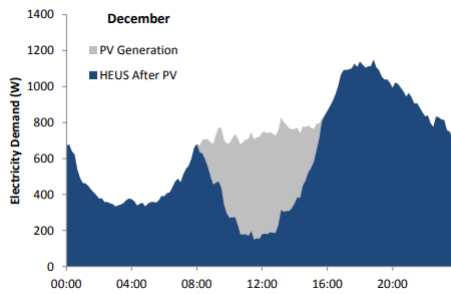
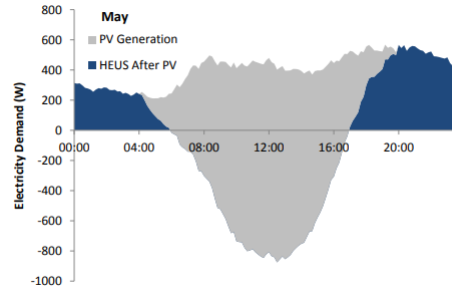
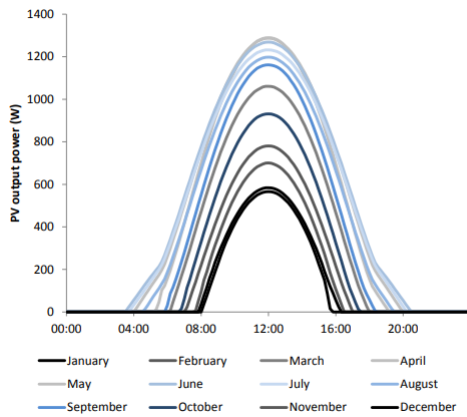
Four people/Whole year 49 Households matched the conditions

## 4.2 Electricity generation

We realise that the electricity generated by wind turbines and solar panels varies a great deal. Therefore, this can mean a great difference in day to day generation. Therefore we decided to use the average generation of these technologies for every month in the UK, using the data received from the report on low carbon technologies [4].

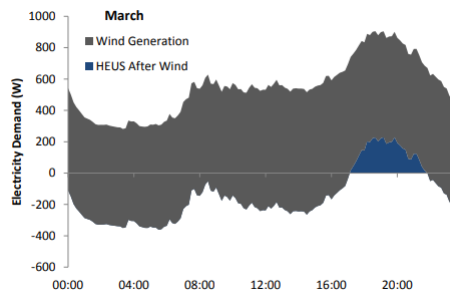
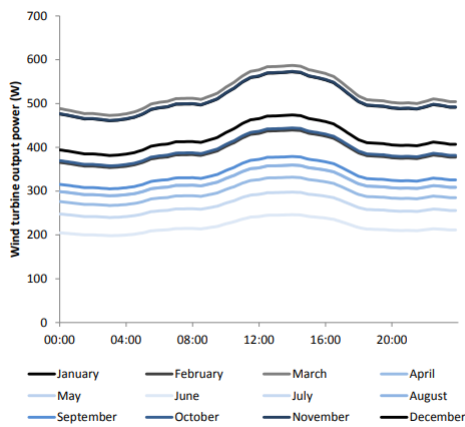
### 4.2.1 Solar photo-voltaic systems

As would be expected, and can be seen below, there is a big disparity between the electricity generated by a solar panel during the summer, and the winter. However, looking at the other two graphs, the energy generated does make a sizeable dent in the net energy consumption throughout the year. Albeit during the winter months it will not create a negative net electricity use. By comparison in May, a large part of the day has a large negative net energy generation.

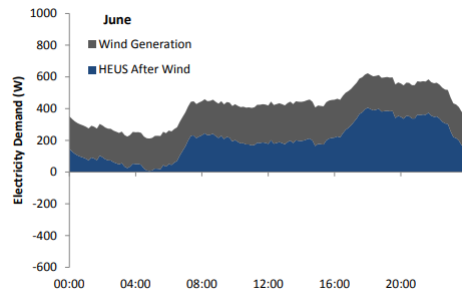


### 4.2.2 Wind turbines

In contrast with the solar panels we can see that there is a more regular electricity generation during the day. Secondly during the winter/spring months there is a greater amount of energy production. However during the summer months as we can see a wind turbine can still make a significant dent in the net usage of electricity for a house, throughout the day instead for a couple of hours. When in March there is largely a negative net usage of electricity. Albeit less than for solar at its peak.







## 5 Solution

Since we have gone through the data and given the reasons why this data is important to visualise. We need to lay down what needs to happen, and how this will be done. In order to determine how the minimal viable product should look like.

### 5.1 Analysis

So far we have given some background information regarding the project. Firstly, we have established that our project will consist of a neighbourhood containing a number of households. Secondly, each household can have a number of energy sources and energy storage options. Thirdly, each household produces and consumes a certain amount of electricity.

The goal of this project is coming up with a good method to present this information to a potential user. The main contrast in this presentation is between linear and nonlinear.

With presenting the data in a linear way, we mean simply displaying information in the form of numbers and text. This way the focus will lay on offering knowledge for as many households as possible.

Conversely, presenting the data in a nonlinear way would be through visualisation of information. This can either be done in a two-dimensional or three-dimensional way. Advantages of this method will be that the presentation can be made quite clear and intuitive. In the sense that a potential user can easily understand interactions between households. However it would be harder to present visualisations for a very high number of households.

For this project we have decided to present our information through visualisation. We favour an easily understandable and intuitive implementation over a linear presentation which might be able to show more households. The visualisation will be done in both a two-dimensional and a three-dimensional way. Each one of these will have its own section, where we will go into more depth about the implementation.

### 5.2 Architecture

Despite having to make two different implementations, both have a similar underlying architecture. Namely, being made out of three separate components; server, program and model. Firstly, the user should communicate with the program part, there he/she communicates what he/she wants to see. Subsequently, the program will request the data from the server, and the server will send the data to the program for every iteration in the simulation. After receiving the data the program will make the necessary calculations and will load the model or display the changes in the model.

All three components therefore have clear tasks. Firstly, the server is solely meant to hold the data, and send it upon request. Secondly, the program requests the data, interprets the data, interacts with the user, and makes changes to the model. Lastly, the model displays the information, however does not contain any underlying logic.

### 5.3 Requirements

The features are divided into two categories: **Important** and **Useful**. The important features can be seen as requirements the interface should have in order to form a minimum viable product, while the useful features are extra requirements which are not essential but would be nice to have.

Below we list the features corresponding to this division.

#### Important

- The implementation shall be able to show at least 2 households
- The implementation shall be able to show consumption for each household
- The implementation shall be able to show production for each household
- The implementation shall be able to show overproduction for each household
- The implementation shall be able to show how many cars and electric batteries each household has
- The implementation shall be able to show how many windmills and solar panels each household has
- The implementation shall be able to show cars, batteries, windmills and solar panels
- The implementation shall be able to show electricity flow from one object to another

#### Useful

- The implementation shall be able to show at least 8 households
- The implementation shall be able to show a cycle of 24 hours (one day) of the community
- The implementation shall be able to show for each appliance how much energy it consumes
- The implementation shall be able to show how much energy is produced by each windmill and solar panel for each household
- The implementation shall be able to show the current storage of a car and battery

## 6 Use cases

The data discussed in the previous section is presented in the form of use cases. We formed two use cases for our project: a community made out of two households and a community made out of eight households. The first community can be seen as a base case. The community consists of two households, which can be used to simulate very basic relations between the two. For example, if one

of the two households generates electricity while the other does not, it can be clearly shown how they trade electricity and make sure together they are independent of the grid. Obviously a more realistic community would contain more households, but a potential user could use this scenario to see how the use cases are visualised and what functionality is available.

Similarly the community of eight households can give more insight into the independence of an external supplier. This use case can however give some additional insight since there are more relations than can be shown, with respect to the trading of electricity between the various households.

Furthermore this use case can be used to display a somewhat more complex community since we can configure the households to all have different energy storage options and energy sources.

Below we will shortly display the configuration of the two neighbourhoods which we used throughout this project, this setup is however easily changeable by changing a few values in the corresponding files which we will discuss later.

