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 lots 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 and supply. There are many different actors and technologies which are connected to each other and interoperate to optimize the grid.

A smart energy system creates a bridge between a power grid and a resilient and reliable smart grid. Users can simulate different 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. If renewable energy sources produce less energy than is needed, the smart grid can buy energy from the main grid. This process of matching demand and supply can be simulated with our smart energy system.

Using day-ahead energy prices and supply and demand forecasts, the system can simulate demand shifting to be less independent from the main grid. Therefore our smart energy system can optimize supply, demand, battery data in regard to minimal energy costs with demand shifting and charging/discharging batteries.

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. The next section contains the theoretical background to understand the implementation and the description of the basic part of our system. In section 4 we present the mathematical equations the optimizer uses to balance supply and demand in regard to minimal energy costs. The implementation of the solver is also described in this section. Section 5 shortly describes the day-ahead energy market and our service to collect those prices for the dynamic pricing in our system. The frontend implementation is described in section 6. In addition to this, we explain which charts we chose to present the data most conveniently to the user. Finally in section 7, we give a conclusion about the project and present an outlook about possible future work.

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, $1kW = 1000W$ [20, 6, 23].

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) [15, 8].

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 [24]. 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 [18].

$$Demand = Voltage * Current$$

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 [24, 10].

2.2.2 Consumption

It is easier to understand electricity consumption because we are more used to this concept [24]. 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 [24][10]. Most of the time the consumption is measured in kilowatt per hour. The formula to calculate the consumption is the following [24].

$$Consumption = Demand * Time$$

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

2.2.3 Difference

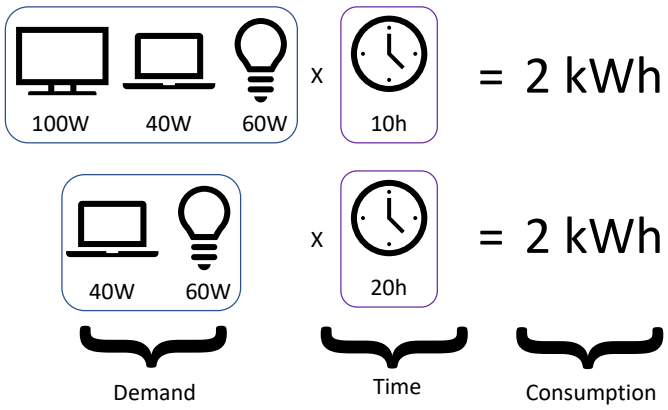

Figure 1: Example for demand and consumption

The demand is the rate of which we use energy and the consumption is the total energy used for a given time frame [24]. 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 create one or more wind-turbines or solar pannels in the frontend view.

24. As a user, I want to be able to modify my supplier attributes, e.g. location to adapt the system on new requirements.

25. As a user, I want to be able to delete a supplier in the web frontend.

26. As a user, I want to create one or more consumer, e.g. hourses or commercial buildings in a frontend view.

27. As a user, I want to be able to modify my consumer attributes, e.g. location to adapt the system on new requirements.

28. As a user, I want to be able to delete a consumer in the web frontend.

29. As a user, I want to create one or more batteries in a frontend view.

30. As a user, I want to be able to modify my battery attributes, e.g. location to adapt the system on new requirements.

31. As a user, I want to be able to delete a battery in the web frontend.

32. As a user, I want to have a compact dashboard to see several kinds of charts with supply and demand.

33. As a user, I want to be able to select data sources for my chart in a simple manner, so that I can analyse my system better.

34. 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.

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

36. 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.

37. 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.

38. 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.

39. As a user I want to see day-ahead prices for my bidding zone so that I can how much energy from the main grid will cost.

40. As a user I want the system to optimize the supply, demand and battery charge/discharge in regard to energy prices to balance supply and demand.

41. As a user I want the system to do some demand shifting to always balance most possible the supply-demand-balance.

42. As a user I want the system to use day-ahead prices to do some dynamic pricing on decision making for the demand shifting.

2.4 Architecture

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

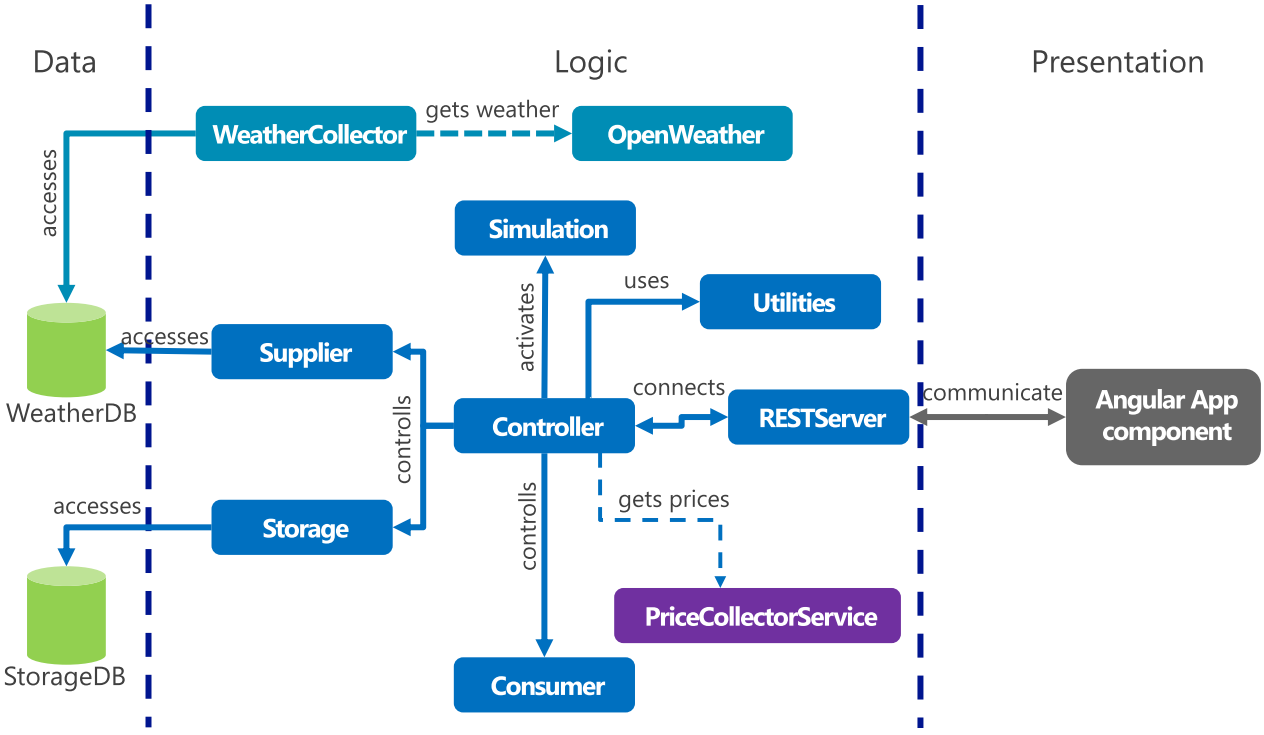

Figure 2: Smart energy system architecture

The `WeatherCollector` component is responsible for gathering actual weather data from *weatherbit* 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 weatherbit 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. There is also a dashboard, where the user can see several charts to analyse the supply, demand and battery status.

The controller of the main system gets day-ahead prices from a `PriceCollectorService`. This service gets day-ahead prices from *ENTSO-E* for bidding zone DE-LU. With the day-ahead prices, supply data, demand data and battery charge and discharge data, the controller can optimize to balance the energy in regard to minimal energy costs, as described in section 4.

3 Implementation

The current implementation consists of two independent parts. The weather collector component which is described in section 3.1 is developed with NodeJS express. The main component which handles all the simulation and calculations is written in Java and uses the Spring Boot framework. The theoretical foundation of this component is described in section 3.2 and the following. Both components are linked through a shared database.

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 an hourly basis. There is one weather service which fulfills our needs more than appropriate. The weather service `weatherbit.io` has different kinds of endpoints on its API. In our case we use three of them, the current weather data API, historical weather data 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. For the historical weather data a user has to specify a start and end date.

