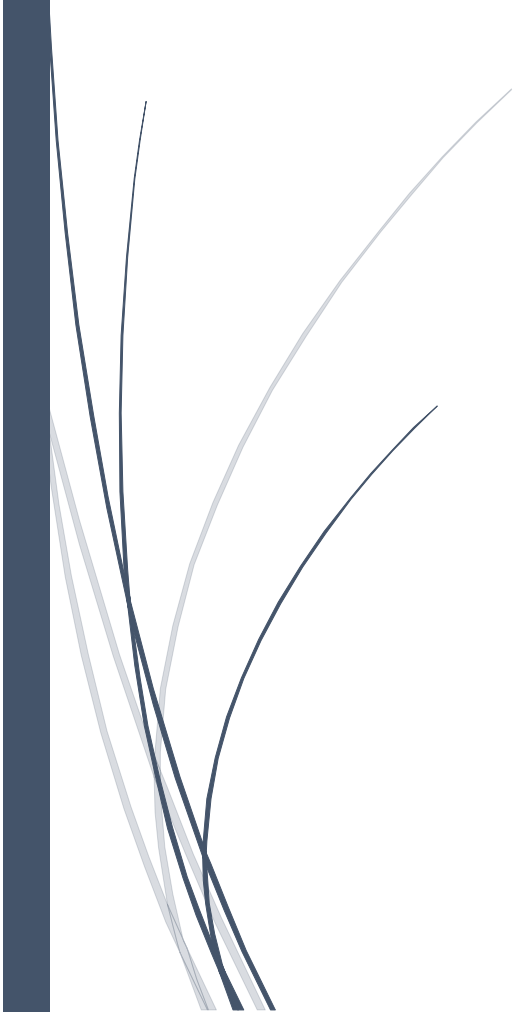
4/24/2022

# Computer Science NEA

## Solar PV Optimisation Project



# Contents

# Analysis

## The Problem

Recently my parents (the client) had solar photovoltaic (PV) panels fitted to their roof with the intent to save energy and money. The panels work directly with a battery and an inverter. The performance of the photovoltaic panels varies a lot with external factors such as the weather, time of day and season. The cost of importing energy from the grid also varies every half hour on the variable tariff. This variability makes predicting and maximising the effect of the panels particularly difficult.

In order to maximise the use of the panels you need to aim to:

- Avoid importing energy from the grid during peak times (where the cost is high, usually 4-7pm and 7-10 am)
- Maximise use of solar generation using the battery to store electricity when generated energy is above what needed at that time of day, and then use that energy when the price of importing energy is high.

GivEnergy is the supplier of the system, and they allow the user access to a CSV file which can be downloaded that contains very detailed data about the system over the past few days. This data is also available via the GivEnergy API. The GivEnergy website also allows users to select the 'mode' of the system. The modes are described below:

- Mode 1: "Dynamic". Battery will charge when excess solar power is available. Battery will discharge when the solar generates less that what is currently being used.
- Mode 2: "Save energy for later use". Stores all excess energy in battery until 4pm (when grid energy cost is high) and then discharges as it needs to.
- Mode 3: "Timed battery discharge". Battery discharges between times inputted by the user
- Mode 4: "Export", exports all excess energy back to the grid rather than storing in the battery

Independent Mode:

- Smart Charge mode: Charges battery up to a limit between a time period from the mains. This mode is independent from all the others

Reference for information above: https://kb.givenergy.cloud/article.php?id=7

Octopus Energy is the energy company that provide the tariff with the dynamic unit rate. Price varies from 0 to around 35p per kWh (price may rarely be negative). The prices vary day-to-day and every 30 minutes over the 24-hour period. The price for any day is determined and available at 4pm day before. Any energy you sell back to the grid is for a price of 5p/kWh. Octopus Energy provide their data via the website and an API.

The panels generate between 0kW and 6kW depending on the amount and direction of sunlight impinging on the panels. Maximum generation is with full sun at midday i.e., 10:30-3:30 where it is 6kW. Moderate generation is with full sun in early morning or late evening i.e., 9:00am-10:30am where it is at 3kW. When it is cloudy generation can reduce to 1kW or even less depending on the level of cloud cover. Hours of good performance are much longer in summer than winter. Home consumption varies significantly from less than 1kW to 5 kW or more throughout the day.

The performance of the system is very variable and hard to predict, and performance is especially hard to maximise by using the modes of the system. The problem is the difficulty the client faces with predicting and setting up the system in order to achieve the aims above. Currently my client is downloading and copying data into a spreadsheet so that he can see all the data about the solar panels, battery, and inverter, which is not very efficient and takes time. A judgement then needs to be made based on this information and by looking at weather forecasts as to which mode the system should be put into. This process would definitely benefit from being performed by a computer program and hopefully it should save a lot of time and effort and make better and more informed decisions on the mode of the system, based on all the available data.

## The Client/User

The client and user for this project are my parents. Their motive for buying the panels is to save money and electricity over the long term and capitalise on a lot of free energy, especially in the summer. Another bonus of the system is the positive impact it should have on the environment as it reduces the need to import energy from the grid – which may not be sourced renewably. The client would like the program to run on their Lenovo Yoga 7i with Windows 10 OS installed. I intend to make the system compatible for any machine with access to the internet and a java runtime. There

is a Wi-Fi dongle on the inverter that the system uses to upload the data to the GivEnergy cloud, and this is where my project will collect the data from, hence part of the need for internet access. My Dad has a lot of IT experience and knowledge. and is currently downloading and copying data into a spreadsheet so that he can see all the data about the solar panels, which is not very efficient and takes time. I hope to improve the situation with my project.

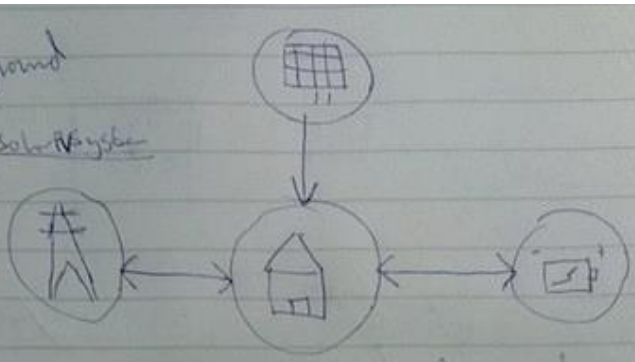
## Communication with the Client

In my first face-to-face meeting with the client we discussed the background of the problem and details of the current system. This included information about the relationships between the inverter, battery, and the panels and how they could all be controlled by using various apps that do not allow full control or flexibility of the system. The client highlighted 2 main aims that are key to using the system and also the problems he faces – To avoid importing electricity at peak times, and to maximise the use of the use of solar generation using the battery to store energy when solar generation is above what is needed at that time. This does help outline the main objectives of the program and allows me to try to make a program that has more functionality than the current system. The client talked about how there is a lot of variability in the system such as the agile tariff for importing/exporting energy and the performance of the panels due to the weather and also the different modes that the system could be put into to take advantage of any particular circumstance. It is clear from my meeting notes that an algorithm needs to be designed that decides on which mode the system should be running based on the many external factors that my program will be able to monitor. Full written notes are below.

Background

Problem

(1) The solar system



- The configurable system with panels and a battery will be ability to import/export from grid
- You choose when to charge / discharge battery
- Can set timings for charging and discharging
- Only charge/discharge battery at 2.5 kW

(2) The Electricity tariff - variable pricing

- Agile Octopus tariff with dynamic unit rate
- price varys from close to 0 up to 35 p (Average 16) during a 24 hour period (sometimes negative) per kWh.
- pricing varies every day, next days price is determined at 4 pm day before. Price just every ½ hour
- Get money back from grid at 5p / kWh when Exporting
- There is a peak between 4 and 7 pm where price very high, small peak from 7-10 am

Aim

1- Avoid importing electricity during peak times

2. Maximise use of solar generation using battery to store electricity when generated above what needed at that time

Problem

2 Apps - Octopus that tells you tariffs for next and current day

- GivEnergy detailed data about system over past days. Allows options of 'mode' of system which it doesn't advise on

