Practical course
Smart Energy Systems
Report

Sandro Speth
Markus Zilch
Dominik Wagner

Wintersemester 18/19

1 Introduction

The traditional power grid is changing more and more over time. Due to increasing sensitization for the use of renewable and reliable sources of energy instead of nuclear power sources or fossil energy sources, there is an increasing accommodation of renewable energy. To fulfill our daily energy need only with such energy sources is quite difficult and needs lot of planning and simulation. In this work we build a smart energy system to simulate a smart grid.

A smart grid is an energy efficient system with information and communication technology, automation and awareness of energy consumption. There are many different actors and technologies which are connected to each other and interoperate to optimize the grid.

A smart energy system creates the bridge between a power grid and a resilient and reliable smart grid. Users can simulate reliable energy sources, as well as different kinds of energy consumers, e.g. homes or offices. Simulation of distributed energy sources and automation of processes build an energy management system with interaction on different levels of the power grid and the consumer grids. In a smart grid a new aspect is becoming important. Contrary to currently used normal power grids the exchange of information becomes more and more important. Especially the consumer side (demand side) of these grids are now being integrated into the power systems in order to ensure optimal power usage and a better understanding of customer needs. Those new smart grids use various methods to control the flow of energy in order to optimize efficiency and grid usage. For a fully functional smart grid, the grid has to be made more resistant to any attacks and have mechanisms to self-diagnose problems and take appropriate recovery actions. The new smart grids offer better ways to integrate renewable but unstable power suppliers like windturbines and solarpanels through better controllable interactions on a much smaller scale than conventional power grids. Through these microgrids we can possibly rely completely on renewable energy sources in the future. This can be checked with our smart energy system.

The report is structured as follows. In section 2 we present our system design. First, we give some basic foundations which are relevant for our smart energy system, such as the difference between kWh and kW. Afterwards we present our functional requirements for the smart energy systems as user stories. From these functional requirements we created our architecture which will be described in this section as well.

2 System Design

2.1 Difference between kW and kWh

W is a measuring scale for energy applied per time instance. There are different possibilities to describe W in common terms. A pretty graphic one is the movement of mass. $1W$ equals $1kg$ of mass moved by 1 meter in one second: $1\frac{kg*m^2}{s^3}$. Or in electrical terms: $1W$ equals 1 Ampere of electrical power with a voltage of 1 Volt. Both of those formulas are equal to a much simpler Term for Watt: $1W = 1J/s$. In simple terms, 1 Watt is the same as one Joule of energy applied over 1 second. For completeness, $1kW = 1000W$ [**Nelson**, **Borvon**, **SIStandard**].

Wh are the common term for measuring energy consumption and production. $1Wh$ is $1W$ applied continuously over 1 hour. $1Wh = 1W * 1h = 1J/s * 3600s = 3600J$. For a scientific context the Wh therefore is simply not used, instead the common SI standard J is used. In comparison, Wh is the total amount of energy used. W is how much energy is used in a specified timeslot (mostly 1 second) [**EURichtlinie**, **Bundesgesetz**].

2.2 Difference between consumption and demand

Electricity consumption and electricity demand are two different properties and measured with different measurement units. The following section contains a description of both and an example at the end.

2.2.1 Demand

The demand is the rate of consumption of electricity or mathematical speaking the demand is the derivation of the consumption [**StonyBrookUniversity**]. Most of the time the demand is measured in Watt. If you turn on a 100W light bulb, it will demand 100W while it is turned on. At the same time the grid must provide electricity at a rate of 100W. In most cases it is possible to calculate the demand with the following formula [**Eggenberger**].

$$Demand = Voltage * Current$$

Some customers also have to pay for the demand or peak demand they have because if you have a higher (peak) demand the grid has to support this [**StonyBrookUniversity**, **enertiv**].

2.2.2 Consumption

It is easier to understand electricity consumption because we are more used to this concept [**StonyBrookUniversity**]. Many people deal with electricity consumption while paying their electricity bill because most German electricity meter measure only the consumption. The consumption is the amount of electricity used per time unit [**StonyBrookUniversity**][**enertiv**].

Most of the time the consumption is measured in kilowatt per hour. The formula to calculate the consumption is the following [**StonyBrookUniversity**].

$$Consumption = Demand * Time$$

For example, 5 W LED bulbs turned on for 1h have the consumption of 5 Wh.