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


```
{
    "data": [
        {
            "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, `ghi` for global horizontal solar irradiance, `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 the metric format you have to add an additional query parameter `unit=m`. Every request must provide an API key in query parameters[25]. This API key can be requested by creating an account on weatherbit.io. After creating an account a user has 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 an 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.

For the historical weather data API we have to use a little trick. The free plan allowes only to request the weather history of one day. Since we need the past 5 days in our system, we send 5 seperate requests for each day to the API, which allows us to gather the weather history of all 5 days. Afterwards we have to concatenate the results. The basepath for the historical weather data is `https://api.weatherbit.io/v2.0/history/hourly` with HTTP verb `GET`. There are the same basic query parameters as in the API for the current weather data. As additional query parameters the user has to specify a `start_date` and `end_date` in form YYYY-MM-DD:HH.

3.1.2 Our WeatherCollector service

To gather weather data for different locations and store it into a 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 and historical weather data on an 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. For the historical data we do the same. As described in the section above the WeatherCollector requests the historical weather API 5 times, one per day and stores the data concatenated in the database. To ease the main component a bit, we

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

store also a concatenated version of historical data and forecast data in a seperate table in our database, so the main component has an easy way to get the last 5 days and one day forecast together. 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 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 dependency 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>"`. Keep 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
- `collect historical`: Collects the last 5 days of weather history for a given location and stores it into the database.
- `get historical`: Gets the latest version of the historical weather data of a given location
- `collect total`: Collects a concatenated version of historical weather data and weather forecast for a given location and stores it into the database.
- `get total`: Gets the latest version of the concatenated version of historical weather data and weather forecast for a given location
- `start`: Starts a WeatherCollector object on hourly picking new weather data (current, forecast and historical) 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 wind turbine is dependent on the radius of the rotary blades. Combined with the constant Pi the area swept can be computed by the equation[5]: $A = r^2 * \Pi$

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 from 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[1, 16].

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[1, 22].

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 [11].

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 that 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 ratio is sum of all losses. With the previous computed losses for temperature and the constant loss L_0 of 0.14 we get the equation $R = (L_0 + L_t) = 1.0 - (0.14 + L_t)$. To use the result in our calculation we have to subtract the result form 1.

3.3.3 Solar irradiance

The computation of the solar radiation incident and therefore the effective solar radiation on the surface of the solar panel can be computed with the equation $(SP_{horizontal} * sin(\alpha+\beta))/sin(\alpha)$. α is the effective tilt of solar radiation on a specified latitude and set as $90-latitude$ of the position from the solar panel $+ \delta$. δ is the tilt of the solar radiation in regard of the day of the year d and can be computed with the following equation: $23.45 * sin((360/365) * (284 + d))$. ($SP_{horizontal}$ is the horizontal solar radiation. Our weather service provides us with this information. The tilt of the solar panel itself is used as β [19].

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 solar panel and is given by the user in m^2. Same goes for the solar panel yield r which is given by the user via the maximum power yield 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

Our demand profile is based on data from `https://open-power-system-data.org/`. We use the „Household Data“ package from 2017-11-10. The calculations are based on the 60 minutes interval data set. We use two different profiles, one is based on the feed *DE_KN_residential2_grid_import* and the other is based on *DE_KN_industrial3_grid_import*. The raw data from Open Power System Data always report the total amount of energy used so far in kWh. Based on this we needed to calculate the difference between the data points to the get change of the total value per hour. After that, we calculated the average value for each hour of the day for all data points. The consumer in profile 1 used in total about 4500 kWh in the time between the 15. April 2015 and the 1. February 2017. This equals roughly a total of 2500 kWh per year. The second consumer used in total 695000 kWh in the time between 11. February 2016 and 09. February 2017. Our electricity provider estimates a small one or two people household for the first consumer based on the total usage per year [9]. Based on this we amused a floor area size of 50 m^2. Figure 3 and figure 4 visualize the demand profiles. The user does not only provide an area but also an average daily occupancy. We assumed that the latter value is the fraction of the time in which the building is in use. Based on this we calculate the final energy demand D for each hour for a building with this formula $D = profileDemandPerSquareMeter_{hourOfTheDay} * floorAreaSize * averageDailyOccupancy$.

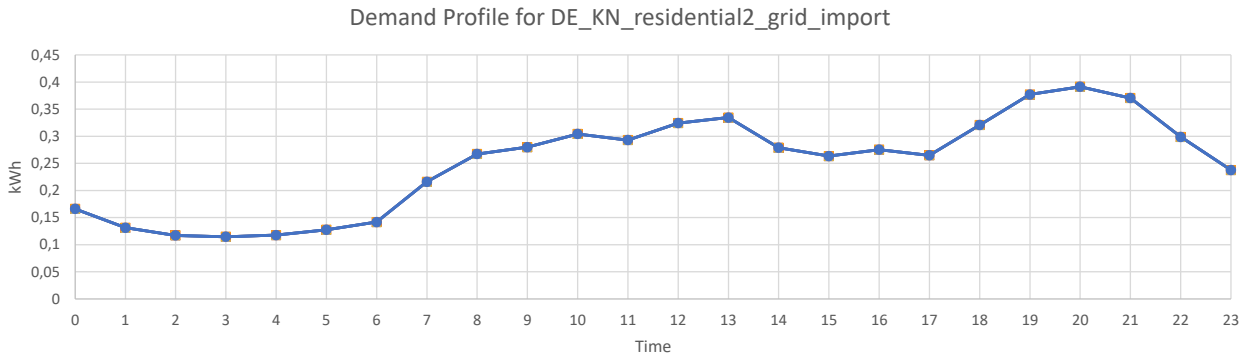

Figure 3: Demand profile 1: Home building

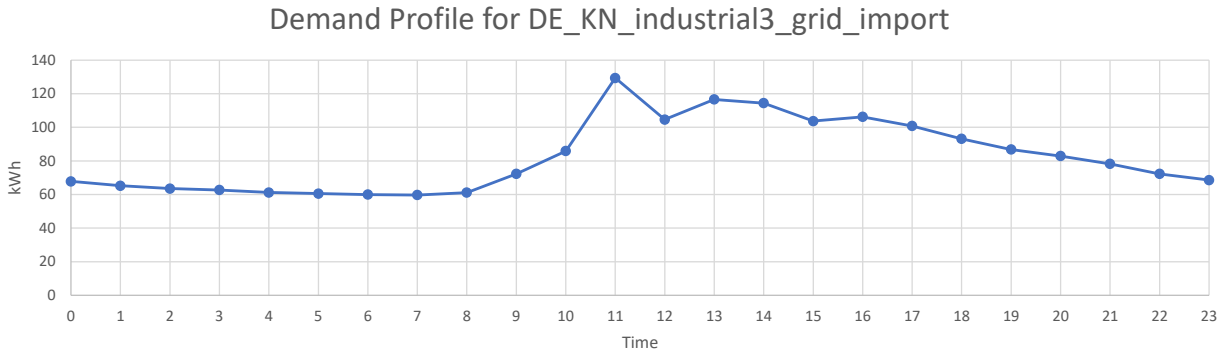

Figure 4: Demand profile 2: Office building

4 Demand Response and coordination

4.1 Balance within the micro grid

To guarantee the balance in the micro grid as defined in the slides we have to formulate a mathematical equation to use as a constraint. We came up with the following equation:

$$\begin{aligned}\forall t \in T : \sum_{s \in S} P_s(t) + \sum_{b \in B} P_b(t) + P_g(t) = \sum_{h \in H} (P_h(t) + PposShift_h(t) - PnegShift_h(t)) \\ + \sum_{c \in C} (P_c(t) + PposShift_c(t) - PnegShift_c(t)) + \sum_{b \in B} Pd_b(t)\end{aligned} \tag{1}$$

As one can see, for all time steps t in the set of all time steps T the equation has to hold true. This is the constraint for the whole micro grid. The left side of the equation represents the power supplied by the supplier ($P_s(t)$), the batteries ($P_b(t)$) and the import or export from the main grid ($P_g(t)$), the right side represents the consumers, which contains homes ($P_h(t)$) and commercial buildings ($P_c(t)$), the batteries ($Pd_b(t)$) while charging. On the right side, there are also the demand shift values for homes and commercial buildings.

In reality, if the balance between the consumers and the suppliers is disturbed, severe consequences can happen. Any imbalance results in a change in the frequency. This change may

force more suppliers to stop production because most suppliers can only operate in a narrow frequency range. As a result of this the imbalance ingresses even more. If the change in the frequency exceeds a certain threshold suppliers and consumers still connected to the grid could be damaged.

The supply side of the equation itself is compartmentalized into different factors which are explained in the following.

Wind turbines and solar panels: On the supply side we have the sum which contains the power provided in time step t by all suppliers of the system ($P_s(t)$), regardless of their type (wind turbines or solar panels). We get those values from our model given data for the time step and weather conditions. Our software calculates a demand forecast and a supply forecast based on models of the supplier and consumers. Additionally, the location of the installations are provided by the user and a weather forecast is provided by a weather service. For example, for the wind turbines, the Herman Wobus equation is used to calculate the saturated vapor pressure [16]. This equation is based on empirical data and helps us to incorporate the weather forecast in the wind supply forecast. Because all these calculations happen before the optimization step, the solver does not need to know anything about the specifics of the supplier or consumer. As a result of this, we don't need any constraints for the consumer or suppliers. This supports reuse-ability of the solver and the models for it. The forecast data is provided as an array of data which contains a value for each time step. The values represent the supply or demand and are measured in $Watts$.

Battery supply: The second part is the power provided by the sum of all batteries at a given time step t. Each battery has a fill level for each time step. It holds the information how much charge is stored in the battery at each step. This results in the following constraints. Additionally, there is a limit on the charge and the discharge rate. Also there is a limit on the fillLevel, because batteries can not hold an infinite amount of energy.

$$\forall b \in B, \forall t \in T : P_b(t) < maxDischargeRate_b \tag{2}$$