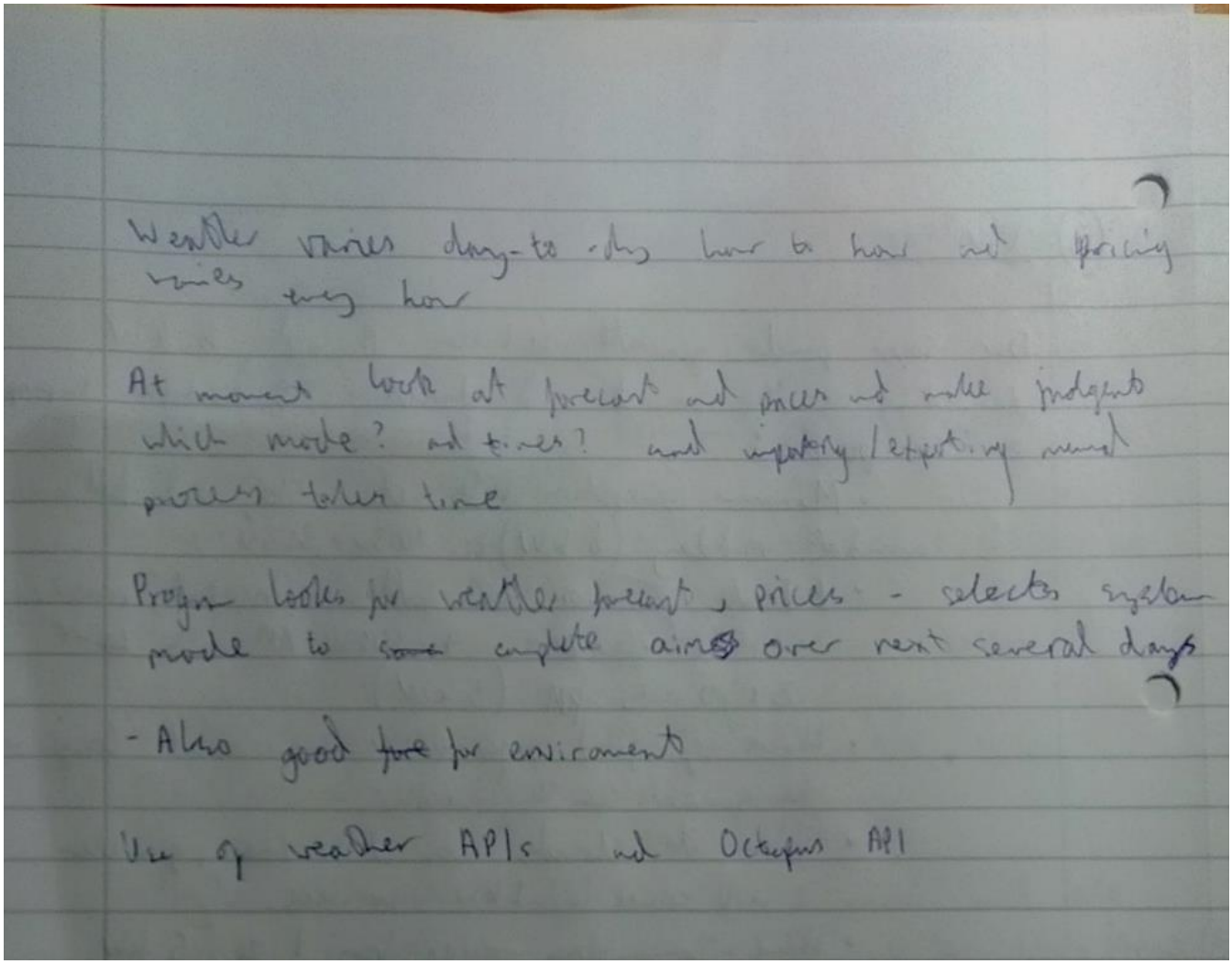
(3) Weather

Our solar panels generate between 0 and 6 kW depending on the amount and direction of sunlight impinging on the panels:

- Maximum generation with clear sky and full sun around midday (6 kW) i.e. 10:30 - 3:30
- Moderate generation with full sun in early morning or late afternoon 9 - 10:30 AM and 3:30 - 5 PM (3 kW)
- Hours of good performance are much longer in the summer than the winter
- When it's cloudy even at midday, generation will reduce to 1 kW or less.
- Home consumption ranges from 1 to 5 kW throughout day and night

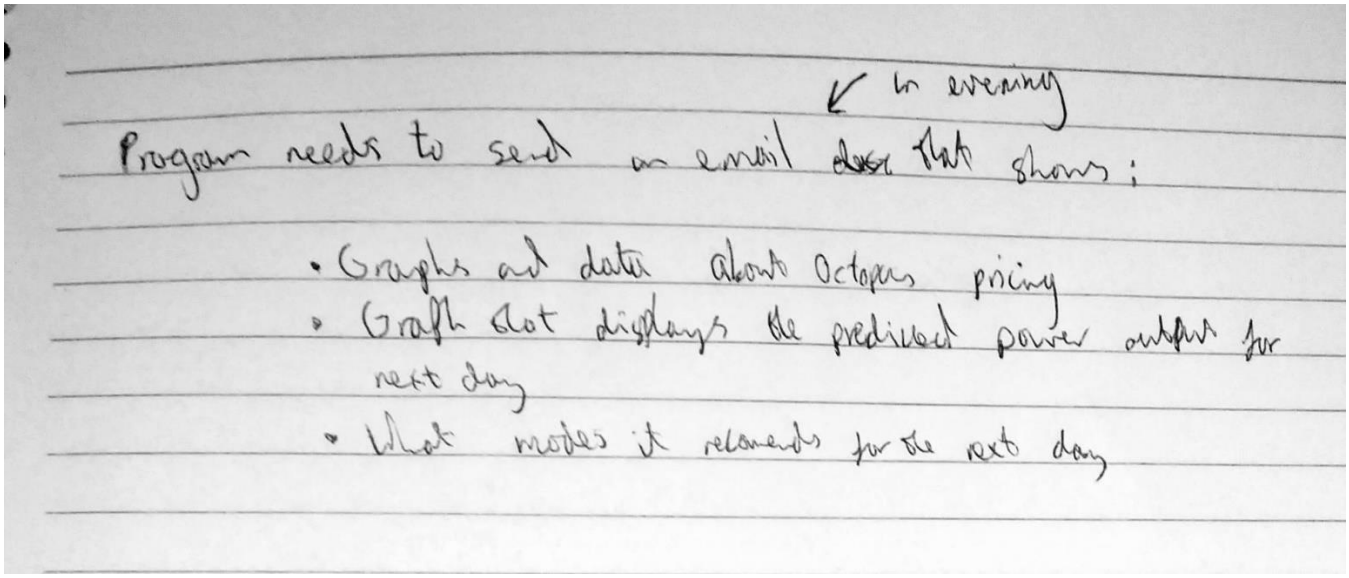
6: GivEnergy

- System AIO
- Mode 1 - Dynamic, battery will charge when excess solar power available. Battery will discharge when solar generates less than what is being used
- Mode 2 - Save for later use, stores excess solar energy until 4 PM, then it discharges when it needs to
- Mode 3 - Timed charging and discharging. Only charges and discharges between times inputted.
- Mode 4 - Export - sends excess energy to grid
- Smart charge - charges up to a limit between a time period from mains

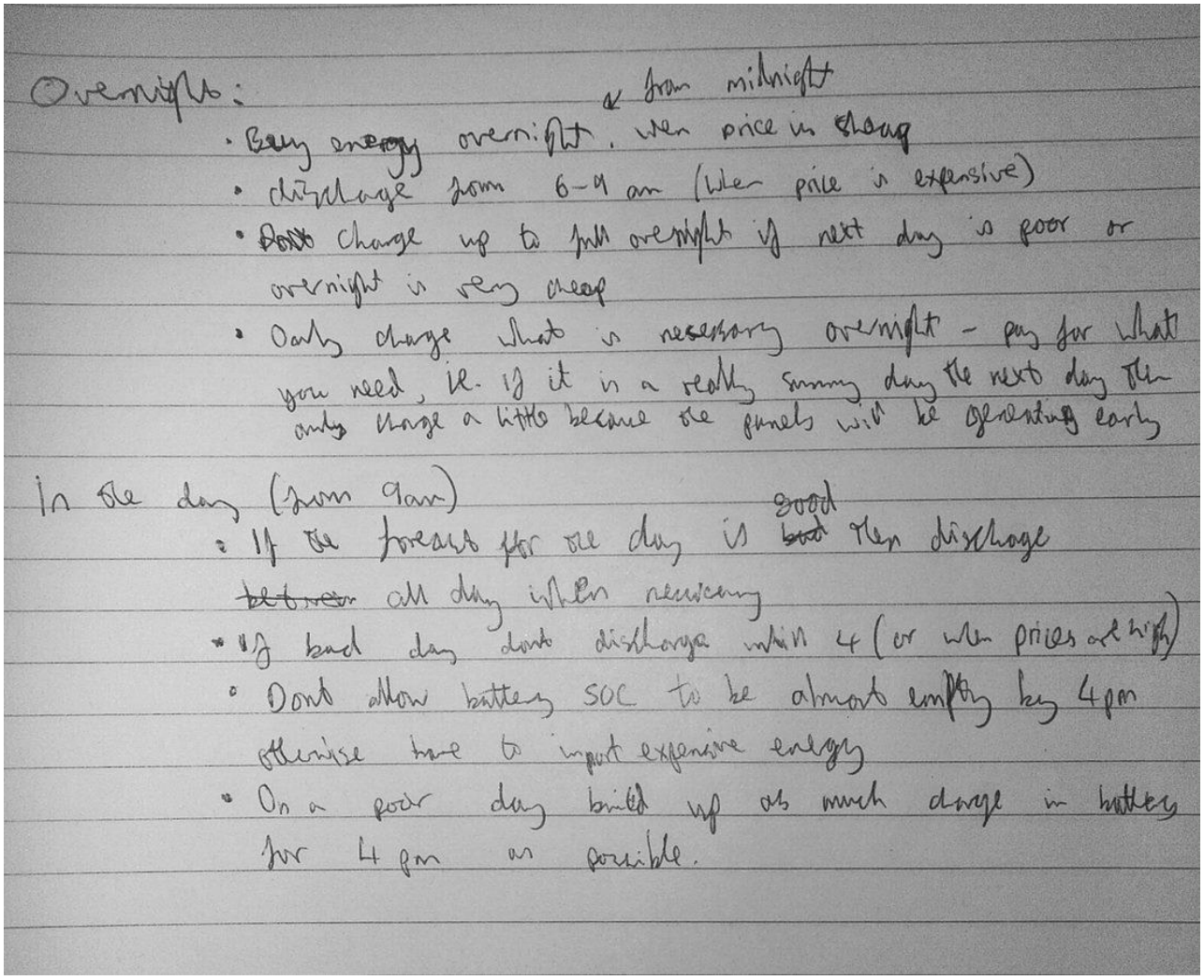



Meeting 2:

In the second meeting the client brought up the idea of an emailing system that emailed them every day in the evening with information about what is expected for the next day and what the program plans to do about it. This is so the client can be kept up to date with what is going on and means they do not need to spend time thinking about the system if everything is looking like it is working properly from the email. This has highlighted to me that I will have to set up a server that is monitoring the forecasts and that can make decisions automatically if the user specifies so. The user should also be able to set system modes manually, but if they choose then the server can handle mode changes automatically. This server will also need to handle the sending of evening emails that summarise the data from that day- this could potentially mean it needs to collect data throughout the day by using the API, but I expect most data could be collected at once the evening as the API provides historic data.




We also discussed the times of day and circumstances when the settings of the system would be changed/discharged. These notes will form the basis of the algorithm I will design to control the system so this was particularly useful to discuss. Here are some notes that describe what the program will aim to do:




It has become clear from this meeting that an important part of making the system work effectively is managing when the battery should discharge or whether to discharge at all. This all depends on the price of imported energy and the weather for that day. The aim is to be only using energy that is stored in the battery when the cost of importing is high, and this means planning out usage over the course of the day and limiting battery discharging to ensure there is enough by 4pm (when the electricity prices usually become high).

## The Current System

The current system involves manually looking at electricity importing/exporting prices and weather forecasts (for example https://www.bbc.co.uk/weather ) viewed on different websites and making a judgement on what system mode it should be in and what times of the day it should be changed, this is very unreliable and there is an element of luck whether it was the correct decision. Currently the client is using excel to store the data files and published electricity prices over the next few days, along with generating a few graphs. Using excel is quite time consuming as currently data is being copied in manually. Here are some screenshots that summarise the current system.

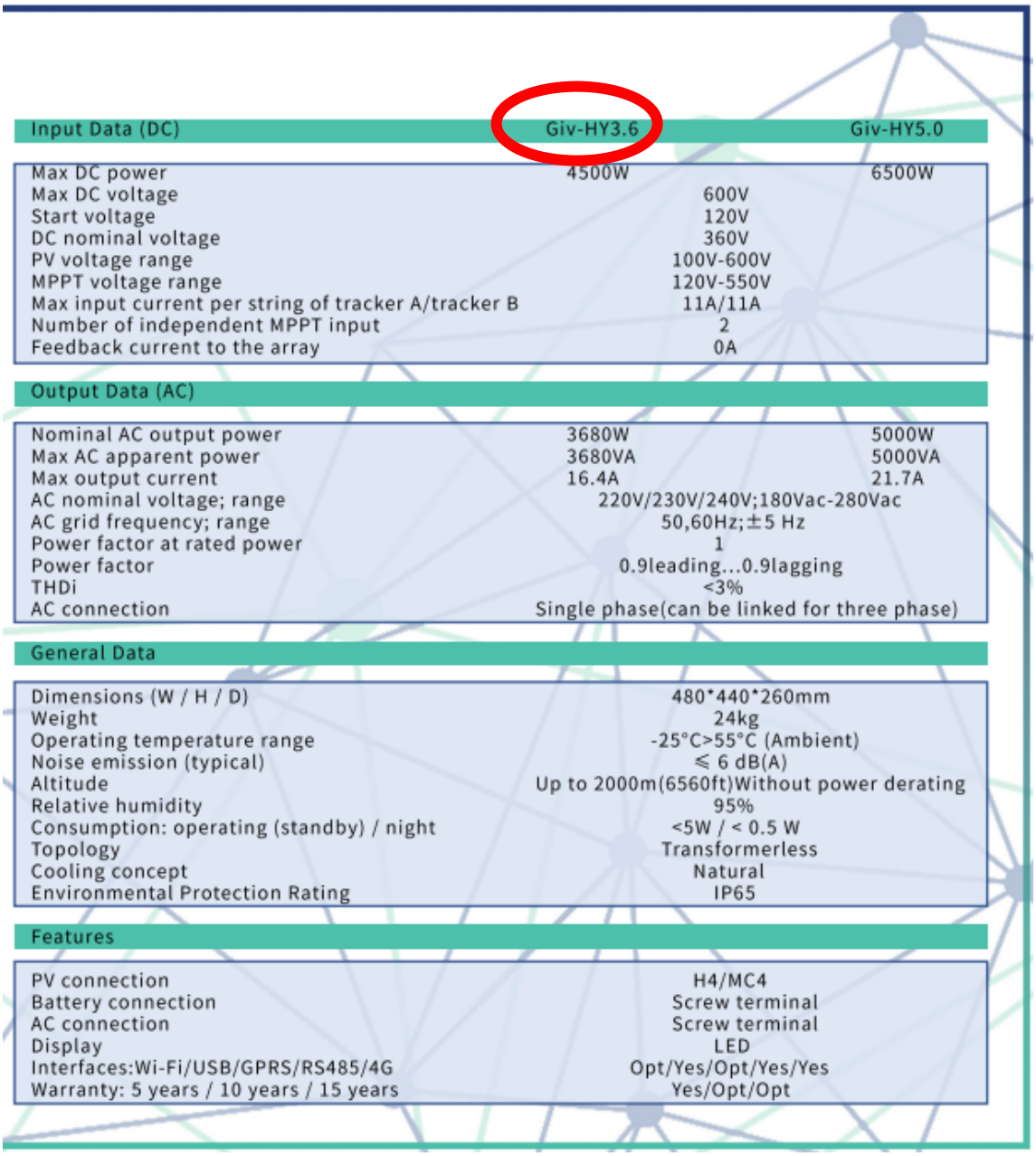
Analysed Data:

This excel document is very large as there is a lot of data it needs to hold, and it takes a lot of time to add new data and edit each sheet every day when more data is generated and so a program that automates this would be very helpful. Changing the mode of the system is also a manual process by using the GivEnergy app or website, but this is also a time-consuming exercise. Changing system modes can be implemented into my project by using the GivEnergy API that they provide. The whole process takes a lot of time and effort to do effectively, with varying success.

## User Needs

The main purpose of the program is to recommend which 'mode' the system should be put in to meet the client's aims – To avoid importing energy at peak times, and to maximise solar generation by using the battery to store electricity when panels are generating above what is needed at that time. Various graphs such as predicted power output over the next few days and graphs for the agile tariff need to be plotted to give the user all the relevant information for the past and next few days. Information about the accuracy of the model, including comparisons between the predicted output and the actual output, should be represented on a graph. A good GUI is needed with a clear and structured layout. Interaction with the software will be entirely through a Swing GUI and this will include buttons that allow direct changes to the mode of the system. The aim is that the project can act as a central control panel where all data, information and control of the system is accessible. The client would like the program to run on their Lenovo Yoga 7i. The specs for this computer are an Intel Core i7 with 12 MB Cache, 8 GB DDR4 RAM, 256 GB SSD, Integrated Intel Graphics. The operating system is Windows 10, 64 bit. The Minimum hardware requirements for my program are an Intel Core i3 (or equivalent), 4GB RAM and 256 HDD. The computer would need at least a 64 bit OS and Java Runtime 1.8.0_301 installed - (https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_system_requirements.html) . The client's specifications will be powerful enough to run my program, as the program will not be particularly resource intensive.