2.2.3 Difference

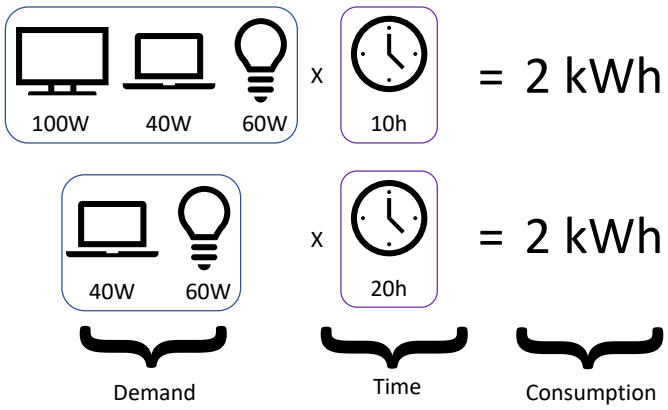

Figure 1: Example for demand and consumption

The demand is the rate of which we use energy and the consumption is the total energy used for a given time frame [**StonyBrookUniversity**]. The formulas also show this relation between the consumption and the demand. The consumption is the demand multiplied with the time. If someone turns on 1 heating unit with a demand of 1kW for 2 hours than the demand during this hours is 1kW, but the consumption is 2kWh. The consumption is the same if two heating units are used for half an hour, but the demand is doubled (2kW). Figure 1 contains another but similar example. To put it in simple terms the demand is comparable with the speed of a car and the consumption is the distance you drive. The faster you drive the more distance is accumulated over time.

2.3 Userstories

1. As a user, I need to create one or more wind turbines in the simulation so that I can calculate the potential energy output.

2. As a user, I need to create one or more photovoltaic panels in the simulation so that I can calculate the potential energy output.

3. As a user, I need to create one or more batteries in the simulation so to save unused energy of in my simulation.

4. As a user, I need to remove one or more suppler modules like wind turbines form the simulation to get an optimal mix of all kinds of suppliers.

5. As a user, I need to remove one or more consumer modules like commercial buildings form the simulation to get an optimal mix of all kinds of consumers.

6. As a user, I need to remove one or more batteries form the simulation to remove batteries which were added by mistake.

7. As a user, I need to get the charging state of my batteries to know the impact of the energy storage on the grid.

8. As a user, I need to create one or more homes on the demand side in the simulation so that I can simulate some energy consumer.

9. As a user, I need to create one or more commercial buildings on the demand side in the simulation to simulate some high energy consumer.

10. As a user, I need to get dynamic energy prices calculated from the simulation to determine if I want to sell my produced energy or store it for later use.

11. As a user, I need to use weather data in the simulation to simulate the smart energy system more precise and realistic.

12. As a user, I need to use already saved weather data in the simulation to not be dependent on the availability of the weather service.

13. As a user, I need to generate a forecast for energy generation and demand using the simulation in order to make informed decisions.

14. As a developer, I want to add more supplier modules than just wind turbines and photovoltaic panels to the simulation to improve the smart energy system in the future with further technology due to adding more kinds of suppliers.

15. As a developer, I want to add more consumer modules to the simulation to be able to add more kinds of consumers to the simulation.

16. As a user, I want to be able to create a smart energy system which is independent to a main power grid to simulate a reliable smart grid.

17. As a user, I want to get a visual notification if the supply of energy is smaller than the demand of energy to know when more energy supplier are needed.

18. As a user, I want to model a rechargeable battery so I can store the energy for later usage, if the supply is greater than the demand.

19. As a user, I need the battery to be able to discharge energy if the supply is lower than the demand in order to make my stored energy usable and keep the demand satisfied.

20. As a user, I need the smart grid to be able to manage peaks in the demand in order to smooth the impact on the grid and reduce the likelihood of power outages.

21. As a user, I need that the demand side consumers feature different load scenarios like home users and commercial users (constant load, occasionally peak loads) in order to make the simulation accurate for real life applications.

22. As a user, I want be able to use the system with my webbrowser so that I can use different platforms to view it and have easy access to the simulated data.

23. As a user, I want to be able to see the energy supply of each individual supply component in order to be able to assess the efficiency of the supplier.

24. As a user, I want to be able to see the energy demand of individual components for efficiency assessment and informed decision making.