$$\forall b \in B, \forall t \in T : fillLevel_b(t+1) = fillLevel_b(t) - Pd_b(t) * 1h \tag{3}$$

$$\forall b \in B, \forall t \in T : fillLevel_b(t) \leq maxFillLevel_b \tag{4}$$

Import and export to the main grid: The connection to the main grid is resolved in the third part of the supply side of the equation. If we have more demand than supply, the solver will either use the energy stored in the batteries or import energy from the main grid. While the demand is satisfied by the suppliers and the battery, no further energy import is necessary. But if the batteries are empty and the demand exceeds the supply more energy has to be imported from the main grid. The import of energy from the main grid costs money. If the supply exceeds the demand it is possible to charge the batteries or to export the energy to the main grid. Exporting energy to the main grid provides revenue.

For the demand side, we chose a similar approach:

Demand from homes and commercial buildings: The first one is the sum over all homes of a given time step t with related demand shift of every single home to calculate the total demand of all homes. The second one is the sum over all commercial buildings of a given time step t with related demand shift of every single commercial building to calculate the total demand of all commercial buildings. Comparable to the supply forecast, the demand

forecast is generated before the optimization step. It is based on the demand profiles described in section 3.3.6. The demand of the homes is provided as an array of constants to the solver and measured in $Watts$. Because the supply and the demand are provided as constants to the optimization component, we did not formulate any constraints for the supplier or the consumers. For the demand shift parts of the demand entities, we have a positive shift and a negative shift. The negative demand shift tells us how much demand is shifted away on this time step. The positive demand shift tells us the opposite, how much demand is shifted to this time step. So the negative demand shift will decrease the demand while the positive demand shift will increase the demand. If the negative demand shift of a home or commercial building is greater than 0, the positive demand shift must stay 0 and vice versa. Additionally, overall the demand has to stay the same. This results in the following constraints:

$$\begin{aligned} \forall l \in H \cup C, \forall t \in T : (PposShift_l(t) > 0 \veebar PnegShift_l(t) > 0) \\ \veebar (PposShift_l(t) = 0 \wedge PnegShift_l(t) = 0) \end{aligned} \tag{5}$$

For all consumers this constraint has to hold true:

$$\forall t \in T : \sum_{t \in T} PposShift(t) = \sum_{t \in T} PnegShift(t) \tag{6}$$

We need to introduce an upper bound for the negative shifting which a user can provide for the house or commercial building demand, e.g. 30%. Additionally, shifts can only happen in blocks. It is only possible to shift the maximal value or noting. The constraint is shown in Eq. (7).

$$\forall l \in H \cup C, \forall t \in T : \left(PnegShift_l(t) = \frac{P_l(t) * n_l}{100} \right) \veebar (PnegShift_l(t) = 0) \tag{7}$$

For this constraint n is provided by the user.
There are two possibilities for shifting demand which can be chosen by a user. The first one is used if the user wants to save calculation time or has limited RAM. This variant allows only one shift per supplier. For example, 30% of the load form 9 o'clock to 10 o'clock or from 15 o'clock to 20 o'clock, but not both. This variant uses the constraint form Eq. (8). The second variant allows shifting form every hour to every other hour as often as it helps to improve the objective. This results depending on the input and the number of time steps in a very long run-time and a high RAM consumption.

$$\forall l \in H \cup C : numberOfShifts_l \leq 1 \tag{8}$$

Demand of batteries: The third part calculates the demand of all batteries at a given time step t. If we have more supply than demand, the solver can either export the excess energy to the main grid or store it in a battery for later use. The storage of energy in the battery results a loss of energy, because charging is not 100% efficient. The *chargeEfficiency* is provided by the user. These constraints have to be added to our model.

$$\forall b \in B, \forall t \in T : Pd_b(t) < maxChargeRate_b \tag{9}$$

$$\forall b \in B, \forall t \in T : fillLevel_b(t+1) = fillLevel_b(t) - \left(\frac{Pd_b(t) * chargeEfficiency_b * 1h}{100} \right) \tag{10}$$

4.1.1 Additional Constraints

There are additional constraints which have to hold true in our simulation.

We don't want the batteries charge and discharge at the same time. So, we need a constraint to prohibit this kind of behavior as shown in Eq. (11)

$$\forall b \in B, \forall t \in T : (P_b(t) = 0 \wedge Pd_b(t) = 0) \veebar (P_b(t) > 0 \veebar Pd_b(t) > 0) \tag{11}$$

The most other calculations like the computation of the sums is implemented as constraints to.

4.2 Objective of the optimization problem

We import or export energy from the main grid with $P_g(t)$. If $P_g(t) < 0$ than energy will be exported since the overall supply gets less than only with the batteries and renewable supplier. If $P_g(t) > 0$ than energy will be imported since the overall supply gets more than only with the batteries and renewable supplier. We can use this to compute our profit or loss since we want the optimizer to maximize the profit. To compute profit and loss we declared two variables $isEnergyImport$ and $isEnergyExport$ as shown in Eq. (12).

$$\begin{aligned} P_g(t) < 0 &\Rightarrow isEnergyImport(t) \\ P_g(t) > 0 &\Rightarrow isEnergyExport(t) \end{aligned} \tag{12}$$

If $isEnergyImport$ is true, the optimizer will use the absolute value of P_g for the revenue calculation. In this case, the export will be zero for timestep t. If $isEnergyExport$ is true, it's the other way round.

To maximize the profit in a specified interval of time steps $t \in T$ between the time steps $a \in T$ and $b \in T$ we propose the following function:

$$\max_{p \in P} (\sum_{t=a}^{b} ((E(t) * price(t)) - (I(t) * cost(t)))) \tag{13}$$

P is th set of all possible decision configurations. $E(t)$ contains the energy export ($isEnergyExport$ is true) of time step t and $I(t)$ ($isEnergyImport$ is true) contains the energy import of time step t. The value of these variables are depending on the decisions made by the optimizer on the equation for the balance in the micro grid of the exercises before. All configurations from P have to satisfy all the constraints. The formula consists of two main parts. The minuend in the sum calculates the profit for exporting energy to the main grid for this time step. For this calculation, the total energy export $E(t)$ for this time step is multiplied by the price for the energy. The subtrahend in the sum expresses the cost for importing energy from the main grid. It consists of the total imported energy $I(t)$ multiplied by the cost for importing energy at this time step. There are different possibilities to improve the total profit. For example, one could sell energy at a high price. Another possibility is to store energy if the price for buying is low. For our simulation we use time steps with the width of one hour, so the values for power and energy are the same, but $E(t)$ and $I(t)$ are measured in Wh while the unit of the most other values of the previous equations is $Watt$. The price for exporting energy and the cost for importing energy are provided as an array of constants. This solution maximizes the total profit at all costs. Usually, this means it also minimizes the import from the main grid, because importing energy results in costs. But sometimes it can happen that importing energy

at a low price, storing it in batteries and selling it later is the most profitable solution. We chose to optimize for the most profit but sometimes other objectives could be important too. Maybe one wants to go easy on his batteries or wants to be as independent of the main grid as possible. To enable this variant we propose two solutions. The first one computes a pareto front with the two objectives *maximal profit* and *minimal import*. All possible solutions are presented to the user. The other way is to prohibit that the battery gets charged if the import from the main grid is not zero. This constraint would realize this $Pg(t) > 0 \Rightarrow Pdb(t) = 0$. In this case, the optimizer is only allowed to import energy from the main grid to balance the demand, not to charge the battery. Both variants reduce the profit to be more independent from the main grid. We implemented the most profitable solution.

4.3 Implementation

To implement the constraints, we use the library choco solver[2]. This library helps us to solve the optimization problem for integer variables. As a consequence of this only integer variables are possible and double variables have to be rounded. As described above we would like to use Watts as the unit of measurements for our variables. But this would result in large numbers and a huge range for the variables. To circumvent this, we convert everything to kW and kWh. This helps to reduce the range for each variable and therefore speeds up the solving process. The bounds for the variables are configurable by the user but should remain as low as the specific problem allows. Otherwise, the calculation time and the RAM requirements are very high. To make the calculation of an optimal solution feasible it was necessary to limit the number of consumers to two. This could be one household and one office. Furthermore, only one battery and one supplier is supported. A workaround to these limitations is to combine the loads of different consumers and to combine the supply of different suppliers. For the suppliers there is no difference in the result if all of them get combined before the optimization. Another limitation is that the computation of more than 4 timesteps requires a lot of spare time and depending on the input and the chosen range a lot of RAM.