	Solar panels	Wind turbine	Electric battery	Electric car	Number of people
House 1	Yes	No	No	No	2
House 2	No	No	No	No	2

Table 1: Configuration for 2 households community

	Solar panels	Wind turbine	Electric battery	Electric car	Number of people
House 1	No	Yes	Yes	No	2
House 2	No	No	No	No	2
House 3	No	No	No	No	4
House 4	No	No	Yes	Yes	3
House 5	No	No	No	Yes	4
House 6	Yes	No	No	Yes	4
House 7	Yes	No	No	No	2
House 8	Yes	No	Yes	No	1

Table 2: Configuration for 8 households community

## 6.1 Configuration cases

The first thing that is sent from the server to the client is the configuration case. The configuration cases were only used in the web interface, the Sketchup implementation does not support them. This case specifies the configuration of the community, it includes for each household:

- The name of the household
- The storage size in watt-hour of a battery, if applicable;
- The storage size in watt-hour of a car, if applicable
- The maximum production of solar panels, if applicable
- The maximum production of a wind turbine, if applicable
- The latitude coordinate of the household

- The longitude coordinate of the household

For both the storage size and maximum production it holds that if the amount specified is equal to zero, the household does not own a storage device or generator. Since the household is placed on a Google map, for the interface implementation, it is necessary to send the latitude and longitude coordinates so that the client knows where on the map the respective household should be placed. The configuration cases are send in the form of a JSON file with the following syntax:

```
[
  {
    "Household": name,
    "Battery": amount,
    "Car": amount,
    "SolarPanels": amount,
    "Wind Turbine": amount,
    "Latitude": latitude coordinate
    "Longitude": longitude coordinate
  },
  ..
]
```

For the batteries and cars we used a value of 50.000 watt-hour. For the solar panels the maximum production is 1300 watts and for the wind turbines 600 watts, these values were derived from the data discussed in section 4. All these values can however be easily changed to the users wishes. Below is shown the configuration file corresponding to the community of two households. The configuration file for the community of eight households was not included but simply uses the configuration listed in Table 2.

```
[
  {
    "Household": "Household 1",
    "Battery": 50000,
    "Car": 0,
    "SolarPanels": 1300,
    "Wind Turbine": 0,
    "Latitude": "53.233091",
    "Longitude": "6.535706"
  },
  {
    "Household": "Household 2",
    "Battery": 0,
    "Car": 50000,
    "SolarPanels": 0,
    "Wind Turbine": 0,
    "Latitude": "53.233125",
    "Longitude": "6.535889"
  }
]
```

## 6.2 Time cases

We decided to make two time cases, one for an average day in December, and another for an average day in July. The days start and end at midnight.

An iteration represents 10 minutes in the real world, this is because the data we received was done that way.

We decided on July and December as we wanted two polar opposites of each other. In terms of electrical production for solar panels and windmills. But also in terms of electrical usage being a lot higher in winter than summer. Furthermore, it shows a great disparity when not a lot of electricity is being produced during a day i.e. December, compared to a lot i.e. July.

The time cases are represented in the form of a CSV file. Every row in the file specifies one time period of 10 minutes, the total number of rows is thus 144 since the file presents a 24-hour period. Every row of the CSV file has the following syntax:

```
production , consumption , production , consumption , etc .
```

Every two columns thus specifies the production and consumption of a household for ten minutes. Every column is separated by a comma.

Below is listed the first 10 rows of the time case corresponding to the community of two households:

```
341 ,0 ,709 ,4  
347 ,5 ,0 ,764 ,6  
335 ,6 ,0 ,719 ,2  
323 ,5 ,0 ,666 ,4  
292 ,0 ,661  
285 ,9 ,0 ,648 ,2  
259 ,6 ,0 ,597 ,9  
271 ,3 ,0 ,558 ,3  
272 ,4 ,0 ,506 ,9  
236 ,6 ,0 ,510 ,1
```

## 7 Implementation Web Interface

This project consists of developing a web interface which will give more insight in an energy independent community. The goal is to develop an interface in the form of a map, which is able to show all needed interactions within and between households.

In order to visualise interactions, data needs to be sent from a server to the client in intervals. The client then needs to be able to process this data and appropriately show it.

Lastly the interface should be intuitive, in the sense that it is clear to the user what is happening at each moment.

### 7.1 Design choices

#### Map API

Firstly we had to decide which tool or framework we would use for displaying the interface in the form of a map. We found the following three options:

- OpenLayers: Free solution for displaying dynamic maps in a web page or application.[6]

- Advantages: Free, Tiled Maps, vector maps, markers
- Disadvantages: Not used as much as other APIs, so less support can be found on the web.
- TomTom: Displaying maps with a reputation of satellite navigation.[6]
  - Advantages: Display maps, search for locations, see traffic density, find best routes from A to B
  - Disadvantages: Paid from 2500 requests per day
- Google Maps API
  - Lots of support and functionality
  - Paid after 28.000 requests (for dynamic maps) [7]

The least interesting option was the TomTom API, since the API is meant more for projects concerning navigation and not so much for creating custom maps.

Therefore we had to decide between using OpenLayers and the Google Maps API, as described above the OpenLayers API is completely free which was a major advantage however the Google Maps API is supported much more and thus in that sense more interesting. However the Google Maps API is paid from 28.000 requests.

We decided to use the Google Maps API, since the number of requests would be more than enough to support our interface, furthermore the functionality and support of the API is much bigger than for OpenLayers.

## **Node.js**

The decision to use NodeJS was based on the factors that there are a massive amount of libraries available, lots of documentation to easily get started and quite easy to use.

## **Bootstrap**

For the frontend of the project we decided to use the library Bootstrap. We already had some experience with the library. Furthermore it can be easily used to make a website look much better with only a few lines of code.

## **Express**

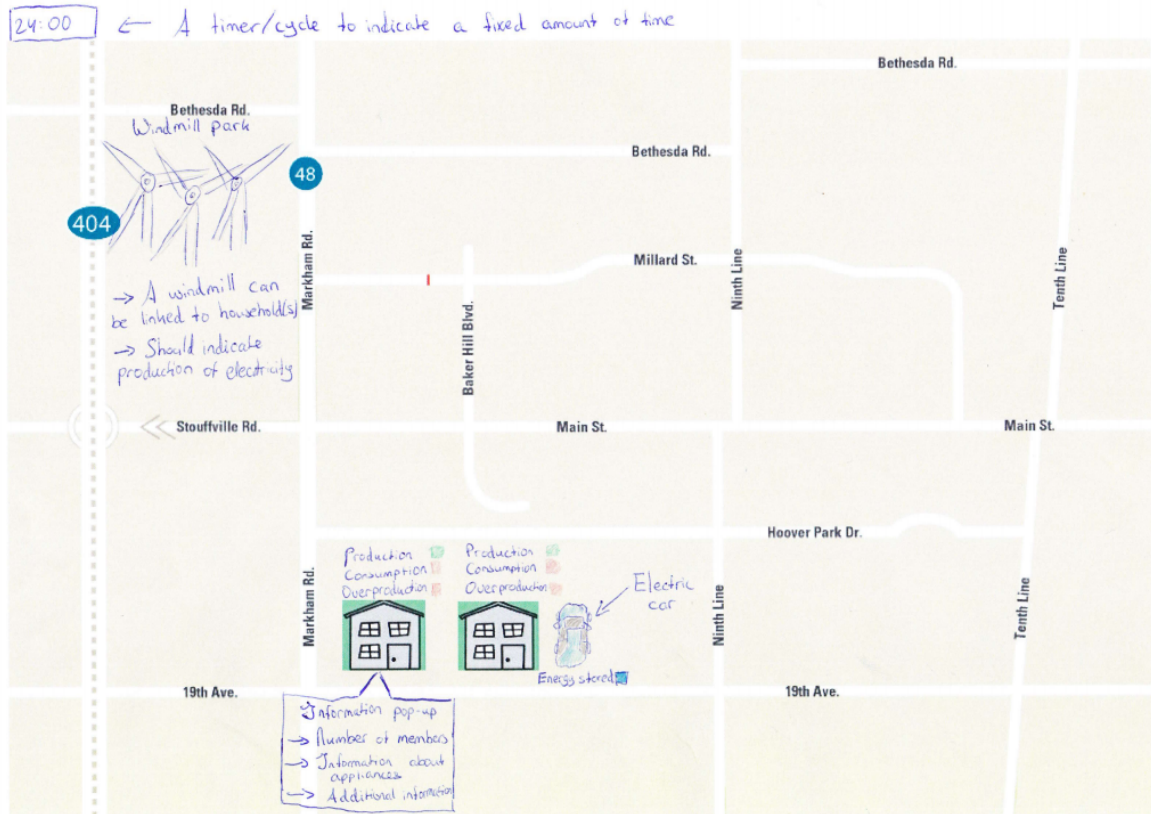
The Express framework was chosen because it is able to provide a simple frontend webpage to a NodeJS backend. This in combination with a large amount of documentation and tutorials made it a good choice.