25. As a user, I want to see a summary of energy supply and demand for all components in order to easily assess the current situation.

26. As a user, I want to adjust the demand by postponing the use of devices during peak hours in order to prevent a complete outage of the grid or to react to one-time-only scenarios.

27. As a user, I want the simulation to adapt the demand based on the price per kWh in order to minimize costs of my energy demands and maximize profits of my energy supply.

2.4 Architecture

This section describes our system architecture. An overview over the architecture is shown in figure 2.

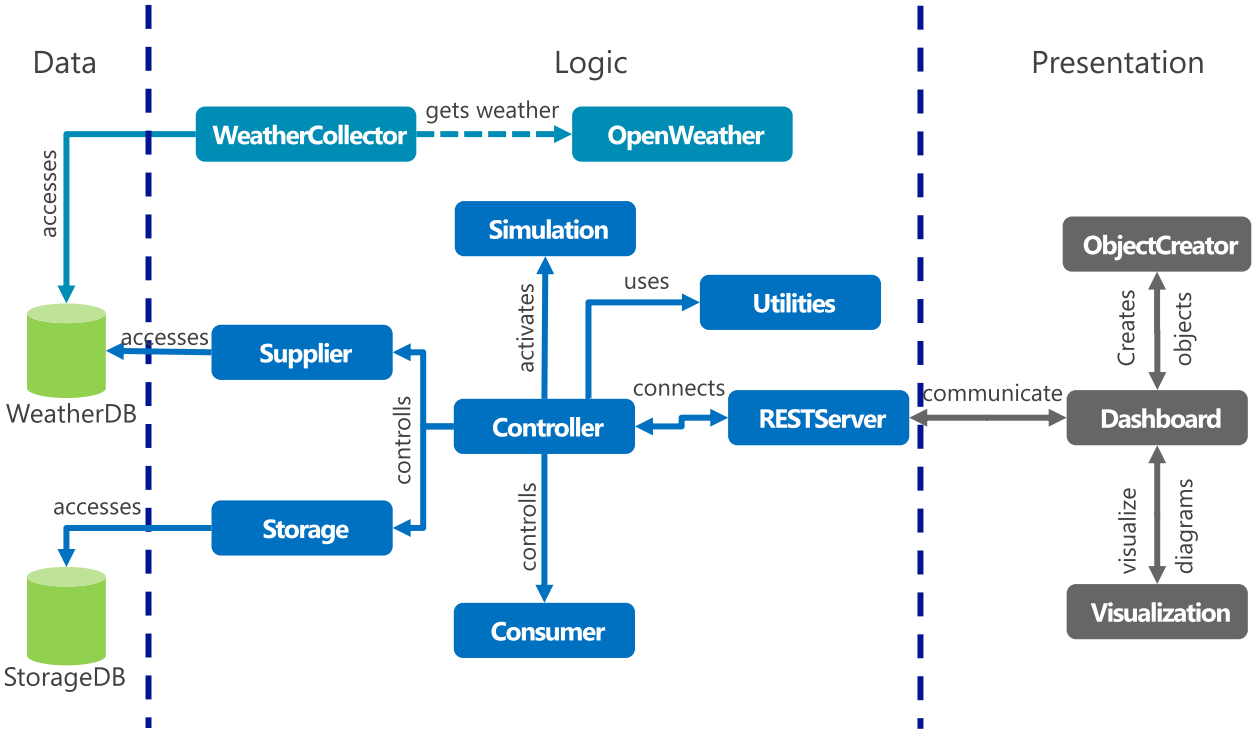

Figure 2: Smart energy system architecture

The `WeatherCollector` component is responsible for gathering actual weather data from *OpenWeather* API. This weather data is stored into a database for further use of the supply and demand components of the system.

In order for the whole system to be reliable and responsive we separated the `WeatherCollector` component from the rest of the system. It speaks only to the weather database where it get the locations for which it should collect weather data. The collected data gets written into the database and stored. Even in the case of a failure of the weather component or OpenWeather the already collected data is still accessible and therefore the system can work with it, making the system independent of the weather component. In the case of a new registration of a supplier component during a weather component failure, the database may return default values for the new component in a way the running system is not being halted by missing data. These measures should make the system reliable and and responsive in that context.