4.4 Nomenclature Table

Symbol	Description	Unit
T	The set of all time steps. The step width is 1h.	Hour
t	A single step from the set T.	Hour
S	The set of all suppliers (Different wind turbines, solar panels...).	
s	A single supplier from the set S.	
$P_s(t)$	The supply from a single supplier for the specific time step t.	Watt
B	The set of all batteries.	
b	A single battery.	

[2] `http://www.choco-solver.org/`

$P_s(t)$	The supply from a single battery for the specific time step t.	Watt
$P_g(t)$	The supply from the main grid for the specific time step t.	Watt
H	The set of all homes.	
h	A single home from the set H.	
$P_h(t)$	The demand for a single home at a specific time step t.	Watt
C	The set of all commercial buildings.	
c	A single commercial building from the set C.	
$P_c(t)$	The demand for a single commercial building at a specific time step t.	Watt
$P_b(t)$	The supply from a single battery at a specific time step t.	Watt
$Pd_b(t)$	The demand from a single battery at a specific time step t.	Watt
$PposShift_h(t)$ & $PposShift_c(t)$	The demand flexibility (in positive direction) for the consumer at specific time step t.	Watt
$PnegShift_h(t)$ & $PnegShift_c(t)$	The demand flexibility (in negative direction) for the consumer at specific time step t.	Watt
$maxDischargeRate_b$	The maximal rate at which it is possible to discharge the battery b.	Watt
$fillLevelb(t)$	Represents the amount of energy stored in battery b at the time step t.	Wh
l	A single consumer from the set $H \cup C$.	
$P_l(t)$	The energy demand from a single consumer from the set $H \cup C$ at a specific time step t.	Watt
n_l	A user defined number which defines how much percent of the demand of the consumer l can be shifted.	1
$numberOfShifts_l$	The number of occurred shifts for one consumer l.	1
$maxChargeRate_b$	The maximal rate at which it is possible to discharge the battery b.	Watt
$maxFillLevel_b$	The maximal charge the battery b is able to store.	Wh
$chargeEfficiency_b$	The user defined efficiency for charging the battery b.	1

$isEnergyImport(t)$	Boolean variable which is true if energy is imported at a specific time step t.	
$isEnergyExport(t)$	Boolean variable which is true if energy is exported at a specific time step t.	
P	The set of all possible configurations.	
p	A single configuration.	
a	A single step form the set T. It marks the first time step of the optimization.	Hour
b	A single step form the set T. It marks the last time step of the optimization.	Hour
E(t)	The total energy exported to the main grid for a specific time step t.	Wh
I(t)	The total energy imported from the main grid for a specific time step t.	Wh
price(t)	The function provides the price for energy per Wh which is sold to the main grid for a specific time step t.	Cent per Wh
cost(t)	The function provides the cost for energy per Wh which is imported from the main grid for a specific time step t.	Cent per W

Table 1: Describes every Symbol

5 Dynamic pricing

This section contains information about the dynamic pricing component. The component fetches day-ahead prices for the German market and provides these information to the main system.

5.1 Day-ahead market service

We considered multiple different sources for the price information. The following section contains the pros and cons of each source.

5.1.1 epexspot.com

EPEX SPOT provides the day-ahead prices among others for Germany and Luxembourg. The information is directly embedded in the HTML code of the website and does not require JavaScript. As a result of this one could download the website easily, only the parsing part would be harder. Additionally, most changes on the website could lead to adjustments in the parsing code. Moreover, they provide an API but it is expensive to use and the terms and

Pro	Contra
• Website moderate difficult to parse • Lot of useful explanations	• API not free of charge [21] • Not clear if the terms and conditions allow parsing [21]

Table 2: Pros and cons of epexspot.com

conditions are not clear if it is allowed to parse the site. Section 5.1.1 provides an overview of the pros and cons.

5.1.2 nordpoolgroup.com

Pro	Contra
• Lot of Nordic countries	• No day-ahead prices for Germany [2] • You need to be a customer to use the API [2] • Harder to parse the website • Automatic data extraction not allowed in the terms and conditions [4]

Table 3: Pros and cons of nordpoolgroup.com

Nord Pool AS provides day-ahead prices for a lot of Nordic and other countries [3]. Data is provided for example for the UK, Norway or Sweden. But Germany is not included. Additionally, the website does not directly contain the data and requires JavaScript to fetch them. So parsing is harder and not allowed in the terms and conditions [4]. They have an API but it is only available to customers [2]. Table 3 provides an overview of the pros and cons of Nord Pool AS as a data source.

5.1.3 smard.de

Pro	Contra
• Data licensed under CC BY 4.0 [7] • Fast support	• No API • Support discourages parsing of the website

Table 4: Pros and cons of smard.de

Smard.de is provided by the German Bundesnetzagentur. They provide data for Germany and Luxembourg (combined bidding zone). They provide the data licensed under CC BY 4.0 [7]. Additionally, they have a fast and friendly support. But they do not offer an API and the support said that they do not want automatic data extraction on the website. So we did not check if it is feasible. See Table 4 for a short recap.

5.1.4 dataminer2.pjm.com

Pro	Contra
• API available [17]	• Does not provide data for Germany [17] • API requires an account[17]

Table 5: Pros and cons of dataminer2.pjm.com

The Data Miner 2 from PJM provides an API but it does not provide data for Germany [17]. So we did not investigate this data source any further. As always Table 5 contains the pros and cons.

5.1.5 transparency.entsoe.eu

Pro	Contra
• Free API [14] • Has data for Germany • Terms and conditions allow free usage of data [13] • Fast and helpful support [13]	• API parameters use non standard formats

Table 6: Pros and cons of transparency.entsoe.eu

ENTSO-E provides the day-ahead prices for a lot of countries. Germany is among them. Additionally, the Transparency Platform provides a free REST API with documentation [14]. Some of the parameters of the API use non-standard formats, but it is still usable. The data provided by the API can be used freely [13]. We also had contact with the support and the replies were fast and friendly. Table 6 summarizes the pros and cons.

5.2 Day-ahead market service conclusion

We reviewed five data sources and only one provides data for Germany and has a free API. As a result of this, we decided that we use the service from ENTSO-E (Section 5.1.5). You have to request a security token from their support, but they replay fast. The data return by this API are the prices for Germany and Luxembourg because this is a combined bidding zone.

5.3 Price collector component

To fetch the data, we implemented an additional component which communicates with the API from ENTSO-E and manages database to store the data. Figure 5 contains an overview of the components used in your price collector. The price collector offers a REST Interface which uses JSON for the communication with the rest of our system. It communicates via an XML Format with the ENTSO-E REST Interface. It reads the data items from the API, filters them for the for us relevant data and saves them in an H2 Database. The necessary security token, the

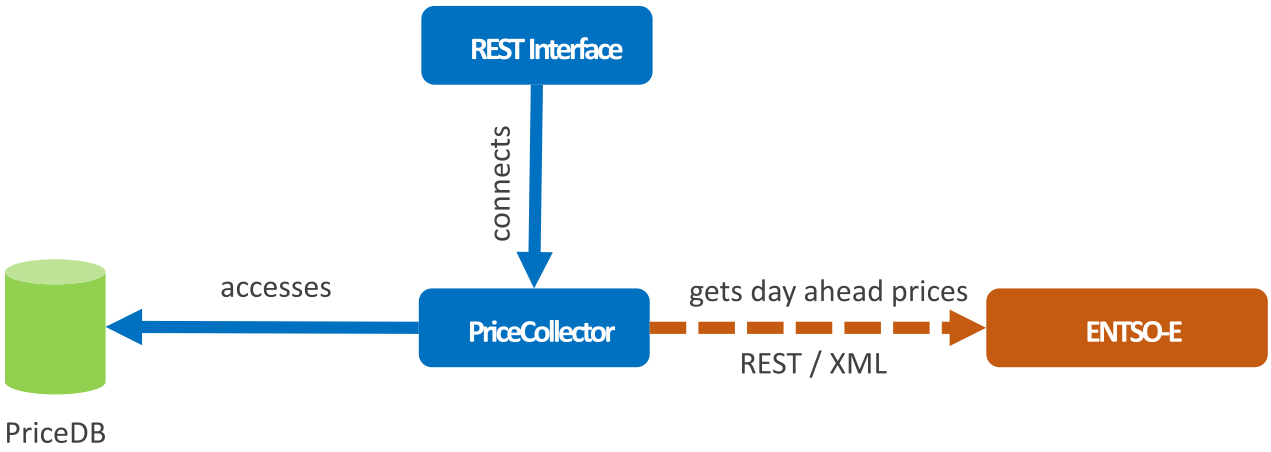

Figure 5: Components: Price Collector

bidding zone and the port for the interface can be configured in a configuration file. The price collector has an auto-fetch feature which is also configurable and allows to request the ENTSO-E API in a user-defined pattern. For example, the day-ahead prices are published between 12 pm and 1 pm (one hour after gate closure) [12]. So it is possible to update the database each day at 1 pm. But also, more or less frequent patterns are possible. For example, each second hour. The price collector always tries to update its data with each request. If the ENTSO-E API is not available or it does not provide data for a specific period, the price collector uses older but still the most recent data from the database to fill the gaps. For example, if you request data for tomorrow at 4 pm and there is no data available from ENTSO-E the price collector will return the data from today at 4 pm. Such old data will be marked with a boolean flag in the response.

We also updated your main architecture to reflect the new requirements. Figure 2 shows the new architecture with the price collector. The database from the price collector is omitted in this figure. The Controller communicates via HTTP with the REST interface of the price collector. The price collector only offers one endpoint which is „/prices“. To make a request the start time and the end time of the required period have to be provided via query parameters. For all time-related information, the time zone UTC is used.

Figure 6 contains an example snippet from an answer of our service. To use this answer our main system has to convert the information into the correct units. As always we provide a swagger documentation in the running competent.

With this component, our system can simulate a micro grid and use real price data. We also added a test script to our repository to test the whole system.