## **Chart.js**

The interface needed a doughnut graph to display the ratio of production and consumption of electricity. We decided to use the JS library ChartJS for this purpose. There is quite a lot of documentation available and it was quite easy to add a graph to our map using the library.

## 7.2 Project building process

Based upon the requirements set before, we started the building process of the project. The first thing we did was make a basic sketch containing the important features of the interface. This sketch was created to get a rough idea of how the end-product should look like and to see if all features could be displayed clearly. Using this sketch we continued the process of building the interface, the sketch can be seen below.



As mentioned before we decided to use the Google Maps Api. We were looking for the following functionalities from said API:

1. Displaying houses, solar panels, wind turbines, cars and batteries in the form of icons.
2. Displaying progress bars and charts on the map solar panels, wind turbines, cars and batteries in the map.

The first functionality could be quite easily acquired through Markers [8]. A marker identifies a location on a map. By default, a marker uses a standard image, however markers can display custom images, in which case they are referred to as 'icons'. We could thus simply change the icon of the marker to a house, solar panel, wind turbine, car or battery and like this the marker could represent the object. An additional functionality of the marker is that it can show an info window when clicked. We used this to show some additional information of the object.

The second functionality is somewhat different from the previous one. The icons are simply images and don't contain any additional functionality. The progress bars and charts however contain completely different html code. We therefore were looking for something that would allow us to display our own html code on the map. For this purpose we used Custom Overlays [9]. We created a new

class extending from the class belonging to Custom Overlays, OverlayView. This new class allowed us to add markers to the map with custom html code.

Lastly the functionality for displaying relations was done using polylines [10]. Polylines can be used to show the path between two points on a map. So we simply add the coordinates of two objects and are so able to draw a path from for example a house to a solar panel.

The next step was creating a skeleton of classes. For each significant object in the interface we created a class, so that it would be simple to add new objects to the interface. The classes we created will be discussed in more detail in section 7.3.

Since we now had some basic code with classes, we started working on our interface. Because all the objects we place are determined by coordinates, we first decided which area the interface will show and where the different households would be located. We then figured out how to place the different objects on the map and first tried to place two markers, corresponding to two households. When we figured out how to do that, it was quit simple to place more markers on the map for the different objects.

The next step was visualising the data coming in. The most important data to be visualised was the consumption and production of electricity. Furthermore since everything in the interface is supposed to be visualised around the households, we decided to literally visualise this aspect around the households. We added a doughnut chart which indicates the ratio of production and consumption for the relative household.

Additionally we needed to visualise how much electricity was being stored in both of the storage devices. In order to make this as clear as possible we used a progress bar. The amount of which the bar is filled, with a blue colour, directly corresponds to the amount of which the device is charged at that moment. The progress thus increases when the device is charged and decreases when discharged. When the bar is filled with the colour blue, the device is fully or close to charged.

Similarly we wanted to indicate how efficient the generators were at each single moment. We also achieved this by adding a progress bar next to the icons. The bar indicates what percentage, of the maximum production possible, is being produced. Therefore, if the bar is completely filled, this means that there is maximum production.

As described before, the relations between the different objects are being displayed using polylines. If we want to draw a polyline between two objects, for example to visualise the electricity flowing from a solar panel to a household, we take the coordinates of both the objects on the map and calculate the coordinates of the path in intervals. This makes sure that the polyline is being animated and slowly moves to its destination. The same is done for the charging/discharging of an storage device and for the trading of electricity between households.

Following finishing most of the visualisation, we decided to display some more information. This would give a potential user some more information about the interface. We chose to place this on the side of the interface and is simply done by adding some html which includes for each household information about its objects. This side-window takes 20% of the screen while the interface uses 80%, this seemed like a good division to us.

Up to now we used some constant values to test the visualisation part of our model. But, as described before, we have some real data available that needs to be used. So we started looking for the best way to send this data to our interface. Communication only needs to happen from the server to the client, i.e. the client wants to receive a constant 'stream' of new data and only needs to specify once



which stream it wants to receive. We found that the best option for this was using Server-Sent-Events.

Server-Sent Events (SSE) is a server push technology enabling a browser to receive automatic updates from a server via HTTP connection [11]. This is exactly what we needed so we decided to implement this in our project. The server logic is written in *server.js*, while the client receives the streams in *main.js*.

After the server was running and working, we adapted the code in such a way so that it would be possible to send both configuration and time cases, as discussed in section 6.1. The client first receives the configuration files and initialises all the objects on the screen. Then the client receives the time cases and converts them to an array to make them more usable.

At this point in the project, the underlying logic was close to done as well as the visualisation. We decided to perfect the project by adding some features.

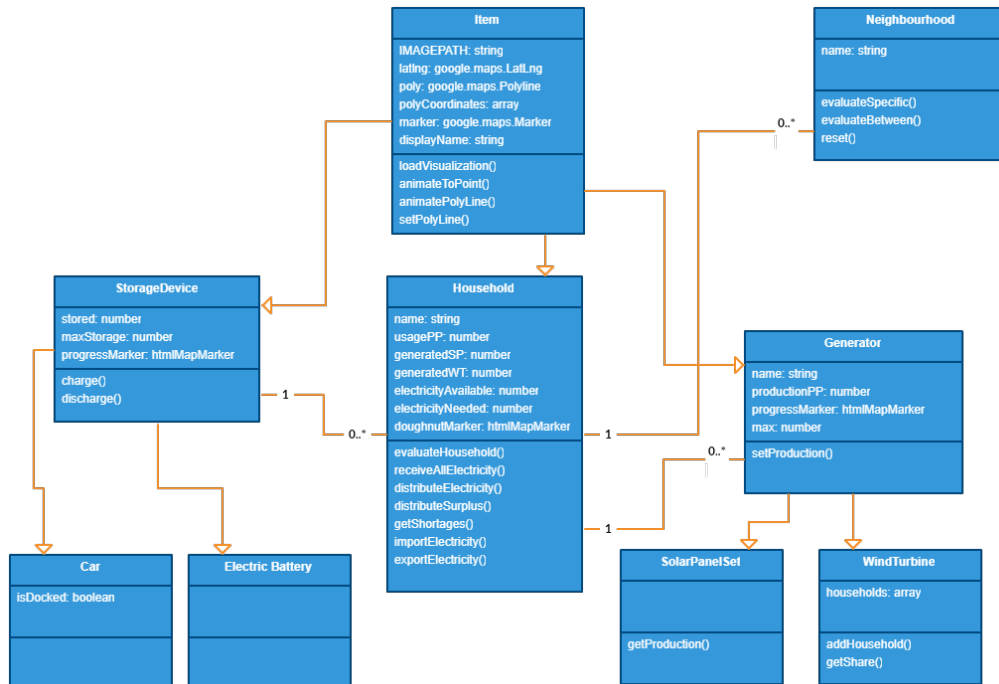
We decided to add a column to the time cases (csv files), which indicates the time at each interval, this time is then shown on the side-window. This gives the user some more understanding of the data presented.

Furthermore we added some more details to the doughnut graph around the household. Up to now this graph only showed the relation between production and consumption, and we appropriately updated this values when electricity was exchanged. We decided to add data values for the export and import of electricity as well as for the charging and discharging of a storage device. This way, it was more clear to the user where the electricity was going to and coming from.

Lastly we had to fix some bugs that were occurring, as well as refactor the code.

### 7.3 Documentation

The class diagram is shown below and contains the most vital classes together with their most important functions and variables.



The *Neighbourhood* class contains 0 or more households. The class contains the name of the neighbourhood and has a function to evaluate all specific households in the neighbourhood. Furthermore, it contains a function to evaluate between households, this function handles the trading between different households. Lastly it contains a function to reset the whole neighbourhood, which also makes sure the whole map is cleared.

The class responsible for handling the flow of electricity is the *Household* class. Firstly it has functions to handle the flow within the household. So it has a function to "receive" electricity from its generators (solar panels or a wind turbine) and a function to distribute electricity to its storage devices (cars or electric batteries). Secondly it has functions to either export or import electricity, these functions thus handle the trading of electricity from and to another household. Lastly it has variables that store the consumption over the last period, production over the last period, electricity available right now and the electricity needed right now.