The three-layered architecture was already planned for in the first conceptions of the project, therefore no extra steps had to be taken. We divided the project in subcomponents and put them in the respective part of the architecture.
The first layer is the data layer which only consists of data-providing hardware and databases. Since the current project does not include data-providing hardware (as far as the current conception goes) only databases are left in this layer.
The second layer consists of the functional components of the project. All suppliers, storages, consumers and utility components are part of this layer, since they mostly take data from the databases and compute their respective power in- and output based on those values and give them to the simulation component. The simulation component is part of this layer, and takes the data the other component in order to model the different interactions of the smart grid. The workflow is controlled by the controller component and connects all components together. The last component in this layer is the REST component which makes the computed data from the simulation component available to the frontend layer.
The frontend layer is the third and last layer in our project. It handles the interaction with the user and makes it possible for the user to add and remove components to the smart grid simulation as he sees fit.

3 Implementation

3.1 Gather weather design

3.1.1 Weather service

For our smart energy system we need a weather service which can provide all needed data on a hourly basis. There is one weather service which fullfills our needs more than appropriate. The weather service `weatherbit.io` has different kinds of endpoints on its API. In our case we use two of them, the current weather data API and the 48 hours weather forecast API.

Before we take a look at the endpoints provided to the user and their responses, we want to present the data collectable on weatherbit.io. An example response of the current weather API is shown in listing 1. The property `data` of the JSON object contains an array of weather data points. For current weather API there is only one object in the array. But for the forecast there are up to 48 objects, one for every hour of the forecast.

Listing 1: Example response of weatherbit.io for $lat = 31.23$ and $lon = 121.47$

```
1 {
2     "data": [
3         {
```

```
            "wind_cdir": "SSE",
            "rh": 100,
            "pod": "n",
            "lon": 121.47,
            "pres": 1012.2,
            "timezone": "Asia/Shanghai",
            "ob_time": "2018-11-07 20:35",
            "country_code": "CN",
            "clouds": 0,
            "vis": 10,
            "solar_rad": 0,
            "state_code": "23",
            "wind_spd": 0.89,
            "lat": 31.23,
            "wind_cdir_full": "south-southeast",
            "slp": 1013.2,
            "datetime": "2018-11-07:20",
            "ts": 1541622900,
            "station": "E7205",
            "h_angle": -90,
            "dewpt": 13.9,
            "uv": 0,
            "dni": 0,
            "wind_dir": 156,
            "elev_angle": -29.2797,
            "ghi": 0,
            "dhi": 0,
            "precip": null,
            "city_name": "Shanghai",
            "weather": {
                "icon": "c01n",
                "code": "800",
                "description": "Clear sky"
            },
            "sunset": "09:00",
            "temp": 13.9,
            "sunrise": "22:14",
            "app_temp": 13.9
        }
    ],
    "count": 1
}
```


From all those data we chose only to store `lat` for latitute, `lon` for longitute, `pres` for air pressure, `rh` for relative humidity, `solar_rad` for solar radiation, `wind_spd` for wind speed, `temp` for the temperature, and the timestamp in UTC format.

The API for the current weather data returns current weather data of over 45.000 stations.

Every API request will return the nearest, and most recent observation. There are several endpoints available in the API. For example a user can get current weather observation by lat/lon as we do or by city name. All endpoints differ only on their required query parameters. They define whether to get the observation by lat/lon, city name or any other possible way. The basepath for all endpoints is `https://api.weatherbit.io/v2.0/current` and the supported method is `GET`. To get the data in metric format you have to add an additional query parameter `unit=m`. Every request must provide an API key in query parameters.[**weatherbit**] This API key can be requested through creating an account on weatherbit.io. After creating an account a user have to choose a pricing plan. For our system the free plan is totally sufficient.

The API for the weather forecast data returns the weather forecast for up to 48 hours on hourly basis. Nevertheless, we only request it for 24 hours. While the basepath is now `https://api.weatherbit.io/v2.0/forecast/hourly`, the provided query parameter are the same with an additional query parameter `hours` which must get a value between 0 and 48 to specify the size of the forecast.