These are the published technical specifications of the Inverter in the system (the Inverter in the system is the Hybrid 3.6kW model) - https://www.givenergy.co.uk/images/GE-A5-V1.4%20PF.pdf this link provides all the equivalent information:



| Input Data (DC) | Giv-HY3.6 | Giv-HY5.0 |
|---|---|---|
| Max DC power | 4500W | 6500W |
| Max DC voltage | 600V | |
| Start voltage | 120V | |
| DC nominal voltage | 360V | |
| PV voltage range | 100V-600V | |
| MPPT voltage range | 120V-550V | |
| Max input current per string of tracker A/tracker B | 11A/11A | |
| Number of independent MPPT input | 2 | |
| Feedback current to the array | 0A | |

| Output Data (AC) | | |
|---|---|---|
| Nominal AC output power | 3680W | 5000W |
| Max AC apparent power | 3680VA | 5000VA |
| Max output current | 16.4A | 21.7A |
| AC nominal voltage; range | 220V/230V/240V;180Vac-280Vac | |
| AC grid frequency; range | 50,60Hz;±5 Hz | |
| Power factor at rated power | 1 | |
| Power factor | 0.9leading...0.9lagging | |
| THDi | <3% | |
| AC connection | Single phase(can be linked for three phase) | |

| General Data | |
|---|---|
| Dimensions (W / H / D) | 480*440*260mm |
| Weight | 24kg |
| Operating temperature range | -25°C>55°C (Ambient) |
| Noise emission (typical) | ≤ 6 dB(A) |
| Altitude | Up to 2000m(6560ft)Without power derating |
| Relative humidity | 95% |
| Consumption: operating (standby) / night | <5W / < 0.5 W |
| Topology | Transformerless |
| Cooling concept | Natural |
| Environmental Protection Rating | IP65 |

| Features | |
|---|---|
| PV connection | H4/MC4 |
| Battery connection | Screw terminal |
| AC connection | Screw terminal |
| Display | LED |
| Interfaces:Wi-Fi/USB/GPRS/RS485/4G | Opt/Yes/Opt/Yes/Yes |
| Warranty: 5 years / 10 years / 15 years | Yes/Opt/Opt |

The panels are electrically connected to the Hybrid Inverter. The purpose of the inverter is to convert the DC electricity into AC electricity, which can be used in the home. The inverter is also connected to the 160Ah lithium battery which can store charge when it needs to. The inverter has an inbuild computer that allows it to be the controller of the system, for example changing the mode of the system or choosing when to charge/discharge the battery. The inverter has a USB port for a Wi-Fi dongle that allows the inverter to connect to the home network and then store all system data on the GivEnergy cloud (https://www.givenergy.cloud/). This data is accessible via the GivEnergy API or downloadable in a CSV format (https://kb.givenergy.cloud/category.php?id=8) . This also means that the client will need internet access to run the program as it will need access to the API.

## Proposed Solution

The program will predict the performance of the solar panels and recommend a system mode that would use the generated energy in the most effective way over the next few days. In order to collect data from the solar PV system (ie. PV power output data), the program should use the GivEnergy

API. This should be collected at the end of each day when the panels have stopped generating any power. At the same time, the program should use the Solcast API which will return half-hourly “estimated actuals” of solar irradiance (GHI - Global Horizontal Irradiation) data for that same day. The solar irradiance values from the API takes into account all cloud coverage at the time and should give a very good estimate of the solar radiation at the chosen location of the solar panels. The solar panel power output data and the estimated actual solar radiation data are then paired up as bivariate data according to the time of day they were recorded and then the data is added to a database table that stores all the datapoints in the model.

Over the course of time the program will develop a model with two variables: solar radiation intensity and actual PV power output. These variables can be used to create a least squares regression model, which will form the basis for the machine learning of the program. Over time, as more data is added, the model will develop and learn and be able to return a prediction for the power output of the panels for a given solar radiation forecast. The program can use this model to predict the performance of the panels with solar radiation forecasts it has collected for the next few days. These predictions will be necessary in the algorithm that decides on the most effective mode that the system should be put into. Along with these predictions, the Octopus Energy API needs to be used collect the next day’s agile tariff prices for importing/exporting energy from/to the grid. These prices will need to be included in the mode recommendation algorithm.

Finally, when a system mode has been decided upon the program should be able to change the system mode using the GivEnergy API that can connect directly to the solar PV system and change them in real time. The user should be able to press buttons for which mode they want after seeing what the program recommends at that time for the various hours of the day. It could also be able to automatically change the system mode by itself throughout the day to save the user even more time. This will be achieved by creating a server program hosted on a Raspberry Pi. This program will run 24/7 and has several purposes – to automatically control the modes of the system when the user specifies it to do so, to send daily emails to the client with information about that day which includes all relevant graphs and a log of mode changes. The client will only need to interact with one desktop application in day-to-day use and they have full control of the system from that, but the server program will handle the tasks described above so that the client is able to close this desktop application and turn off their computer whenever they like.

The program will plot various graphs that will be useful for the user. One of the graphs will be a bar chart which shows the Octopus Energy agile tariff so the user knows the costs of importing electricity for the next day. Another graph will be the power output of the panels against the time of day for the past day or any day in the next few days, the data for this can be collected from the database. This type of graph can also be generated for the predicted output of the panels to the user can get a visual representation of the future performance. Another graph that may be useful is the regression scatter plot and line of best fit which will show how effective the model is, also the program could give a value for the current accuracy of the model. These graphs will be sent in the evening email to the client to summarise that day.

Within the desktop application there will be several areas that display different types of information. There will be an “Overview” tab that displays the real time performance of the panels and battery such as the power being generated right then or the battery SOC. Within this area there will be graphs that display historic data about the system for example, energy generated by month for the last year. In another section there will be a “forecasting” tab that displays graphs for the forecasted weather and most importantly the forecasted Power output from the panels – generated from the model. This gives the user a good understanding of how the panels are likely to perform over the

next few days and provides the context behind and decisions the program makes when recommending modes etc. This section will also include comparisons between the forecasted performance Vs the actual performance so that the user can get a feel for the accuracy of the forecasting. A calculated value of accuracy could also be displayed on this. Another section in the application is a control area where the user will be able to press buttons and change the real system settings for the panels, inverter, and battery. The user's decision to change the modes is up to them but the program will give a recommendation as to which mode to choose and time periods that that mode should be selected for. If the user wishes, an ON/OFF switch can be used to change whether the program can automatically change the system modes. In this case the user would need no input into the system until they turned this off as this would all be handled by the server on the Raspberry Pi automatically.

https://kb.givenergy.cloud/article.php?id=54

https://kb.givenergy.cloud/article.php?id=7

## Objectives

### Inputs

1. The program will communicate with APIs to collect data needed by the program
    1.1. The program will communicate with the Solcast API to get forecast data about the next 48 hours of solar radiation (GHI), Cloud Opacity, Sun Azimuth and Sun elevation Angle
    1.2. The program will communicate with the Octopus Energy API, which will return data about the agile price tariff for importing energy from the grid
    1.3. The GivEnergy API will be used to collect data produced by the solar PV system, this includes both live real time values and historic data.
    1.4. All data collected from APIs will be validated and error messages will show if it appears as if the API response was erroneous
    1.5. The program will also send a message if there is likely to be no internet connection and this means that no API inputs will be able to be retrieved
2. The program will be able to collect data from a database
    2.1. Will be able to collect data from a table that holds all datapoints used in the model
    2.2. Will be able to collect data from a table that hold calculated coefficient vales for the model
    2.3. Will be able to collect charging and discharging periods from a table that stores these overnight for the next day.
3. The user will be able to set parameters about specific details of their solar panel site such as location, azimuth, and tilt of the panels.
    3.1. This will be in a “settings” page within the main GUI
    3.2. For setting the tilt angle of the panels a slider component will be used
    3.3. A slider will also be used for the azimuth settings
    3.4. The latitude and longitude of the site location will be inputted into text fields
    3.5. These text fields will have validation where an error box shows if the inputted coordinates are outside a valid range
4. The program will be able to read from files
    4.1. It will be able to read data stored in JSON format from a text file that holds information about the orientation of the panels
5. The user will be able to control the solar PV system within the GUI
    5.1. The user will be able to select any of the 4 discharging modes available via the GivEnergy API using a radio button
    5.2. Only one radio button will be able to be selected at a time
    5.3. When entering discharging periods into text fields, the time needs to be of ‘HHmm’ format and error message will show if it is not
    5.4. The user will be able to set charging periods under the ‘independent features’ section and also input a percentage value the battery should charge to
        5.4.1. This section is optional and so it will be able to be enabled or disabled with a toggle button
    5.5. The user can set a minimum battery reserve charge value so that the battery never runs out
    5.6. There will be a save changes button that finalises the changes made and uses the API to make the changes
    5.7. There will be a revert changes button that resets the page if the user enters the wrong values and forgets what it is supposed to be
    5.8. [Extension] There will be a toggle button that allows the program to automate setting changes so that the user never has to do the changes manually

6. The program should have a graphs and data page, that allows the user to input which data they would like to view and potentially view variations of multiple data sets on the same axis.
    6.1. There will be 5 radio buttons for the power graph that allow the user to select the following datasets to be shown on the graph: Import, Export, Solar Generation, Demanded Power, Charging/Discharging
    6.2. The user should be able to select the date of when the data is from so any data from the past can be displayed
7. The user will navigate around the GUI using a tabbed pane
    7.1. The tabbed pane layout will have a different section in each tab according to what types of information need to be displayed
    7.2. There will be sub-tabs within the main tabs that display related information under the same topic
    7.3. In order to change tabs the user can click on a different tab