The household class can contain 0 or more storage devices, contained in the *StorageDevice* class. This class stores, in a variable, the amount of electricity stored right now and the maximum storage capacity. Furthermore the class has functions to either charge or discharge the device.

There are two classes extending from the *StorageDevice* class: *Car* and *ElectricBattery*. This was mainly done to make the separation between the two storage devices more clear. The *Car* class has one additional variable indicating if it is docked or not.

Finally the household class can contain 0 or more generators, defined in the *Generator* class. The class stores the amount of electricity over the last period and the maximum amount of production possible.

Again we have two classes that extend from the *Generator* class: *SolarPanelSet* and *WindTurbine*. The solar panel class is quite simple, it has a function which returns the production over the last period. One set of solar panels can belong to only one household. However a wind turbine can be connected to multiple households and therefore stored all the households it is connected to. The wind turbine class has functions to add a household and return the share of production for a specific household.

All the classes, except for the neighbourhood, extend from the *Item* class. This was done because all these classes have a lot of features in common, mainly corresponding to the fact that they are all markers on the map so they all need the same certain functions and variables.

We have three additional files which are not obtained in the class diagram: *htmlMapMarker*, *helper* and *constants*. As explained before, in the *htmlMapMarker* class we implement a feature which allows us to add markers to the map with custom html code. The *serverCommunication* file contains a function to convert a csv file, which we receive from a server, to a usable array. Lastly the *constants* file contains a number of constants describing the offset of the various objects from the household (cars, wind turbines, solar panels and electric batteries), furthermore it specifies constants that store how much time is between the updating of graphs and polylines.

Lastly all of the server logic is contained in *server.js*. The server is able to process four different get requests:

1. **2housesdec**
2. **2housesjul**
3. **8housesdec**
4. **8housesjul**

All four of the requests directly correspond to one of the four use cases available. After receiving the request, the server will first confirm the connection with the client in the following way:

```

response.writeHead(200, {
  'Content-Type': 'text/event-stream',
  'Cache-Control': 'no-cache',
  'Connection': 'keep-alive',
  'Access-Control-Allow-Origin': 'http://localhost:3000',
})

```

The *Content-Type* header indicates that the content will be valid server sent event messages.

The *Cache-Control* header tells the client not to cache the information.

The *Access-Control-Allow-Origin* header indicates that the response can be shared with requesting code from the given origin.

Finally the *Connection* header tells the client to keep the connection alive so that events can be sent over the same connection over time, safely reaching the client.

After establishing the connection, the server will read the correct csv and json files from their storage location and start sending events in intervals using the function `sendEvents()`.

## 7.4 Functionality

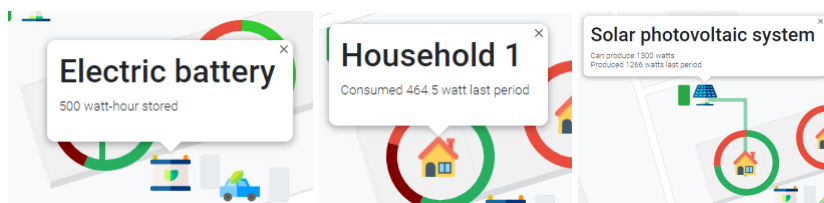
Firstly, as explained before, we have a side window on the left side of the interface. This window displays some basic information about the interface at that point.

Since we have adapted the time cases in such a way that it sends the time for the corresponding interval, we display this at the top of the window, as can be seen below.

Furthermore, for each household we display its name, its generators and its storage devices. Also after every round of production of electricity, we display the amount that was generated next to the generator.



Similarly we show some more information in the form of pop-up info windows. For each marker on the map, if the user puts its mouse on the icon, a window will pop up with some additional information about the object. Some examples are shown below for a battery, household and solar panel system.

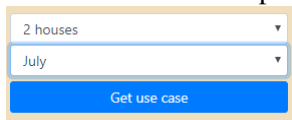


The four use cases available are as follows:

- 8 households, July

- 8 households, December
- 2 households, July
- 2 households, December

We thus had to give the user the option to choose which use case it wanted to visualise. We created a form where the user could indicate the number of households and the month. After the user has chose both options it clicks the button "Get use case" and a connection with the server is established. From that point on the server sends a stream of use cases to the interface, every interval a new use cases is received and processed. The form which shows the options can be seen in the image below.



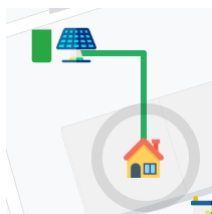
The image shows a web form with two dropdown menus. The first dropdown menu is labeled "2 houses" and the second is labeled "July". Below the dropdowns is a blue button with the text "Get use case".

After the user clicks the button, a request will be send to the server and the server will start sending events. The first thing the server sends is the configuration file, the interface processes this file thereby initialising all the right objects. All these objects are then rendered on the map, including the progress bars for the generators and storage devices. The scene that is first seen by the user can be seen in the image below (for two households).



Since the configuration file has been sent and processed, the server will now start sending the time cases. The processing of these files is done in a number of phases. The first phase is the distribution of electricity: a line is drawn from a generator (solar panel or wind turbine) which visualises the production of electricity. Furthermore the progress bars for the generators are updated with respect to their production. This can be seen in the image below:

**Phase I:** Distribution of electricity.



After receiving the produced electricity, the doughnut chart of the households will be updated to show the ratio of production and consumption. As can be seen in the image below, the first household has overproduction since the green (production) part is bigger than the red part (consumption). The second household does not produce any electricity so its chart is completely red. In the case that both

sides are equal in size, the chart is balanced.

**Phase II:** Show ratio production and consumption.



Followed by the processing of production and consumption is the trading of electricity. In case of underproduction for a specific household, this household will loop over all the other households and export electricity from them if possible. This is possible when some other household has overproduced electricity. As we said before the first household has overproduction, therefore electricity will be trading and imported into the second household. We take the coordinates of both households and calculate the path from one coordinate to the other in intervals. Since the coordinates are added in intervals, there will be a line moving or animated from one household to the other.

**Phase III:** Trading of electricity



As a result of the trading of electricity, the charts again need to be updated appropriately.

Since the first household exported electricity, the red section of the chart will become bigger while the green part will become smaller relatively. We indicate the export of electricity with a different shade of red to make it clear that this part was not consumed but exported. In the image below can be seen the household is not balanced yet, there is still a small amount of surplus present.

Furthermore the second household imported electricity so we update the chart for that household as well. We indicate the imported electricity with a different shade of green (from the production shade) to indicate that it was not produced by the household itself. As can be seen in the image below the second household is now completely balanced.

**Phase IV:** Update chart for export and import

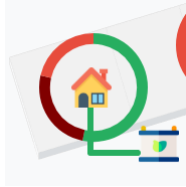


The last step in the flow of electricity is the charging or discharging of electricity. If some household still needs electricity to suffice its needs, a potential storage device will be discharged. On the other hand if some household still has excess, electricity will be charged into some storage device.

In the current scenario, the first household still has some surplus, which will be charged into the electric battery owned by the household. This is visualised by a green line going from the household

to the battery and can be seen in the image below. The discharging of a storage device would be the same, except for the fact that the line will be moving from the storage device to the household.

**Phase V:** Charging and discharging electricity.



Now that the flow of electricity has been completed for this interval, we can update the chart one last time to see if the households are balanced.

The first household charged some electricity, we therefore again update the red section of the chart which will increase in size relatively. The red part indicating the charging of electricity will again be a different shade of red (the same would apply in case of discharging for the colour green). The green part will somewhat decrease in size to balance the chart.

As can be seen in the image below, both households were able suffice in their needs by producing, charging, exporting and importing of electricity.

**Phase VI:** Update chart for charging and discharging



To conclude, all of the functions described above make sure that all of the requirements, specified in section 5.3, have been met.