3.1.2 Our WeatherCollector service

To gather weather data for different locations and store it into an database we build a service, which is under MIT licence available on GitHub[1]. The service runs a NodeJS express server with two endpoints to create or delete `WeatherCollector` objects. A `WeatherCollector` object gathers every hour current weather data for a given location as well as 24 hours weather forecast on a hourly basis and stores the data in a database. While gathering new forecast data, the old deprecated data is getting replaced, so there is always the latest 24 hour forecast data. Multiple WeatherCollector objects are possible to collect weather data for different locations at the same time.

There are several dependencies you have to install before running the server. Since the the code runs on NodeJS version 8 or higher with a current npm, you need to have this installed first. First, run `npm install` to install needed dependancy packages. You can look up the installed libraries and their versions in `package.json` file. After installing the packages, you need to create the database. Therefore, run `node DBCreator.js` to create the database file and required SQL tables. Now you are nearly ready to run the server.

You can test the functionality without running the complete server. You just need to run `WeatherCollectorTest.js` to play around with some locations. On the first run of the module you need to provide it with your weatherbit.io API key, latitute and longitute of the location: `node WeatherCollectorTest.js "<API Key>" <lat> <lon> "<mode>"`. Take in mind that you already need the API key requested from weatherbit.io. There are 5 modes available to test:

- `collect`: Collects a current weather data of the given location and stores it into the database
- `get`: Gets the latest stored current weather data of the given location
- `collect forecast`: Collects a 24 hour weather forecast on hourly basis for a given location and stores it into the database
- `get forecast`: Gets the latest version of the forecast weather data of the given location

[1] `https://github.com/smart-energy-system/WeatherCollector`

- `start`: Starts a WeatherCollector object on hourly picking new weather data (current and forecast) and stores the data into the database.

If you did run the complete run command once, a test config file was created. So unless you want to change your API key or the location you can shorten the run command a bit, using only `node WeatherCollectorTest.js "<mode>"`.

You can simply run the server with `npm start <API key>`. This will create a config file with your API key, so later you can start the server just running `npm start` command.

Since the service is a REST service, there are two HTTP REST endpoints available on the server, one for creating WeatherCollector objects and one for deleting them. To create a new WeatherCollector object, you can send an `POST` request against `<basepath>/weathercollectors` providing the body shown in listing 2 as content-type `application/json`.

Listing 2: Body of `POST` request endpoint

```
{
    "lat":<lat>,
    "lon": <lon>
}
```

If `lat` and `lon` are correct and the object is created successfully you will get status 201 and a JSON object as shown in listing 3 in return.

Listing 3: Body of the response of the `POST` request

```
{
    "lat": <lat>,
    "lon":<lon>,
    "id":<id>
}
```

The `id` is the ID of the object created. You will need it to delete the object later, so better save it anywhere. After creating the object, the object is requesting the weather data (current and forecast) every hour. If one of the request body properties is not defined you will get a status 400 in return.

To delete a WeatherCollector object of given ID, you can send an `DELETE` request against `<basepath>/weathercollectors` providing the in body the data shown in listing 4 as content-type `application/json`.

Listing 4: Body of `DELETE` request endpoint

```
{
    "id":<id>
}
```

If `id` is set and there is a WeatherCollector object with such an ID, it will be deleted and therefore stops gathering data. The return will be status 200. If id is not set or there is no WeatherCollector object with such an ID, the return will be status 400.

3.2 Windturbine

3.2.1 Swept area function

The swept area of a windturbine is dependent on the radius of the rotary blades. Combined with the constant Pi the area swept can be computed by the equation: $A = r^2 * \Pi$
Source: $Beckmann, Petr(1976), AHistoryofPi, St.Martin'sGriffin, ISBN978 - 0 - 312 - 38185 - 1$

3.2.2 Vapor pressure

To compute the actual vapor pressure in the air the relative humidity has to be given. The relative humidity is computable with the equation $H = P_{av}/P_s$ with P_{av} being the actual vapor pressure and P_s being the saturated vapor pressure at a given temperature. This equation can be restructured to get the actual vapor pressure: $P_{av} = H * P_s$. The relative humidity is a parameter for the function, which is filled with data form the before mentioned weather data collection.