```
[
  {
    "time": "2019-01-07T00:00+0000",
    "priceInEuroPerMWh": 47.97999954223633,
    "oldValue": false
  },
  {
    "time": "2019-01-07T01:00+0000",
    "priceInEuroPerMWh": 47.84000015258789,
    "oldValue": false
  },
  {
    "time": "2019-01-07T02:00+0000",
    "priceInEuroPerMWh": 46.11000061035156,
    "oldValue": false
  }, ...

]
```


Figure 6: Example response snippet

6 User Interface

The following section describes our user frontend and especially how we show data to the user in a compact manner. The frontend is written in Angular[3] with Typescript using Angular Material[4], Chart.js[5], Moments.js[6] and the C3.js[7] library. Angular Material is used to get a consistent look for all view and Moments.js provides us with better time and date handling capabilities. Chart.js features beautiful line charts and C3.js is used to display the bar chart for our price overview. Of course we also use many smaller dependencies which can be found in the respecting dependency file. We divided the front-end info five parts. The user can choose in a navigation bar between *Dashboard*, *Solver*, *Supplier*, *Consumer*, *Batteries* and *Prices* view. The views are described in the following subsections.

6.1 Supplier and Consumer

The run a simulation In the supplier view the user can create, delete or manage supplier. Creating, delete or manage consumer works equivalent so we will explain it only for suppliers.

The interface for suppliers is shown in figure 7. On the left side there is a navigation bar which shows all created supplier. A user can add a new one using the *plus* button under the suppliers, delete a specific supplier using the *bin* button right to the supplier or manage the supplier by clicking on it. The main part of this page is the form with the user given data such as latitude, longitude or additional supplier specific needed data, e.g. rotator radius for wind turbines. The kind of supplier can be chosen with the radio button above the form. Due to Angulars modularity, it is easy to add more kinds of supplier in the future. Next to the form is a map using the Angular Leaflet[8] openstreetmap API. The nice part is that a user does

[3] https://angular.io/
[4] https://material.angular.io/
[5] https://www.chartjs.org/
[6] https://momentjs.com/
[7] https://c3js.org/
[8] https://leafletjs.com/

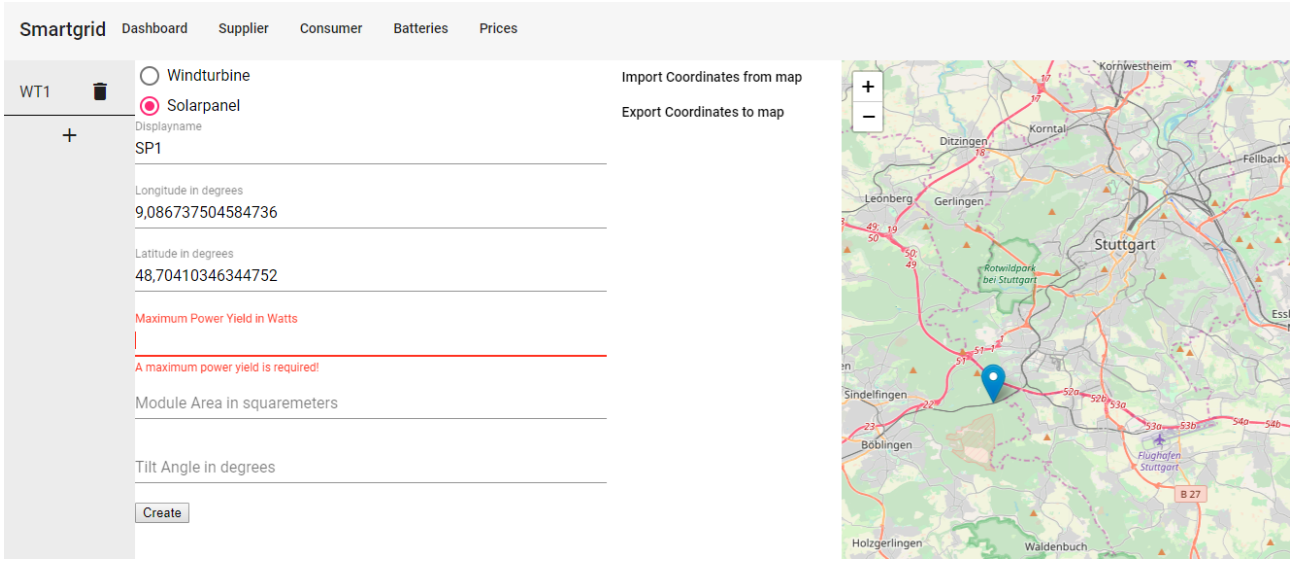

Figure 7: View to create, delete or manage suppliers

not need to know the specific latitude or longitude to create supplier. Clicking on the map adds a location point which can be exported easily to the form which allows the user to create suppliers at a specific place in the map. To support the user even more, required fields which are left blank are highlighted. This is shown in figure 7 for the maximal power yield. Once created a supplier, it is available in the navigation bar at the right. Clicking on it shows all data already filled in the form and allows the user to change it conveniently.

6.2 Batteries

The batteries view is shown in figure 8. It shares a lot of the functionality with the suppliers view to provide a consistent user experience. On the left side of the page, there is the same kind of navigation bar we also used in the supplier and consumer pages to create, delete and manage batteries. In the middle of the view, there is also a form to set the batteries data. The user can specify battery capacity, charge, discharge rates and efficiency as well as it's position. The position can either set directly through the form or just like in the supplier or consumer view through the map using the export button. After pressing the create button the view automatically updates to reflect the changes and to provide the user with feedback for his action. This is also true for suppliers and consumers.

6.3 Prices

The prices view shows the energy prices for a preconfigured bidding zone as table and bar chart. The prices originate form ENTSO-E. The user can specify the start and end date of the prices request, so he can take a look at the day-ahead prices as well as the past 5 days of the energy prices. This process is supported by a date and time picker component to allow a comfortable user experience. The page is shown in figure 9.

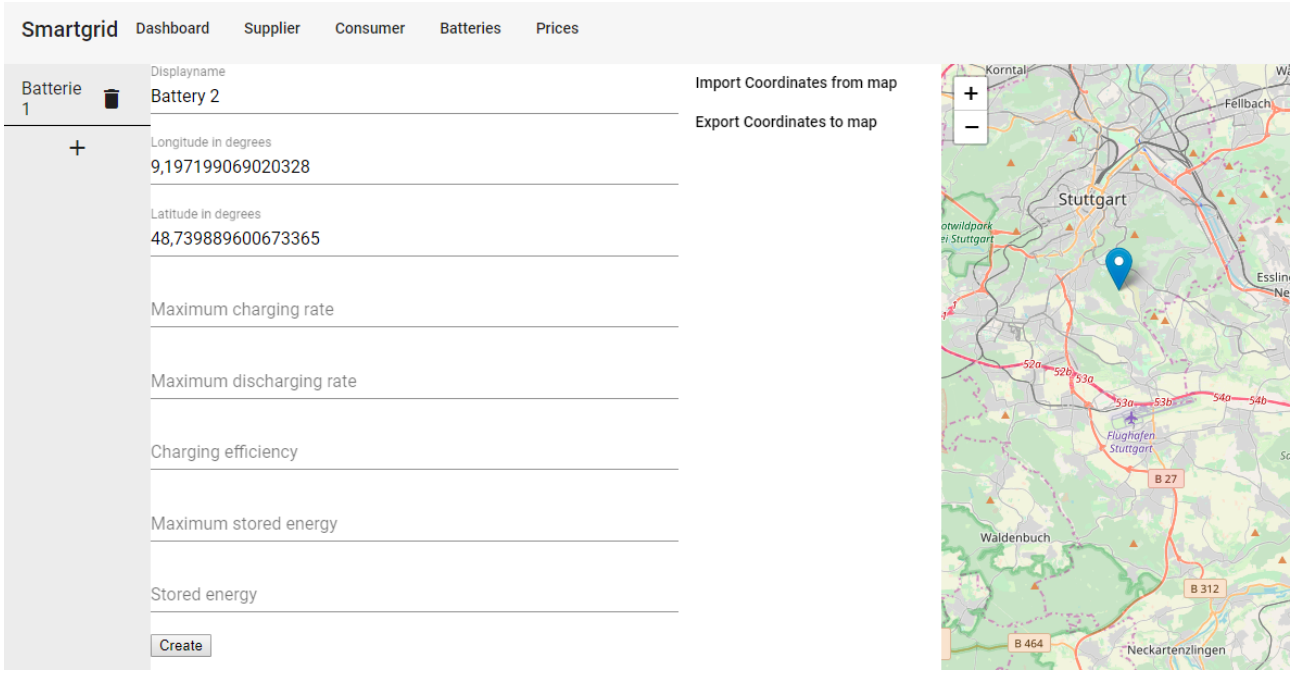

Figure 8: View to create, delete or manage batteries

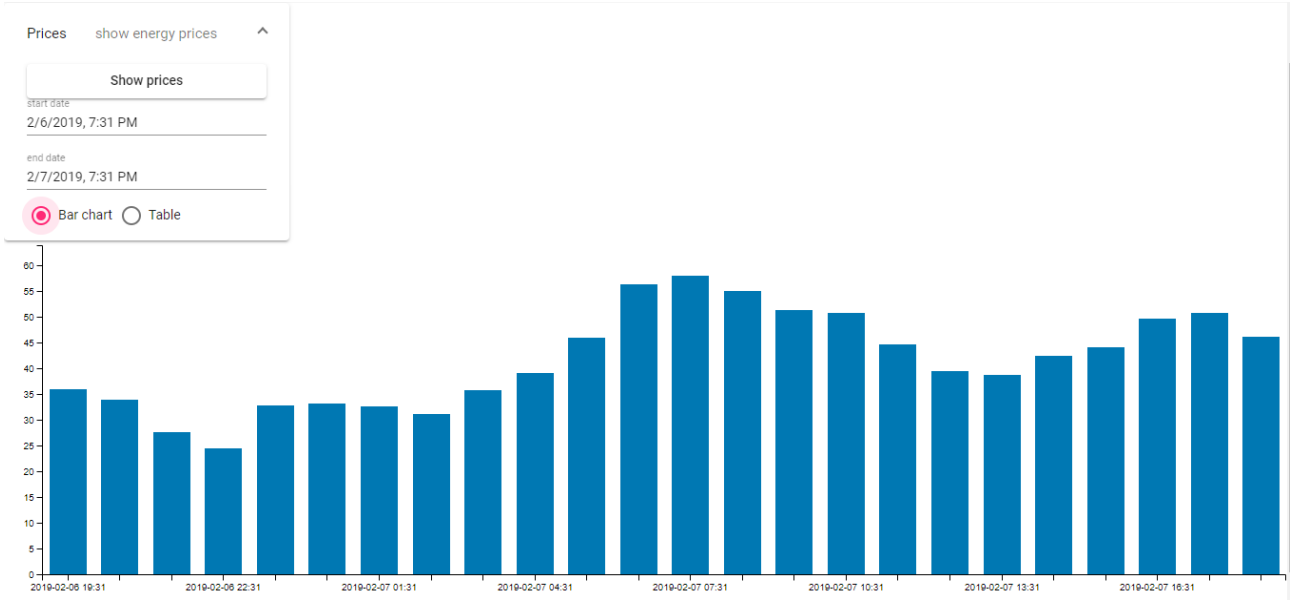

Figure 9: View show energy prices for specific time interval

6.4 Dashboard