## 8 Implementation Sketchup

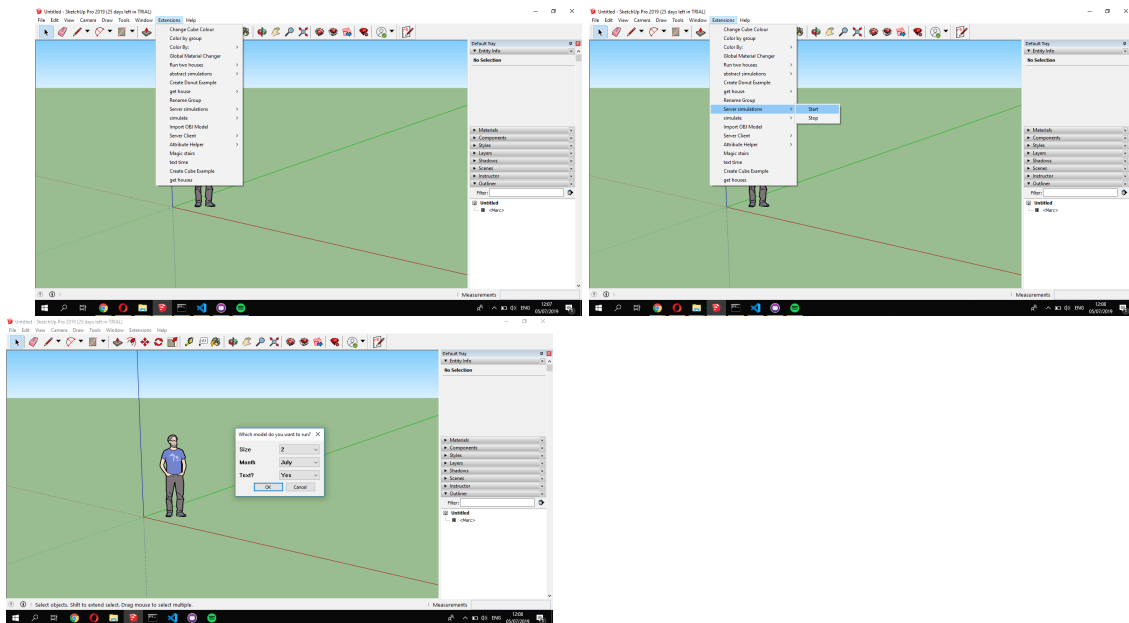
In this implementation we decided to make 3D models in Sketchup, where the user could see the electricity usage through colour changes and optionally through text. This would have to be done through making a plugin, which would be programmed using Ruby. Along with a backend server which would send the data line by line. The server in this cases was a TCP socket, and sent lines of the CSV data file to the plugin.

### 8.1 Functionality

In general the simulations are run as a plugin inside Sketchup, easily installed by going to the plugin file inside Sketchup, and then accepted in the extension manager, and run through the extension option in Sketchup. The simulation will run if the backend server is on. Once you have selected the option of *Server Simulation* you will get the option of choosing which model you want to run. Which are:

- 8 households, July
- 8 households, December
- 2 households, July
- 2 households, December

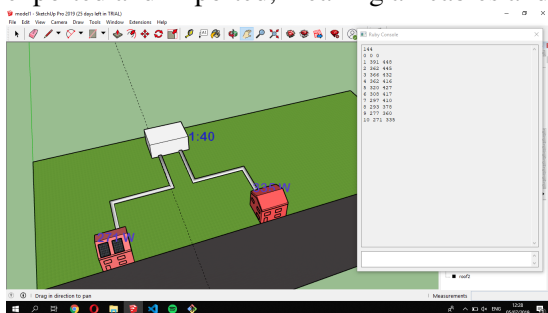
Along with choosing if you want text to be displayed or not. There is an option to stop the simulation, and this is done by clicking on *Server Simulation* and choosing the stop option



Once run you can move around the model and view the simulation. The simulation will be for a whole day which will take 144 iterations, every iteration taking 1 second. Moving around the model, you will see the text move with the camera and change size according to how close you are to the text. Every iteration the data is received and processed and is shown in the 3D display. This is made out of 4 general cases.

### Case 1

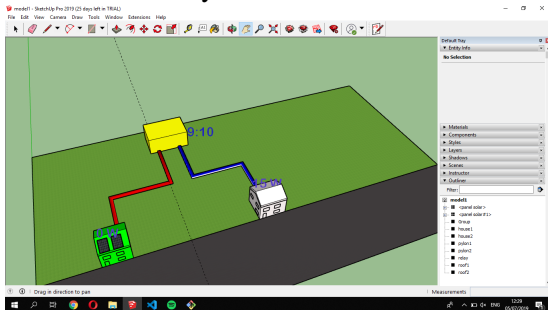
All houses are running a deficit. This will mean that all houses will be red and no electricity will be exported and imported, meaning all cables and the relay are white.



### Case 2

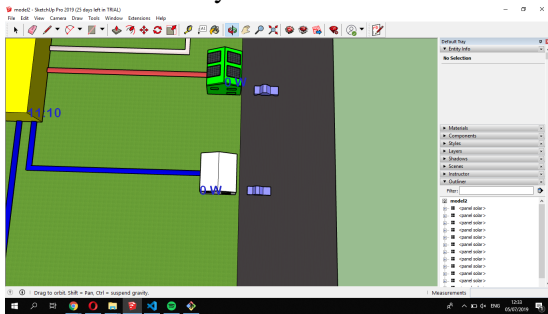
This is when one of the houses produces a surplus and exports electricity. This means that the other house(s) receive this electricity. This will make the exporting house's cable go red, and the importing

house's cable to blue, the relay will go yellow signalling that electricity is being traded in the community. This can either be enough for another house, which means it will turn white if it fulfils all the house's electricity needs.



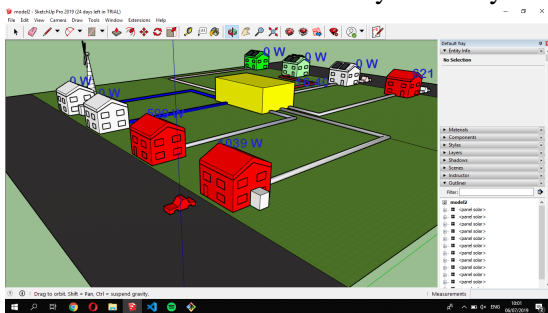
### Case 3

If there is full overproduction, then for the 8 household model, the electricity can be exported to cars and batteries. They will become blue when charging.



### Case 4

If the batteries contain electricity then they can funnel electricity back to the house, becoming red.



## 8.2 Design choices

Like the data sets, we chose two models. One having two houses and one having eight houses. The houses are all linked up with a cable to a box which is meant to represent a relay, so the surplus electricity gets sent to the other houses. The cars are outside the owner's homes and the batteries are represented as small cubes next to the houses. Furthermore the solar panels are placed above the roofs and the windmill is placed behind house1. All objects are abstract representations, this was firstly done because it makes the programming side easier to colour the objects, and therefore makes the changes in colours more obvious. Secondly it was done to save time.

The houses change colour according to the amount of electricity they are net using. If they are net neutral, so the total usage minus the total production then the house will turn white. This is the same case for when there is no electricity going through the cables or relay. The batteries do the same if



they are not being charged or discharged.

When the houses are in an electrical deficit, using more electricity than producing, they turn red. At 500 Watts or more they will be red. Below that they will be a lighter red or pinkish tone till eventually becoming white. For a surplus, at -500 Watts or below, the houses will be green. Everything above that it will become more white.

The cables, they become red if electricity is being exported from a house to another house, and blue when the a house is receiving electricity. The situation is the same for when a car or battery is exporting or receiving electricity. Both have a scale, where if they are passing 200 watts, then they become red or blue, depending exporting or importing. Nearer to zero they will become more white. Finally the windmill, can only become green when producing net electricity, not red like the houses.

For the houses we decided to use green, white and red. Green because we associate production of green electricity and possibly surpluses in general being good and green is a colour associated with it. Red being the opposite colour and often used in contrast to green, was therefore an obvious choice. Think fire vs plants, or stocks going down vs up. White we felt was a good colour to show no change or neutrality. For the relay we felt yellow versus white was good, as to signify a light going on and off. For the cars, cables and batteries red versus blue felt like a good combination. Red signifying loss of blood, and blue being the opposite of that. We felt it was best not to have green instead of blue. As it would make people viewing the model, think that these objects were producing electricity. Lastly for the scales (-)200 Watts and (-)500 Watts was decided as it reflected the average in the data found.

Above every house there is some text showing the amount of electricity used or produced at a given moment. Above the relay there is text showing the current time. We chose the text to be size 24 and blue, so that it would stand up against the houses. Furthermore, the text moves and changes size when you move around the model.