### Data Processing

8. The program needs to be able to communicate with databases and APIs to update the complex mathematical model
    8.1. At the end of the day the program will be able to update the complex mathematical model after an update model button is pressed
    8.2. The program will collect power output data from the GivEnergy API for that day and store it in an array
    8.3. The program will then use the Solcast API to collect estimated actual values of solar radiation for that day
    8.4. A process will be run to recalculate the solar radiation values to calculate the component of solar radiation incident on the plane of the solar panels
    8.5. Then the Solar radiation data and solar panel power output data are all appended to a database table that stores all the datapoints
    8.6. Then the model will be re-evaluated with these new data by doing least squares regression calculations and these calculated model coefficients will be stored in another table in the database
9. The program will run an algorithm based on the predicted power output from the panels over the next few days to determine which modes at which times the system should be put into, then it can recommend this mode to the user, or automatically change the mode without the user having to be involved.
    9.1. The algorithm will use the predicted power output values from the complex mathematical model and the Octopus agile electricity prices
    9.2. The amount of energy produced between every half hour period of a day will be predicted e.g how much energy the panels produce between 7:30-8:00am by using the trapezium rule
    9.3. The program uses these energy predictions and electricity prices to determine the most optimum periods of charging and discharging for the next day
    9.4. The program will then decide on the most advantageous strategy and time periods for charging/discharging and presents this to the user as the “Recommended Settings” for the next day
    9.5. [Extension] If the user has selected that the program automates the changing of modes then these calculated timings and modes will be dealt by with the server and everything will be automated

9.6. If the user has not selected the automated mode changing, then the server will not handle this and the user will be able to either follow the advice of the program or do what they want with the settings available

### Outputs

10. The GUI will have a Tabbed Pane layout to simplify navigation around the program and also has all the necessary features to allow full control over the system.
    10.1. This Includes having no more than 6 sub-tabs per main tab
    10.2. There will be 4 Main tabs: System Overview, Forecasting, System Control, Settings
        10.2.1. System Overview will have 3 subtabs: Live Values, Power Data, Octopus Agile Prices
        10.2.2. Forecasting will have 5 subtabs: GHI, Cloud Opacity, Sun Azimuth, Sun Elevation, PV Output
            10.2.2.1. PV Output will have 2 subtabs: Power Output Forecast and Model Data
    10.3. All crucial buttons that are needed for the selection of mode changes need to be aligned and in a logical order and grouped by their purpose
    10.4. [Extension]A toggle button will be included that allows an automatic system control to take over which means the user does not have to open the software as often
    10.5. A loading animation will be added so that when the program starts up and makes initial connections to databases and APIs it does not seem like the program is crashed
11. Graphs will display data from calculated values and collected data
    11.1. Power data from the system provided by the GivEnergy API
    11.2. Octopus Energy Agile tariff bar chart
        11.2.1. There will be a date chooser that allows the user to select which date the Agile prices are from
    11.3. Predicted power output of the solar panels
    11.4. A scatter plot and curve of best fit for the model
    11.5. [Extension]Graph comparing predicted power output and actual power output over the previous 48 hours
    11.6. Weather forecast graphs from Solcast will include:
        11.6.1. Forecasted GHI graph
        11.6.2. Cloud coverage forecast graph
        11.6.3. Sun elevation graph
        11.6.4. Sun Azimuth graph
12. There should be an “Overview” page that shows live values from the system such as current power output from the panels, power usage from the home and even a value for tonnes of $CO_2$ saved by the system for that day
13. [Extension]The program will send a daily email to the client with graphs for predicted power output and electricity pricing over the next day, it will also include what settings it plans to change
14. The program should be able to write to files
    14.1. It will be able to save Location data (longitude and latitude) when that is entered by the user in the setting tab after a save changes button is pressed
    14.2. It will be able to write panel tilt and panel azimuth values to a file when entered by the user after a save changes button is pressed

### Data Storage

15. The program will use a remote online hosted MySQL server to store all the data it needs
    15.1. The program will store all datapoints used in the model in one table
    15.2. The program will store the model coefficients in another table

15.3. The program will be able to access this model data when it needs to make calculations about the future performance of the solar panels.
15.4. If the database is inaccessible, e.g. the user's computer is not connected to the internet, then a message will show up to help them fix the issue
15.5. All data being stored in the database will be validated before it is added – check for values outside of an acceptable range, null values and not including repeat values
15.6. The program will display a loading spinner while data is being send and collected from the database to add to user experience

16. The program will use a text file to store user preferences and settings
16.1. This text file will store data in JSON format