The most interesting view is the dashboard. In this view, the user can take a look easily at different kinds of information in a compact manner. The view shows supply, demand and other information in a chart, as shown in figure 10 Clicking on the labels in the legend (right part of the chart) allows the user to disable or enable data in the chart. This makes the chart more clear and allows the user to concentrate better on the information he is interested in. Figure 11 shows a chart containing only the consumers.

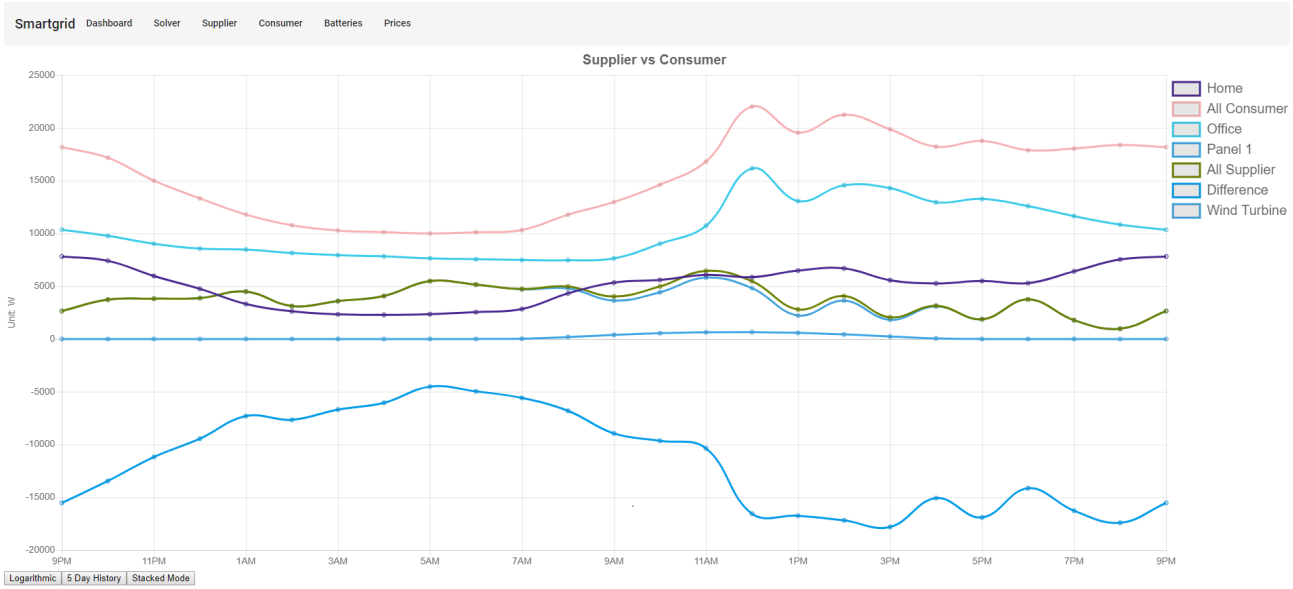

Figure 10: Dashboard view to show the charts

If a user is interested only in the energy produced by the supplier, he can select only the supplier as shown in figure 12.

Using the buttons under the chart switches between several charts. We provide a logarithmic view or normal view. In addition to this, we can show the user 5 days history or the forecast.

Sometimes it is useful to see a stacked view of the data. For this reason, we chose to support stacked charts and normal charts. In stacked charts, the values can be stacked above each others. In figure 12 a stacked chart is used, where the supply of *Panel 1* is added above the supply of *Wind Tubine 1*. Figure 10 shows a normal chart.

6.5 Solver

Our system is made to allow a user to simulate a micro-grid for a period of time. The user should be able to configure a simulation and then post it to the system. After the simulation is complete the results are shown to the user. This could be used to understand the dynamics of a micro-grid. To start a simulation the user has to provide some information. Figure 13 shows the configuration page for a simulation. First the user has to enter the start and end time. There is also a date and time picker to ease the user interaction. For the simulation part more information is necessary. Prices for importing energy via the main grid are provided by the price collector. But it is also possible to sell energy. Therefore, the user has to specify the price which he gets for exporting energy to the main grid. It is also possible to start with a filled battery. For this option a fill level can provided. The last group of input information are specific to the solver. To reduce the time which it takes to simulate the micro-grid all values are bound to a range. This range Can be specified but it should be used with caution, because it can increase the run time and the memory consumption enormously. The specific increase depends on the input values and the other settings. The next setting the number of steps which should be calculated. Higher settings increase run time and memory consumption. The last setting is the timeout time. In most instances the solver finds a good or good enough solution quit fast, but it runs for a long time to find a better one. With this setting a user can

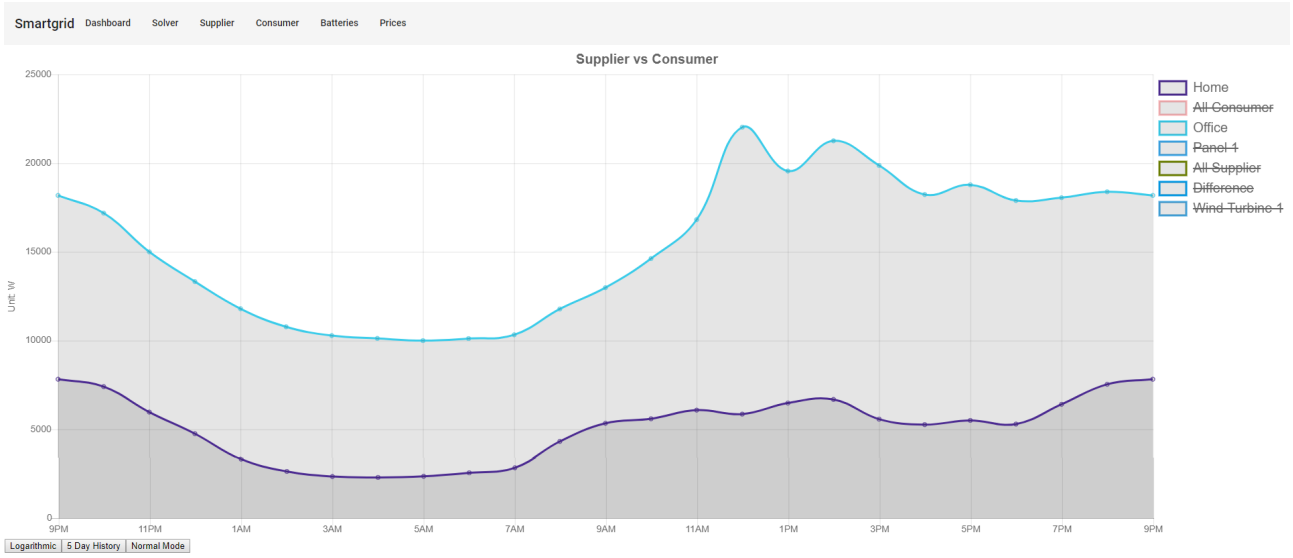

Figure 11: Chart showing only the energy demand of the consumers

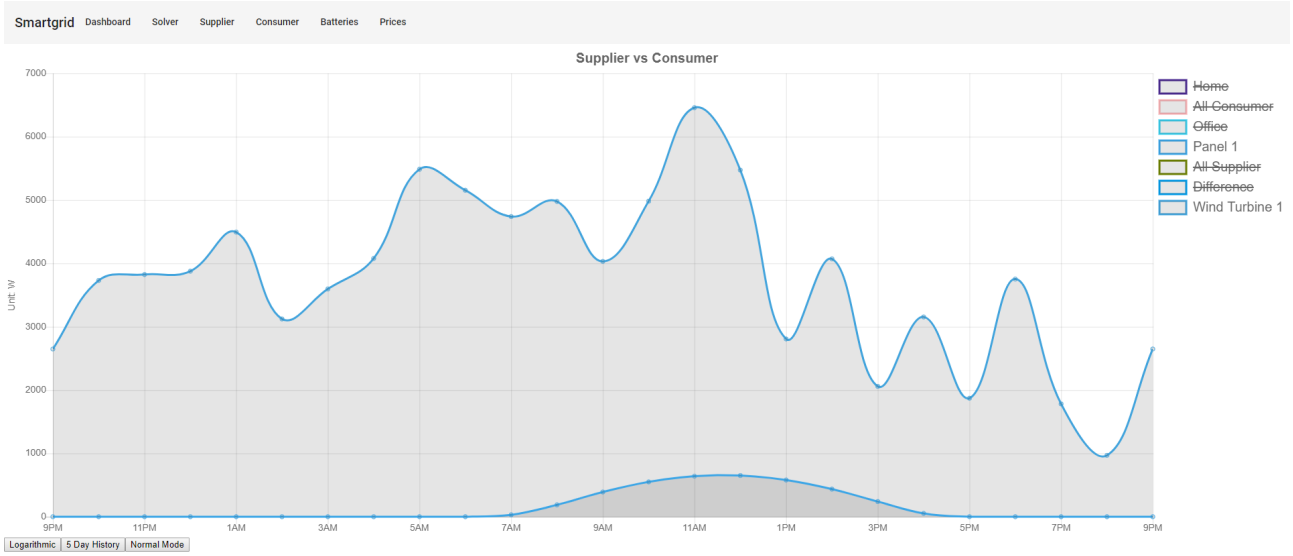

Figure 12: Chart showing only the energy produced by the supplier

determine how long the calculation should run before the solver cancels the search end returns the best solution so far. After pressing the Submit button the simulation request is posted to the system and a progress indicator is shown until the results are back. In the result view the user can see the steps of the solver in three possible ways. The first way is to show the steps in a chart. Figure 14 contains this chart view. Depending on the values in the chart, it is sometimes hard to overview everything fast. But it is possible to remove information from each chart to improve the readability. But some information and connections are easy to spot, for example that the export profit raises (Chart 2) while the supply also increases. All suppliers and all consumers of one type are aggregated before simulation happens. For instance all suppliers and all homes are combined. This helps to keep the simulation times and the memory consumption

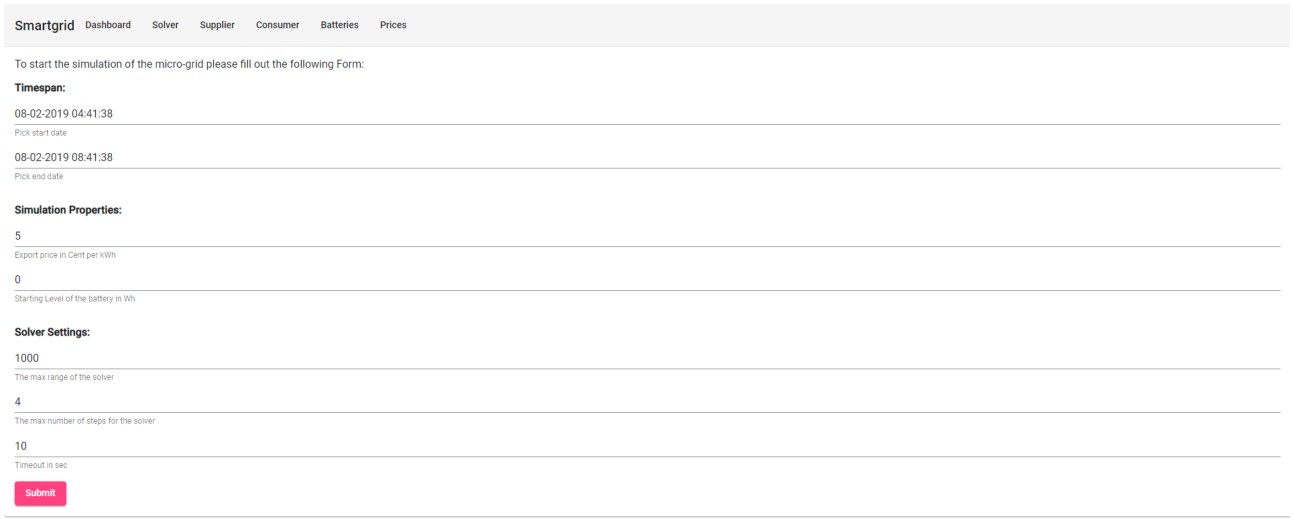

Figure 13: This view is used to configure the solver

at a feasible level. The other view (figure 15) is a tabular view which is best to overview all information fast. There is also the possibility to show a textual description of the steps. The last possibility to view the data is in a very short textual summary for each step. Under the table, chart or text there is always a summary of the costs and profit of the system. Each view in this project is designed to help the user and to make interaction as easy as possible while providing a wide selection of possibilities.