### **8.3 Details**

#### **Technologies used:**

- Ruby
- Ruby Sketchup API [12]
- Sketchup 2019 Pro

The reason for using Ruby and Ruby Sketchup API, was because these were easiest and had the most resources to use as apposed to C, C++. Furthermore, as Ruby uses an interpreter debugging was a lot easier. We decided that using the latest version of Sketchup was the best approach in order to use the most efficient and powerful version of Sketchup around. All other free versions apart from Make 2017, were not possible as you cannot run/make plugins for them. Note the only version this plugin properly works on is Sketchup pro 2019, it can be run on Sketchup Make 2017, however that is ill advised as visually it is less appealing and works less smoothly.

#### **Resources used:**

- Sketchup forum
- ruby tutorials and forums
- Examples online, namely from the School of architecture of Nancy [13]

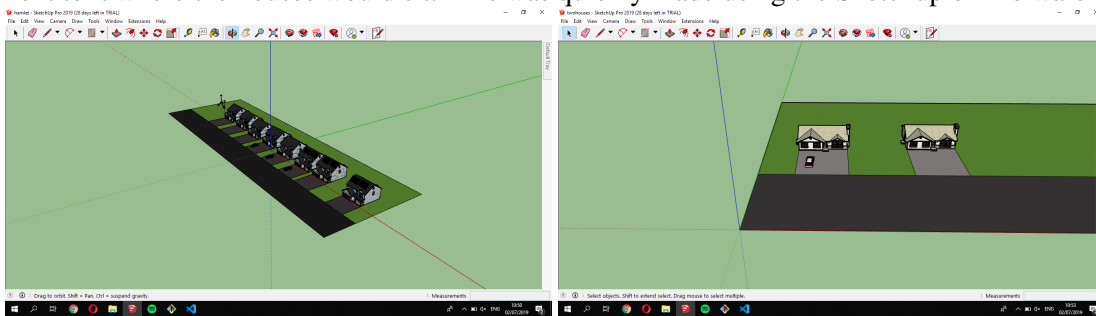
- Sketchup book *Automatic Sketchup* [14]

The last two resources were the most important. The examples were all open source, so it was possible to tweak them and see how they worked. This is not always the case with plugins in the extension warehouse. The book explained a lot more in depth how Sketchup worked, compared to the ruby Sketchup API. Both were very helpful despite being relatively old. To conclude, all of the functions and design choices described above make sure that all of the requirements, specified in section 5.3, have been met.

## 8.4 Project building process

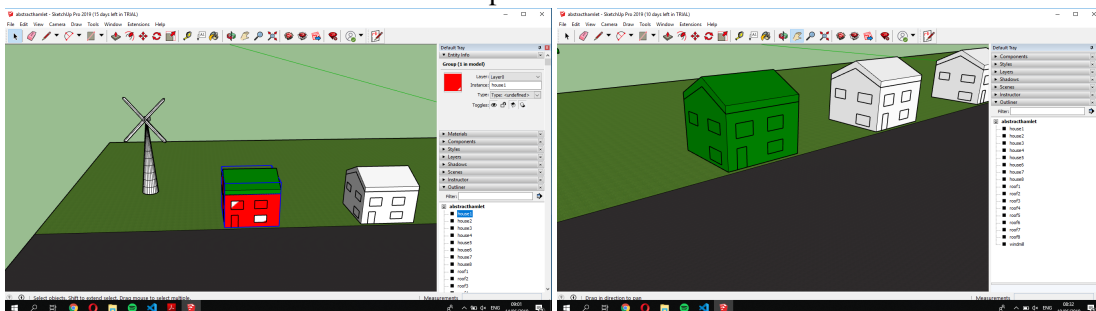
The model was first started up with experimenting with making shapes and being able to change their colour. Furthermore, using available plugins and code on forums to see how they had programmed certain aspects. Along with getting to know how the Sketchup API works. Learning what is and isn't possible in Sketchup.

Getting a basic sense of how the model should look like, was also done in the initial stage. It would be a plane where you could see a tarmac street where the cars are, and a green grass looking plane next to it where the houses would sit. This was quickly made using the Sketchup online warehouse.



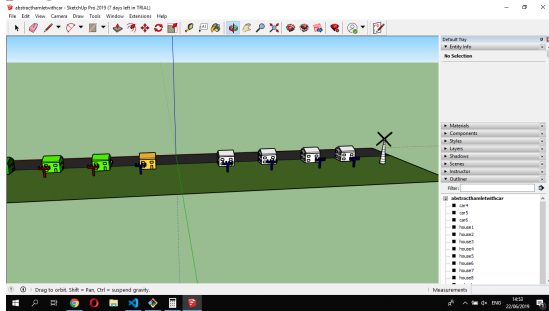
The initial stage was being able to make cubes and be able to change their colour several times. This was then turned into loading a house and changing colour, which progressed into several houses changing colour into several different colours. Finally through a server, which involved threading.

The experimenting was finalised by building the houses, in an abstract form, which meant that the houses were simple and had no textures. This means that from the programming side of things it is a lot easier to implement. Furthermore, The roof is a separate entity, which means that it potentially can be another colour than the house it is part of.

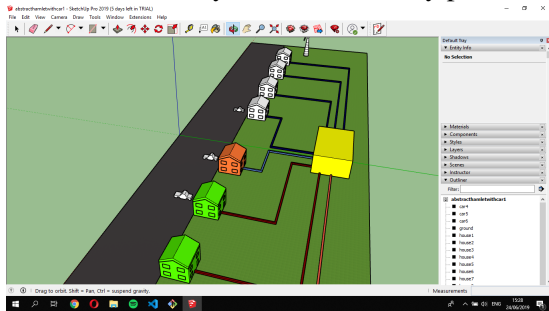


The first proper plugin involved the model being loaded, the plugin loading the CSV data and calculating the difference between the production and the consumption. If there was a surplus it would turn green, otherwise it would turn red. Every iteration i.e. real world 10 minutes, would take 1 second in the model.

In the second iteration of the plugin, there was a backend TCP socket added. Which loads the requested CSV file, and line by line sends the data to the plugin. Third iteration added pylons to accommodate the possibility of sending surplus power to other homes. The pylons would change colour according to if they were exporting or importing electricity, same as the cables. Furthermore, in the plugin the functions were made to transfer electricity from one house to another.



The fourth iteration, converted the pylons into cables as they were not visually very clear in comparison to cables. Furthermore a scale was added between green and red, so that the user could see how large the deficit was. It was initially chosen that yellow was the neutral colour. Finally a relay was added to show that the electricity was being transferred and all the cables would go in and out of it, and it would turn yellow if electricity passed through it.



Finally, the scale was changed to red, white and green for the houses. Batteries and cars were added to take up the complete surplus of electricity and give back electricity to the owner's house in times of a deficit. Text was added, as an opt in, to see exactly how much a house is producing at a certain time and see the time displayed above the relay. Furthermore, the eight house model was made more compact having four versus four instead of a line of eight houses. Finally the code was debugged and refactored. The plugin, now analysis the model to find all the entities, making the code more versatile and easier to use when expanding it for other models.

## 8.5 Scaling

Scaling this project would come in two different cases. Firstly, how will it look like visually? Will it be a ramshackle mess of all the colours of the rainbow, or a relatively nice looking and easy to understand grand simulation. Secondly, how would this work under the hood? Do many things need to be changed? or can this quite easily be made to work.

### 8.5.1 visual

This can be answered in two parts, by answering the following question. Does the user need to see the electricity be transferred to other houses? If the answer is no, then it would mean that you can remove the current cables and the relay, and the model can very simply be expanded to look like a

suburb or a bigger village. If the answer is yes, then that would probably mean that the maximum houses around a relay would probably be 10, and everything more than that you would need separate relays and connect the relays up, with a cable showing the flow from one part of the community going to another. If this were to be greatly expanded then there should be groups of houses all connected up with a local relay, these relays should then be connected up to a central relay. This could probably visually be alright till a maximum of 80-100 houses. For extra cars or batteries this can probably quite easily be accommodated for. Likewise, if the user wishes to make an area/field full of windmills, these objects could take up the places otherwise occupied by houses.

### **8.5.2 practical**

There are two issues concerning the practical part, the first one being the data and the second one being the model.

The data can quite easily be generated by the excel sheet. However if the user wishes it to be more dynamic, this would have to be done through generating all types of houses' electrical use, and possible production made into single column CSV files. When requesting the data, the server would stick the files together and make a new file which it would send to the plugin.

For the model part, one could make a new model and simply add it to the code.