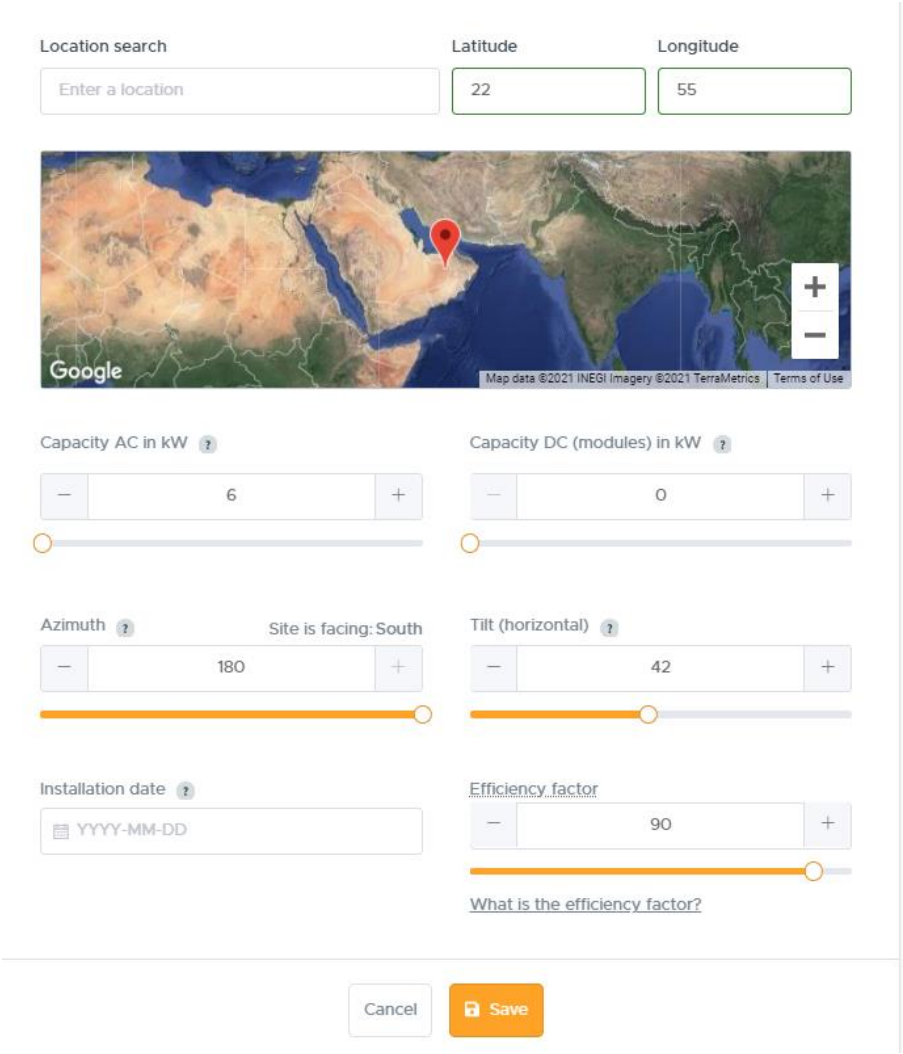
# Research

## Alternative Systems

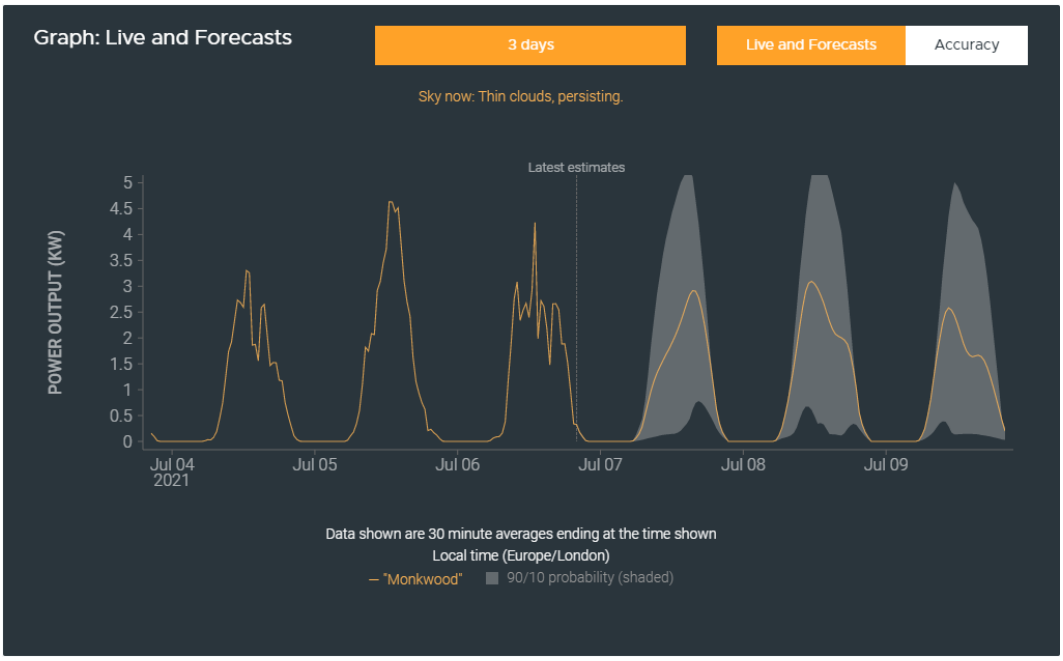
### *Alternative System 1*

https://toolkit.solcast.com.au/live-forecast

This is the Solcast API Toolkit, it mainly focuses on providing power output forecasts and solar radiation forecasts for a 'site.' A site is a location with solar panels, and you can set this up and enter lots of details about the site including latitude, longitude, AC capacity for the inverters and the azimuth and tilt of the panels. I would like to have a menu like this in my project where the client can input information about their site, especially info about the panel orientation and tilt.




The website can use all of this to predict future performance. I have taken a lot of inspiration from this website, and I really like how they can represent their forecast graphically, especially with the shaded area for the uncertainty of the forecasts.



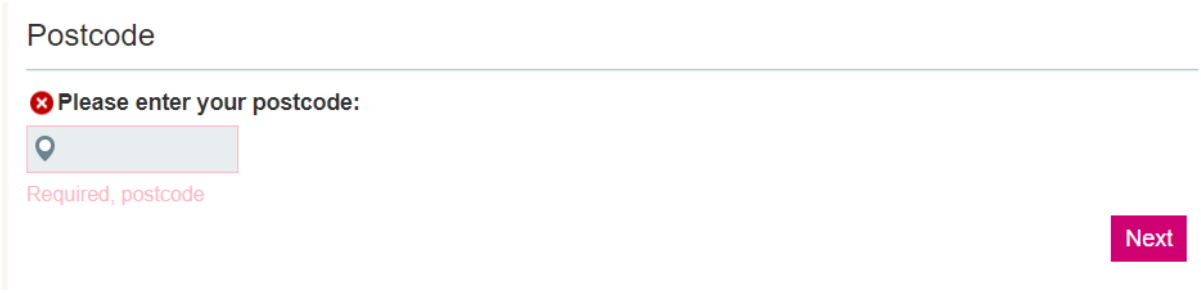

They also have an "Accuracy" graph which compares estimated actuals from satellite data and 1 hour ahead forecasts which provides a bigger picture on the accuracy of the data they forecast:


Solcast also provide an API that returns all their solar irradiation and weather forecasts, and I will be using this API in my project.

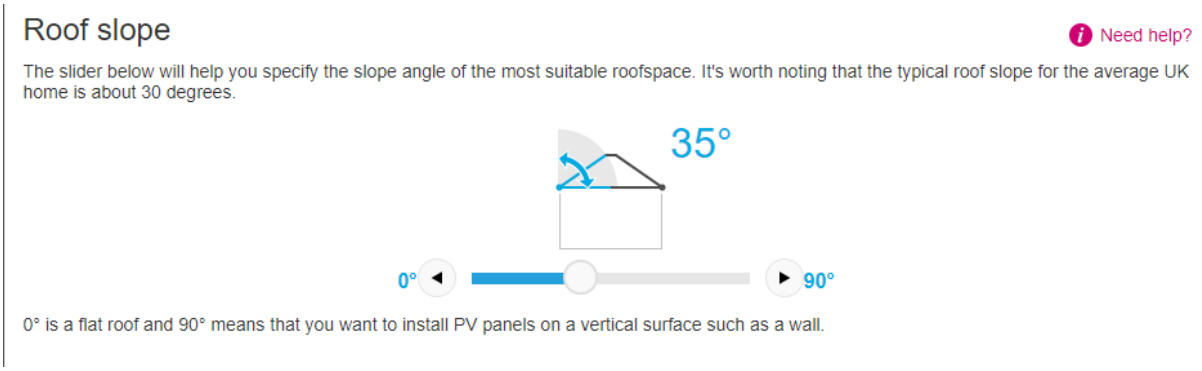
### *Alternative System 2*

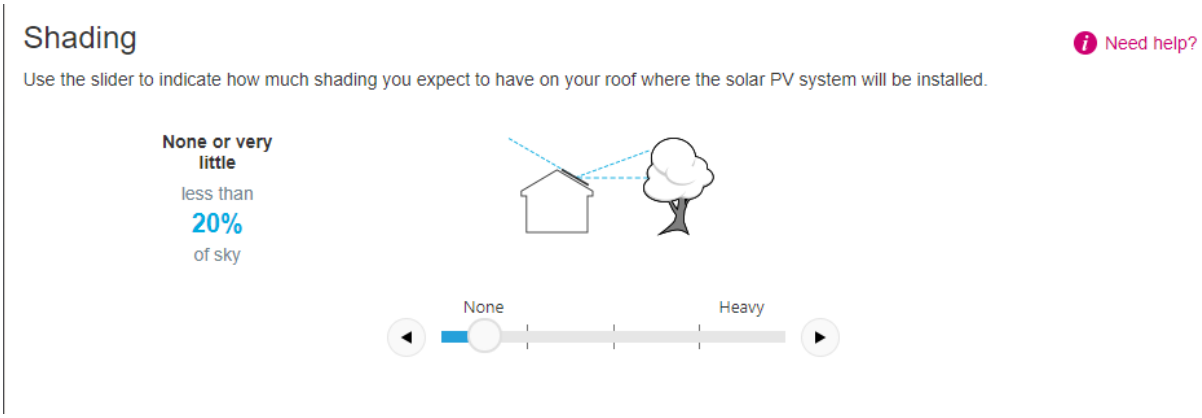
https://www.pvfitcalculator.energysavingtrust.org.uk/

The second alternate system is the Solar Energy Calculator by the Energy Saving Trust. This website provides information based on location, orientation, and size of the solar panel system. Firstly, the website asks for you postcode, in my project I will work with a latitude and longitude as this tends to be a requirement in many APIs.




In the next stage of the process, it asks for details such as roof slope, shading, roof direction and installation size. This does give me lots of variables to consider, especially location specific details like shading and I may have to allow the user to input this in the program.



## Roof direction

Need help?

The slider below will help you specify the direction of the most suitable roofspace.

Adjust the map to find your property and then adjust the compass to align it with the desired direction.

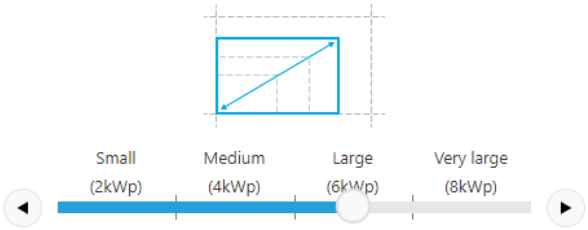

Please adjust slide bar to correct direction

## Installation size

Need help?

Please choose the size of the system you're interested in installing. View sizing guide for more help.