To be able to now compute the actual vapor pressure at a given temperature we still need the saturated vapor pressure. For this we can use the Herman Wobus equation (E being the vapor pressure):
$E = e/p^8$ with
e = 6.1078 and
$p = c_0 + T * (c_1 + T * (c_2 + T * (c_3 + T * (c_4 + T * (c_5 + T * (c_6 + T * (c_7 + T * c_8 + T * c_9)))))))$
with c_0 to c_9 being constants and T being the temperature in degrees Celsius.
Source: $http://www.emd.dk/files/windpro/WindPRO_AirDensity.pdf$

3.2.3 Density of moist air

In order to compute the density of moist air, we have to have a look at how it is compounded. Moist air density is a mixture of dry air and water vapor. The physical equation for this is:
$D_m = (P_d/(R_d * T_k)) + (P_v/(R_v * T_k))$
D_m is the density of moist air, P_d is the pressure of dry air at the specified temperature, R_d is the gas constant for dry air, P_v is the pressure of water vapor at the specified temperature, R_v is the gas constant for water vapor and T_k is the given temperature in degrees Kelvin. With the previously implemented methods this equation system is able to compute the air density with only the temperature and relative humidity given.
Source: $Shelquist, R(2009)Equations - AirDensityandDensityAltitude$
Source: $http://www.emd.dk/files/windpro/WindPRO_AirDensity.pdf$

3.2.4 Wind turbine model

The wind turbine model contains some variables which can be set by the user. The function for computing the generated energy is derived from the power coefficient of the turbine (basically the efficiency), the size of the turbine (shows in area swept), the density of the air in the area and of course the weather conditions which apply at the moment given. The general equation for this setup is: $P_{avail} = (1/2) * p * A * v^3 * C$ where p is the air pressure, A the area swept, v

the windspeed and C the power coefficient.
Source: $https://www.raeng.org.uk/publications/other/23-wind-turbine$

3.3 Photovoltaic

3.3.1 Temperature loss

The function for the temperature loss is giving a linear function for the percentage loss of energy depending on the degrees over 25 degree Celsius. We made the assumption the efficiency does not go over 100 % even for temperatures below 25 degree Celsius.The resulting equation is as follows: $L_t = max((T - 25) * 0.005, 0)$ with the result being the percentage of energy lost due to temperature as a point number.

3.3.2 Performance Ratio

The performance ration is computed with the equation $1.0 -$ (total percentage of losses). With the previous computed losses for temperature and the constant loss L_0 of 0.14 we get the equation $R = 1.0 - (L_0 + L_t) = 1.0 - (0.14 + L_t)$

3.3.3 Solar irradiance

The computation of the SolarRadiationIncident and therefore the effective Solarradiation on the surface of the solarpanel can be computed with the equation $(SP_{horizontal} * sin(\alpha + \delta))/sin(\alpha)$. α is the effective tilt of solarradiation on a specified latitude and set as 90 - Latitude of the object $+ \delta$. δ is the tilt of the solarradiation in regard of the day of the year d and can be computed with the following equation: $23.45 * sin((360/365) * (284 + d))$ Source: https://pveducation.org/pvcdrom/properties-of-sunlight/solar-radiation-on-a-tilted-surface

3.3.4 Photovoltaic Model

The formula for computing the generated energy as given in the slides is: $E = A * r * PR * S$ A is the area of the solarpanel and is given by the user in m^2. Same goes for the solar panel yield r which is given by the user as well. The performance ratio PR is computed in on of the previous steps and is a percentage. S is the solar radiation incident and is computed in regards of solar radiation and other factors taken from the database. The measurement unit for S is W/m^2. The resulting unit for produced energy is therefore W.

3.3.5 Battery Model

The first function for returning the current state of the battery is a simple return of the currently stored energy. The formula for charging and discharging is more complex and takes several parameters. The parameters are charging and discharging rate as well as charging efficiency. The function returns the actual change of energy stored in the battery. The returned value is positive if the battery is charged and negative if the battery is discharged.

3.3.6 Demand Profile

4 Conclusions

The power girds of the future will be different from the existing ones. More information will be used to make better decisions, more renewable energy sources will be integrated and more automation will happen. Also, microgrids will be a part of the energy concepts of the future. The simulation system presented in this report could help to gain a better understanding about different aspects of microgirds. The first section provided a motivation and introduction to microgirds. The next section contains basic information which are necessary to understand the problem, the functional requirements and an architecture description. The architecture is split in three layers. The webfrontend, the logic layer which contains the simulation and the database layer. It also features a reliable approach to integrate an external weather component.

In the future more custom modules could be developed, to integrate even more suppliers and consumers.

All links were checked last on October 25, 2018.