However, to make it properly scale better one would probably want to be able to make a script where the program builds the model, using scripting and building the houses from scratch. This is a difficult task in making it fully dynamic, although not impossible. However, if there is a general range of how many houses a person would want to simulate this can also be done by using the hide function in Sketchup, where the user simply selects what he/she wants, and everything else gets hidden. This would be a more simple solution.

## **9 Discussion**

In this section we will discuss some of the strengths and weaknesses of both implementations. Furthermore we will compare both implementations with each other and lastly discuss some alternative solutions.

### **9.1 Strengths and Weaknesses of the Web interface**

Currently a user of the web interface is able to easily see the flow of electricity between a number of households. Furthermore the interface provides the user with information about the neighbourhood through a side window, animations and pop-up info windows, as described in section 7.4.

However a weakness leading from this, is that the interface is constrained to a few households. It is possible to add more households, but the user would need to move the map around to see all of the households present on the map. A possible solution would be zooming out the map, but this would lead to a less clear interface, since everything would be much harder to see. We therefore choose quality over quantity, i.e. we would have rather have a few households with clear interaction than a lot of households where the interaction would be much more complex.

Regarding the construction of the project, an advantage is that the Google Maps API is very easy to use. There is much information about the API available on the web, which makes it easy to find solutions or implementations that are needed.

Lastly we have constructed the code in such a way that the model is quite dynamic. Using the configuration cases it is quite easy to add different kinds of combinations. A disadvantage is that right

now, through the configuration cases, it is only possible to link a wind turbine to a single household. The code supports adding more households to a wind turbine but we didn't implement this in the server.

## 9.2 Strengths and Weaknesses of Sketchup

The strengths of Sketchup are that it is relatively easy to build 3D models, give the entities in the model colour and modify the entities. Sketchup boasts some impressive plugins and a vast 3D warehouse, that can be used if one wanted to, such as filming, rendering and adding augmented reality.

Weaknesses are that it is a rather niche API, and it is very hard to add other libraries to the plugin. Making the usage of ruby in Sketchup almost exclusive to the Sketchup API and basic ruby functions. Meaning, that all the huge amount of libraries available for ruby are not able to be used in Sketchup. Furthermore currently all simulations take 144 minutes, and uses TCP sockets. Which means the program is rather restricted and local. Furthermore it is not fully dynamic, meaning the code would have to be changed if more models were added. The text used is not optimal, as it can go inside a building and become so big that it engrosses the whole model. Some smaller frustrations are that the units used in the ruby console in Sketchup are imperial not metric. Meaning that if you give the coordinates e.g. [1,1,1] this will mean [0.0254 m ,0.0254 m ,0.0254 m] so you will have to constantly change between metric and imperial. Furthermore, working with entities which are the objects in the model, is quite a messy business. However, to conclude for 3D modelling and simulations, Sketchup is probably the best tool to use. Not only in ease of use, but also because of the vast amounts of online 3D models and plugins that can easily be loaded and used.

## 9.3 Sketchup visualisation versus Web Interface visualisation

While there are some similarities between both of the implementations, there are also some differences. We will compare both visualisations by looking at some of the advantages that each one has over the other.

Firstly an advantage of the Sketchup implementation is that it is three-dimensional, while the web interface is two-dimensional. This leads to the fact that it can give the user an immersive experience when running a simulation. Furthermore it makes the model much more detailed in terms of the objects being visualised. Combined with the ability to move around and focus in on other parts of the model, it makes the model much more realistic.

However the Interface implementations also has some advantages over the Sketchup one. For example, the interface is quite simple to use, you simply load the webpage and specify a use case. By comparison, the Sketchup implementation is a little bit more complicated to the typical user. Furthermore it isn't fully dynamic, meaning the user will have to go through some more steps in order to visualise the correct use case.

Another advantage of the interface is that it is somewhat easier on the eye. All the icons on the map are very simple and basic, conversely the objects in the Sketchup implementation are quite complex. Lastly, most of the technology used for the web interface is free, or has many free alternatives. This is not the case for Sketchup pro, one has to buy a version to use it. Additionally, there are few free alternatives.

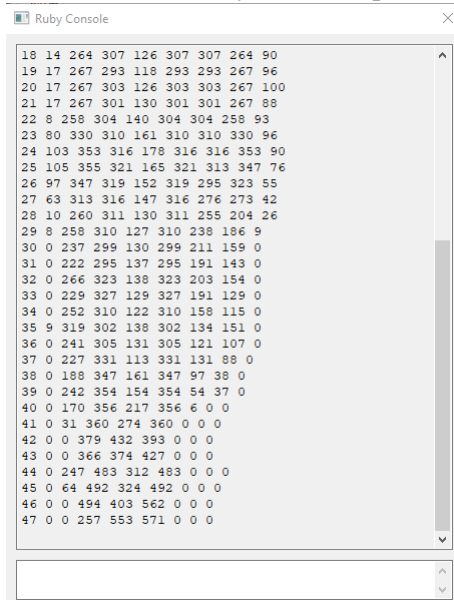
To conclude we can't clearly say that one implementation is superior. The use of either of one of the visualisations greatly depends on the type of user and the reason for using it, which leads to the user preferring one over the other.

## 9.4 Current solutions versus other potential solutions

Our current solutions are not the only way of visualising local energy communities. Below, there are a number of other potential implementations, which we will compare and contrast with our own solutions.

### Simple text

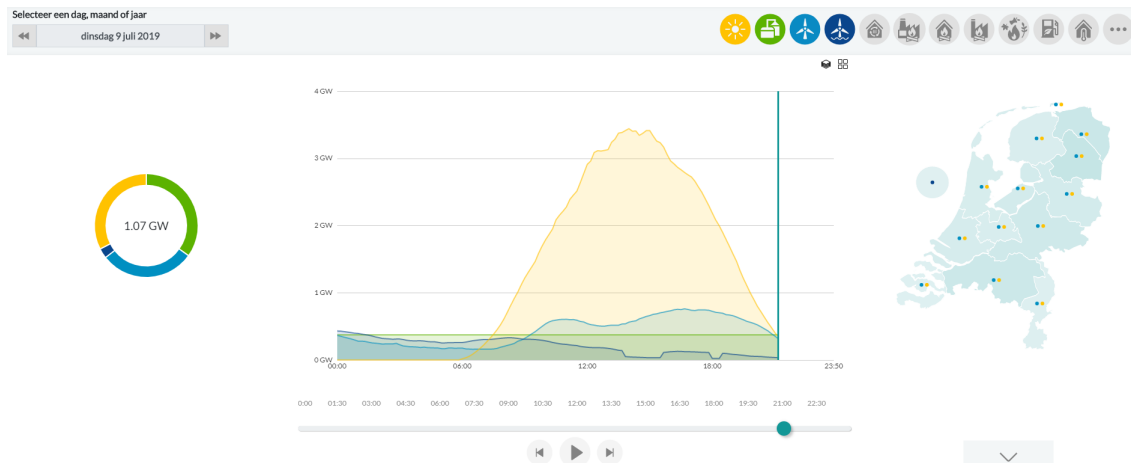
The most basic alternative would be simple text, showing for every house how much they are (in total) producing and consuming. Most likely, made out of a number of columns representing the number of houses, similar to the picture below. Additionally, the text could change colour depending on if there is overproduction or underproduction. This solution is less visually appealing than the other current solutions. As it is not able to show the transfer or storage of power very well and neither able to visually scale very well. However, it is very basic and is able to quickly give you a general sense of how much electricity would be produced and consumed in a small community.



```
Ruby Console
18 14 264 307 126 307 307 264 90
19 17 267 293 118 293 293 267 96
20 17 267 303 126 303 303 267 100
21 17 267 301 130 301 301 267 88
22 8 258 304 140 304 304 258 93
23 80 330 310 161 310 310 330 96
24 103 353 316 178 316 316 353 90
25 105 355 321 165 321 313 347 76
26 97 347 319 152 319 295 323 55
27 63 313 316 147 316 276 273 42
28 10 260 311 130 311 255 204 26
29 8 258 310 127 310 238 186 9
30 0 237 299 130 299 211 159 0
31 0 222 295 137 295 191 143 0
32 0 266 323 138 323 203 154 0
33 0 229 327 129 327 191 129 0
34 0 252 310 122 310 158 115 0
35 9 319 302 138 302 134 151 0
36 0 241 305 131 305 121 107 0
37 0 227 331 113 331 131 88 0
38 0 188 347 161 347 97 38 0
39 0 242 354 154 354 54 37 0
40 0 170 356 217 356 6 0 0
41 0 31 360 274 360 0 0 0
42 0 0 379 432 393 0 0 0
43 0 0 366 374 427 0 0 0
44 0 247 483 312 483 0 0 0
45 0 64 492 324 492 0 0 0
46 0 0 494 403 562 0 0 0
47 0 0 257 553 571 0 0 0
```

### Graphs

An alternative to the current options would be a live graph showing the amount of energy consumed, produced and stored during the day. This can be good in giving the user a general view of how much electricity is exactly being produced. Like the picture below from the website [energieopwek.nl](http://energieopwek.nl) [15], which shows how much electricity is being produced by alternative electricity sources during the day in The Netherlands. Despite its clean and simple appearance, this will only be able to give a general view of the production and consumption. Neither will it realistically be easy to show more than four houses. Furthermore, it will not be able to show the movement of electricity between houses or between houses and batteries.