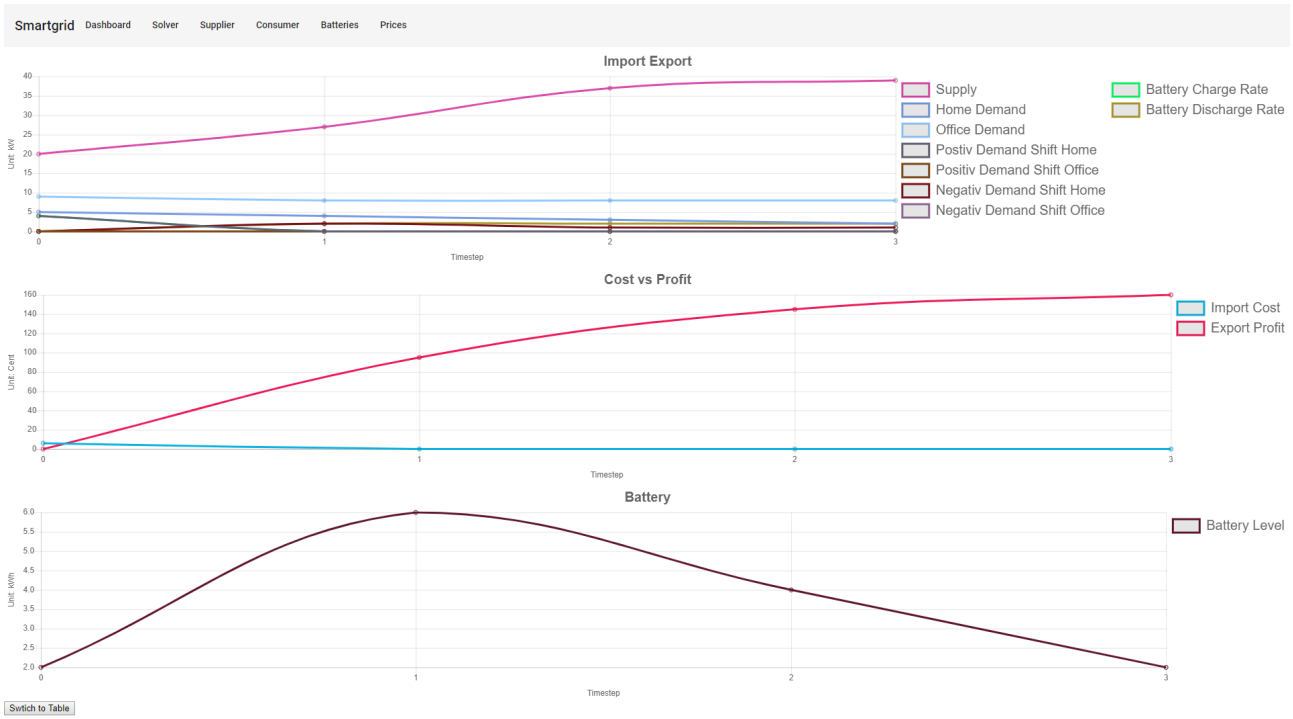

Figure 14: Chart view of solver steps

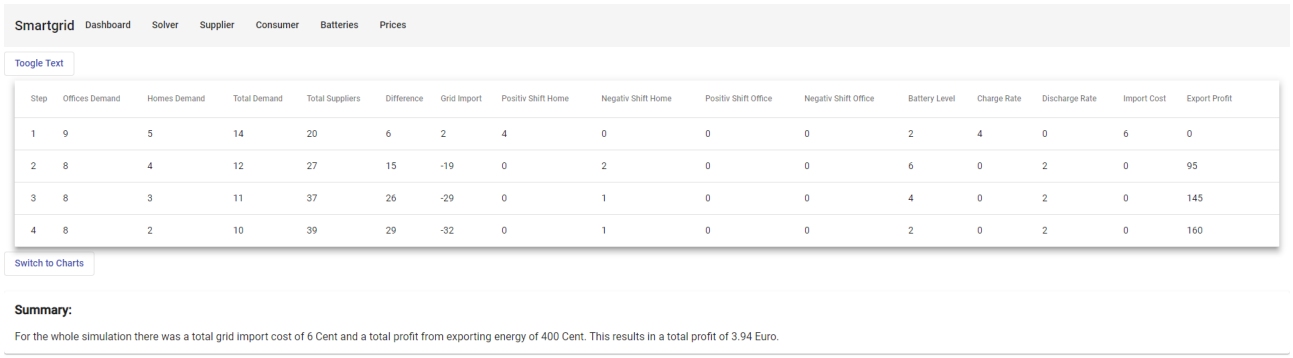

Figure 15: Table view of solver steps

7 Conclusion and Outlook

The power grids 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, micro-grids 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 of different aspects of micro-girds. The first section provides motivation and an introduction to micro girds. The next section contains basic information which are necessary to understand the problem, the functional requirements and an architecture description. The architecture is split into three layers. The web frontend, the logic layer which contains the simulation and the database layer. It also features a reliable approach to integrate an external weather component. In addition to this, there is a service to collect day-ahead energy prices to optimize the smart grid in regard to minimal energy costs. Section 3 contains a description of our implementation. Section 4 provides a description of the optimization problem which has to be solved in order to find the best solution possible for the micro grid. This solution describes when it is profitable to charge batteries, when to shift the demand. The dynamic pricing is described in section 5. In section 6 we described the user frontend. This section focuses especially on which kinds of charts we supply to a user in order to show the user all information in a compact manner.

To summarize this, we built a smart energy system to simulate smart grids. The system is built modular to allow easy extendability with new features, e.g. a new kind of supplier. Using real weather data for specific locations the system can compute the energy produced by the suppliers. Using this data and simulating consumer and batteries, in addition to energy prices for a preconfigured bidding zone, the system can solve an optimization problem to balance out supply and consume regarding to minimize energy costs. The data is shown to the user in a compact dashboard in an Angular based application component, where he can also create supplier, consumer and batteries.

In our implementations, we used for the frontend Angular with Typescript, Angular Material, Charts.js, Moments.js and C3.js. The WeatherCollectorService is written in NodeJS express with ECMAScript 6 features. The main component and the PriceCollectorService are written in Java Spring Boot[9]. All components are written with modularity in mind to support expandability. To support reusability of the different components we decided to use REST for communication among them.

[9] `https://spring.io/projects/spring-boot`

7.1 Outlook

Our system supports at the moment only wind turbines and photovoltaic panels as supplier normal houses and commercial buildings as consumers. Due to the modularity of the system it would is possible to add additional kinds of supplier or consumer to the system in the future.

A big problem is the long run times and the huge memory consumption of the used choco solver if we use extend ranges or calculate more steps. We can only solve the balancing problem for one entity of each supplier, consumer and one battery in a time interval of 4 hours in a fast manner. Using more data ends up in long computation times or in a stack overflow to to memory shortage in the solver. To ease the problem a bit, we concatenate supplier and consumer before giving the data to the solver. A good improvement for the system would be to implement a stronger optimization solver, possibly in another programming language and invoke it from the main component to compute with more data. Also, a distributed solver could be feasible to support more data. Since the rest of our system already supports multiple instances of suppliers, consumers and batteries this change could be easily made, assuming a better solution for solving the optimization problem exists.

A user can create entities using the Opensteetmap component in our user interface. It's possible to add them everywhere in the world. But simulating a smart grid e.g. in China with German bidding zone date for energy prices would be meaningless. To support locations all around the world better, it is necessary to add more price data providers and to add the possibility to configure the bidding zone at runtime. Therefore the system should be changed to collect dynamically energy prices for any bidding zone in the world, regarding the location of the consumer.