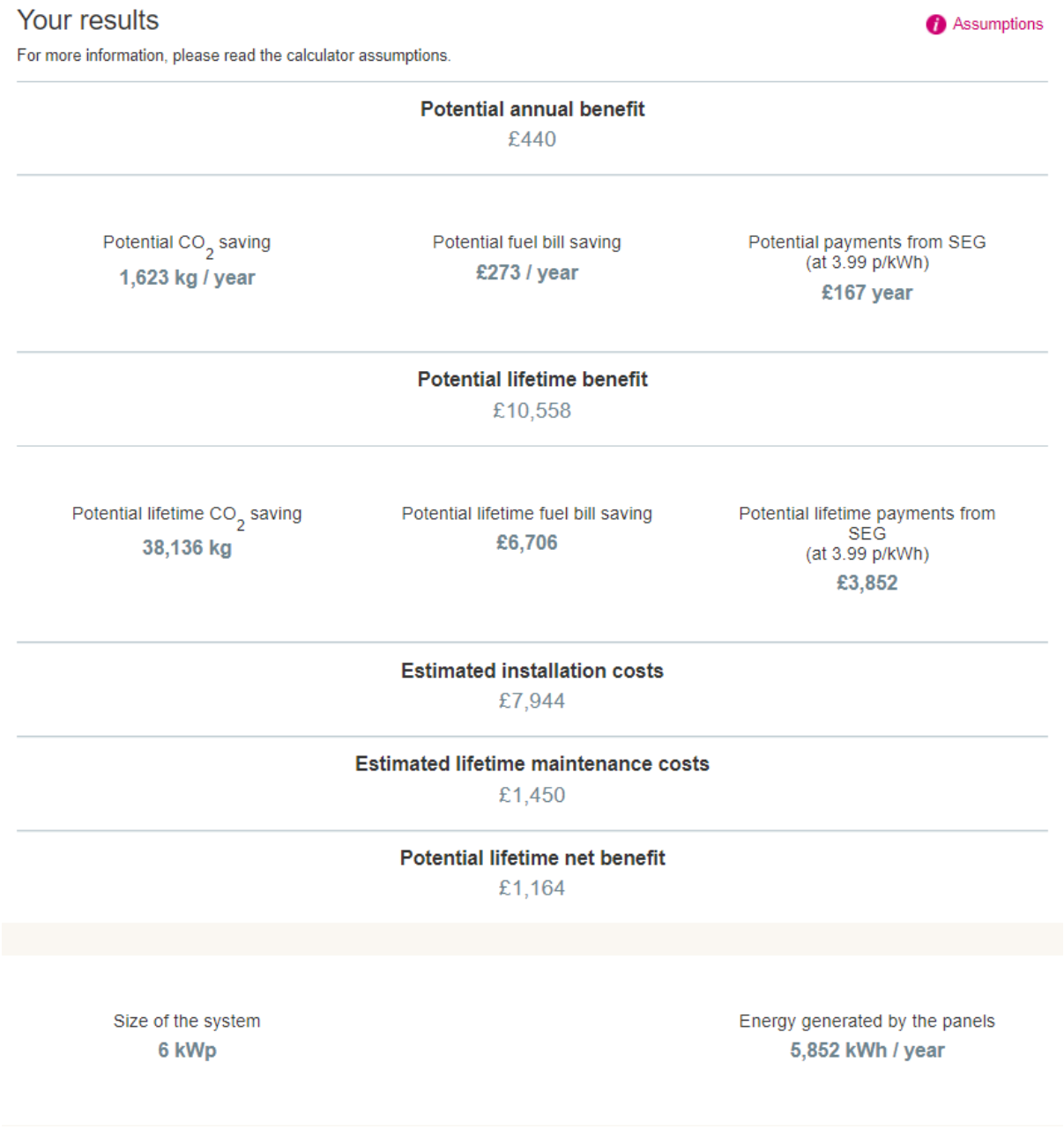
Our standard assumption is Medium (4kWp) as this is around the average size of solar PV panels that have been installed on domestic properties. Typical solar electricity systems usually require 10-20m$^2$ of unshaded space. Smaller systems are possible, but less likely to be cost effective.




**OR,** If you know it, what is the specific peak generation of the system you plan to install in kilowatts?

(Optional)

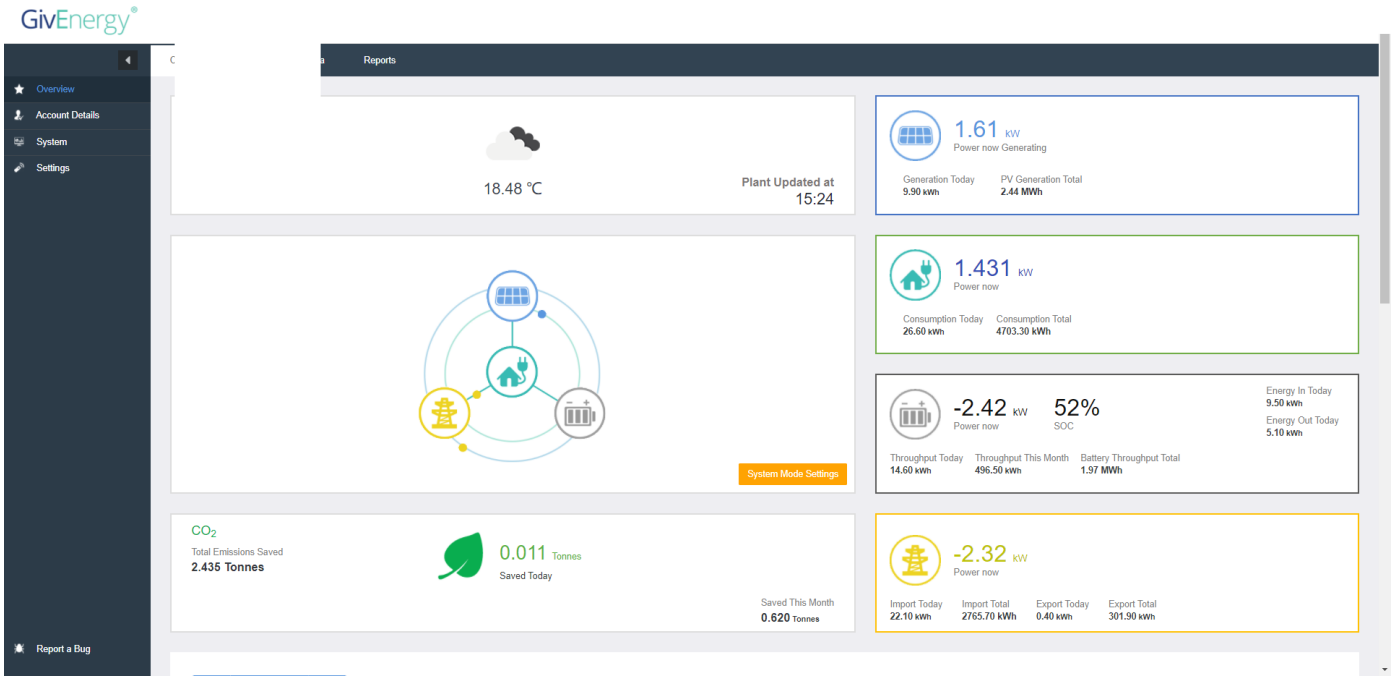
After all the details have been filled in the website return lots of calculated prediction on financial and environmental benefits over the long term of the system. This is a really interesting feature, and this is definitely something I would like to integrate into my project as a long-term prediction aspect so that the client can see the overall benefits of their system.




*Alternative System 3*

https://www.givenergy.cloud/GivManage/login

The third alternate system is the GivEnergy website and software that acts as a companion to the system. It is directly connected to the client's inverter and battery and will display all data and allow the user full control over the system. My project will have many similar features and will take lots of inspiration for the style of data representation and system control that the GivEnergy website has. All the data displayed on this website is accessible via the GivEnergy API, I will be using this to collect all data the program needs about the system. Here is a screenshot of the main 'Overview' section:

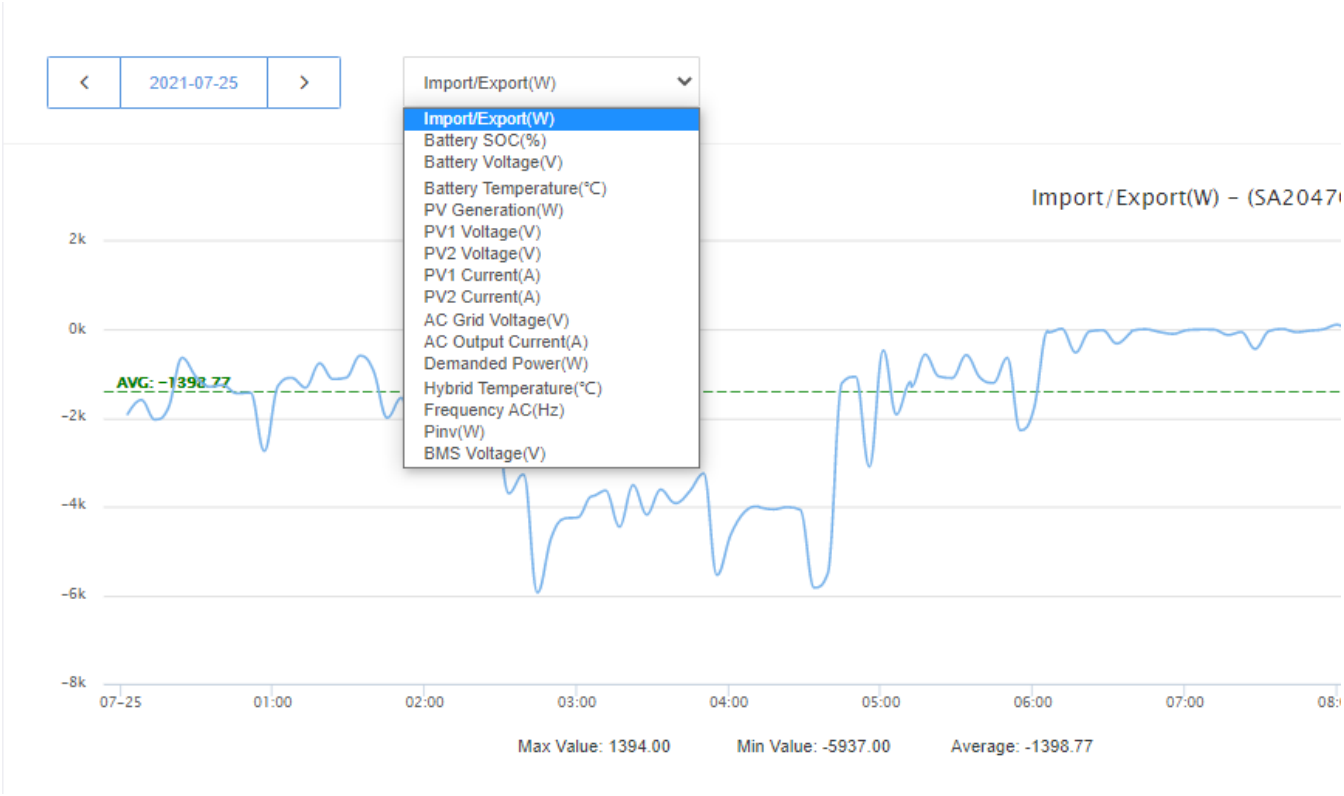



I like the idea of this overview page that displays live data from the system, and this is something I will implement into my project.