### Static web model

Another solution would be to have a static web model, where unlike the current web model you will not have animation or graphs. You will simply see the roofs of houses, where the colour corresponds to the amount of electricity being produced or consumed. This would have several problems, for example storage and transferring of electricity would be hard to visualise, which both other solutions are able to do. However, this model will probably be a lot better when it comes to scaling and being able to visualise a large community.



## 10 Improvements and known problems

### 10.1 General

A general improvement, would be that both programs were to use the same backend server. This is unfortunately not the case. Moreover, it would be an improvement if it were possible to generate different cases more easily. In the easiest sense meaning different months and changing the population of every house. In more advanced cases being able to change the specific weather, specific days of the week, make longer or shorter models and different types of houses. Unfortunately, due to time constraints and the pool of data we acquired this was not possible. Certainly, concerning the data for the solar and wind power we cannot make simulations for different kinds of weather conditions. To make more precise and interesting simulations, one would probably need to use tools such as Open studio [16] or TRNSYS [17] to make estimations of how much a house would consume, and how much electricity would be produced. Alternatively, one would have to look for other surveys.

### 10.2 Web Interface

Firstly a known problem is that the animation of the paths can act weird when going to a different tab and then going back to the tab of the interface. We however have identified this as an issue of the Google Maps API and thus not something we can easily fix. We also don't identify this as a significant problem since the problem is only temporary.

A good improvement would be if it would be possible to dynamically add more than one household to a wind turbine. The code for this is already available but it is not yet possible to specify this from the configuration cases.

Another possible improvement would be the possibility to 'pause' the interface at any moment. This would give the user some additional time to see the status of the interface at a specific point in time.

Lastly an additional way to give the user more input into what is happening in the interface, would be to give an option to specify at which time the interface starts. Right now this feature is hardcoded on the server side at noon, but it would be interesting if the user could specify this on the client side.

### 10.3 Sketchup

Currently the backend is quite basic, and can be done with improvements. Furthermore the server does not respond well to the stop button, and can crash if wanting to do a new simulation after using the stop function.

Furthermore, currently there is no way in telling how much a car or battery has stored. Therefore, the user cannot foresee how long the battery will be able to be used, and how much capacity it still has left.

As mentioned before it can greatly be improved on, by making the program more dynamic, and the user being able to make more choices in the simulations. Another possible improvement would be the possibility to 'pause' the interface at any moment. Lastly the interface can be improved on by making it link up to a website, or JavaScript interface. This would make it easier to use if there are more options available in the simulation.



## 11 Conclusion

In conclusion, both the web interface as the Sketchup part of the project can be considered to be quite complete. Because both are able to show, through colours and animation, when electricity is being produced, and in what quantity. Additionally both implementations can show electricity moving from one object to another in a clear way.

As discussed in section 10 there are some small problems with the current implementations. Despite these problems, they are superior compared to the other potential implementations that we discussed in section 9. Because none of the other possible solutions were able to either scale as well or show the use of storage and movement of electricity.

Furthermore there are some interesting improvements to be done. These improvements are mainly concerning giving the user more input in the data being visualised. This is however not part of a minimum viable project but mainly a change that can be added in the future.

## References

- [1] Interreg Europe, “Renewable energy communities,” 2018, [Online; accessed 19-June-2019]. [Online]. Available: [https://www.interregeurope.eu/fileadmin/user\\_upload/plp\\_uploads/policy\\_briefs/2018-08-30\\_Policy\\_brief\\_Renewable\\_Energy\\_Communities\\_PB\\_TO4\\_final.pdf](https://www.interregeurope.eu/fileadmin/user_upload/plp_uploads/policy_briefs/2018-08-30_Policy_brief_Renewable_Energy_Communities_PB_TO4_final.pdf)
- [2] Wikipedia contributors, “Renewable energy — Wikipedia, the free encyclopedia,” 2019, [Online; accessed 19-June-2019]. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Renewable\\_energy&oldid=902365770](https://en.wikipedia.org/w/index.php?title=Renewable_energy&oldid=902365770)
- [3] —, “Solar panel — Wikipedia, the free encyclopedia,” 2019, [Online; accessed 19-June-2019]. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Solar\\_panel&oldid=902518554](https://en.wikipedia.org/w/index.php?title=Solar_panel&oldid=902518554)
- [4] M. H. A. Chan, “Further analysis of data from the household electricity usage study: Correlation of consumption with low carbon technologies,” 2014, [Online; accessed 4-July-2019]. [Online]. Available: [https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment\\_data/file/326092/HEUS\\_Lot\\_II\\_Correlation\\_of\\_Consumption\\_with\\_Low\\_Carbon\\_Technologies\\_Fina....pdf](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/326092/HEUS_Lot_II_Correlation_of_Consumption_with_Low_Carbon_Technologies_Fina....pdf)
- [5] T. K. J Palmer, N Terry, “Further analysis of the household electricity survey early findings: Demand side management,” 2013, [Online; accessed 4-July-2019]. [Online]. Available: [https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment\\_data/file/275483/early\\_findings\\_revised.pdf](https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/275483/early_findings_revised.pdf)
- [6] NORDIC APIS, “5 powerful alternatives to google maps api,” 2018, [Online; accessed 2-July-2019]. [Online]. Available: <https://nordicapis.com/5-powerful-alternatives-to-google-maps-api/>
- [7] Google cloud, “Pricing sheet,” [Online; accessed 2-July-2019]. [Online]. Available: <https://cloud.google.com/maps-platform/pricing/sheet/>
- [8] Google Maps Platform, “Markers,” 2019, [Online; accessed 2-July-2019]. [Online]. Available: <https://developers.google.com/maps/documentation/javascript/markers>
- [9] —, “Custom overlays,” 2019, [Online; accessed 2-July-2019]. [Online]. Available: <https://developers.google.com/maps/documentation/javascript/examples/overlay-simple>
- [10] —, “Simple polylines,” 2019, [Online; accessed 2-July-2019]. [Online]. Available: <https://developers.google.com/maps/documentation/javascript/examples/polyline-simple>
- [11] Wikipedia contributors, “Server-sent events — Wikipedia, the free encyclopedia,” 2019, [Online; accessed 4-July-2019]. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Server-sent\\_events&oldid=903263539](https://en.wikipedia.org/w/index.php?title=Server-sent_events&oldid=903263539)
- [12] Sketchup, “Ruby sketchup api,” [Online; accessed 2-July-2019]. [Online]. Available: <https://ruby.sketchup.com/>
- [13] Various, “ruby library depot,” 2016, [Online; accessed 6-July-2019]. [Online]. Available: [http://www.crai.archi.fr/rld/plugins\\_list\\_az.php](http://www.crai.archi.fr/rld/plugins_list_az.php)
- [14] M. Scarpino, “Automatic sketchup -creating 3-d models in ruby,” 2010, [Online; accessed 4-July-2019]. [Online]. Available: [http://rhin.crai.archi.fr/rld/pdf/Automatic\\_SketchUp.pdf](http://rhin.crai.archi.fr/rld/pdf/Automatic_SketchUp.pdf)

- [15] energieopwek.nl, “energieopwek.nl,” 2019, [Online; accessed 10-July-2019]. [Online]. Available: <https://energieopwek.nl/>
- [16] Open Studio, “Open studio,” 2018, [Online; accessed 6-July-2019]. [Online]. Available: <https://www.openstudio.net/>
- [17] University of Wisconsin Madison, “A transient systems simulation program,” 2018, [Online; accessed 6-July-2019]. [Online]. Available: <http://sel.me.wisc.edu/trnsys/>