Showing the supplier, consumer and batteries in an overview map would make it possible also to add some connectors between them e.g. power cables. This could introduce more realism to the simulation. Or based on the location data of the consumers more variance in the demand profile could be added.

There are several possibilities to improve our smart energy system in future work. We published the project on GitHub under MIT license to allow developers to extend the system, e.g. with pull requests. In our opinion, the long run times of the choco solver are the biggest problem of the current system and should be replaced first before adding additional features.

References

[1] EMD Internation A/S. *WindPRO / ENERGY Modelling of the Variation of Air Density with Altitude through Pressure, Humidity and Temperature*. 2005. URL: `http://www.emd.dk/files/windpro/WindPRO_AirDensity.pdf`.

[2] Nord Pool AS. *API*. 2019. URL: `https://www.nordpoolgroup.com/trading/api/`.

[3] Nord Pool AS. *Day-ahead prices*. 2019. URL: `https://www.nordpoolgroup.com/Market-data1/Dayahead/Area-Prices/ALL1/Hourly/?view=table`.

[4] Nord Pool AS. *Terms and conditions for use*. 2019. URL: `https://www.nordpoolgroup.com/About-us/Terms-and-conditions-for-use/`.

[5] Petr Beckmann. *A History of Pi*. 1976.

[6] Gerard Borvon. *History of the electrical units*. 2011. URL: `http://seaus.free.fr/spip.php?article964`.

[7] Bundesnetzagentur. *Download Marktdaten*. 2019. URL: https://www.smard.de/home/downloadcenter/download_marktdaten/.

[8] Schweizerische Eidgenossenschaft. *Bundesgesetz über das Messwesen*. 2018. URL: https://www.admin.ch/opc/de/classified-compilation/20101915/.

[9] enbw. *Gut zu wissen: durchschnittlicher Stromverbrauch im Haushalt*. 2018. URL: https://www.enbw.com/strom/stromverbrauch-senken.

[10] enertiv. *What is Peak Demand?* 2018. URL: https://www.enertiv.com/resources/faq/what-is-peak-demand.

[11] The Royal Academy of Engineering. *Wind Turbine Power Calculations*. 2018. URL: https://www.raeng.org.uk/publications/other/23-wind-turbine.

[12] ENTSO-E. *Day-ahead prices*. 2019. URL: https://transparency.entsoe.eu/content/static_content/Static%20content/knowledge%20base/data-views/transmission-domain/Data-view%20Day-ahead%20prices.html.

[13] ENTSO-E. *Terms and Conditions*. 2019. URL: https://transparency.entsoe.eu/content/static_content/Static%20content/terms%20and%20conditions/terms%20and%20conditions.html.

[14] ENTSO-E. *Transparency Platform restful API - user guide*. 2019. URL: https://transparency.entsoe.eu/content/static_content/Static%20content/web%20api/Guide.html.

[15] EU-Richtlinie. *EU-Richtlinie 80/181/EWG in den Staaten der EU bzw. dem Bundesgesetz über das Messwesen in der Schweiz*. 2018. URL: https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32009L0003.

[16] NOAA Environmental Research Laboratories. *PROFS Programm*. 1981. URL: https://icoads.noaa.gov/software/other/profs.

[17] PJM. *Day-Ahead Hourly LMPs*. 2019. URL: http://dataminer2.pjm.com/feed/da_hrl_lmps/definition.

[18] NB Power. *kWh and kW Demand*. 2019. URL: https://www.nbpower.com/en/products-services/business/demand-and-energy/kwh-and-kw-demand/.

[19] PVEducation. *Solar radiation on a tilted surface*. 2018. URL: https://pveducation.org/pvcdrom/properties-of-sunlight/solar-radiation-on-a-tilted-surface.

[20] Applied Technology Institute Robert A. Nelson. *The International System of Units*. 2018. URL: https://www.aticourses.com/international_system_units.htm.

[21] EPEX SPOT SE. *ETS Message Interface* Guidelinne*. 2013. URL: https://www.epexspot.com/en/extras/download-center/technical_documentation.

[22] R Shelquist. *Equations - air density and density altitude*. 2009.

[23] Das Internationale Einheitensystem (SI). "Deutsche Übersetzung der BIPM-Broschüre "Le Système international d'unités/The International System of Units (8e édition", 2006)". In: 117 (2006). URL: https://www.ptb.de/cms/fileadmin/internet/Themenrundgaenge/ImWeltweitenNetzDerMetrologie/si.pdf.

[24] Stony Brook University. *Consumption vs. Demand*. 2018. URL: https://www.stonybrook.edu/commcms/energy/facts/demand.

[25] weatherbit.io. *The High Performance Weather API for all of you Weather data needs.* 2018. URL: `https://www.weatherbit.io`.

All links were checked last on February 7, 2019.