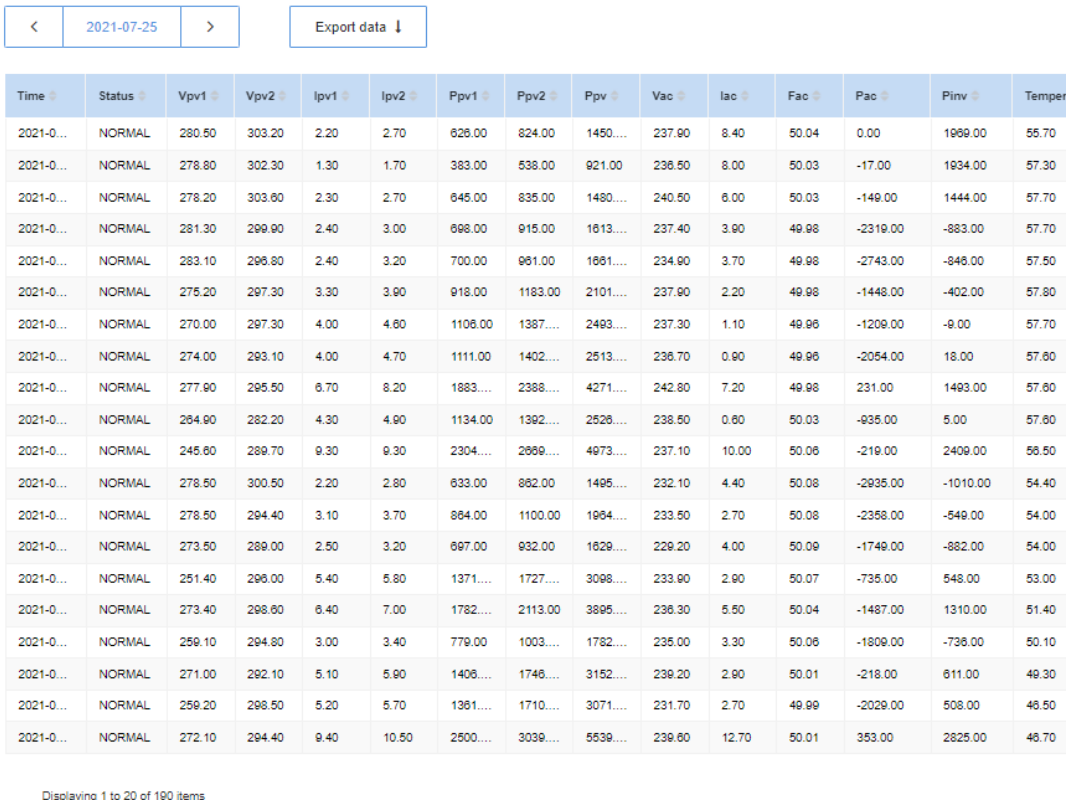
They also have some interactive graphs that display information about energy and power for any given date:


This is a nice feature, and it gives the user a good understanding of past performance from any date they would like, all the data is accessible via their API.

They also provide graphs of specific data from any part of the system:



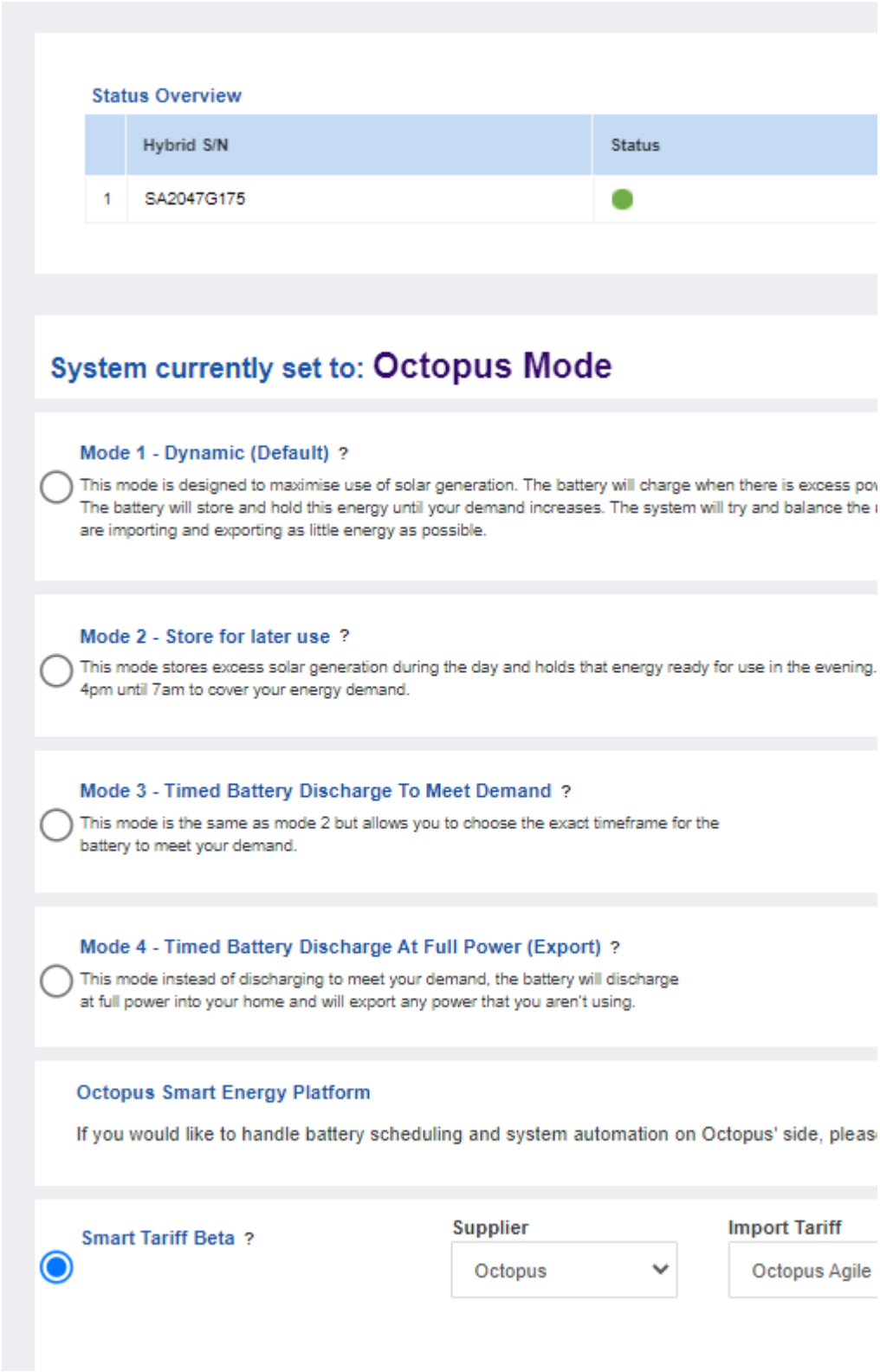

They allow access to raw data and allow the option to “export” the data to your computer:




| Time | Status | Vpv1 | Vpv2 | Ipv1 | Ipv2 | Ppv1 | Ppv2 | Ppv | Vac | Iac | Fac | Pac | Pinv | Temper |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2021-0... | NORMAL | 280.50 | 303.20 | 2.20 | 2.70 | 626.00 | 824.00 | 1450.... | 237.90 | 8.40 | 50.04 | 0.00 | 1969.00 | 55.70 |
| 2021-0... | NORMAL | 278.80 | 302.30 | 1.30 | 1.70 | 383.00 | 538.00 | 921.00 | 236.50 | 8.00 | 50.03 | -17.00 | 1934.00 | 57.30 |
| 2021-0... | NORMAL | 278.20 | 303.60 | 2.30 | 2.70 | 645.00 | 835.00 | 1480.... | 240.50 | 6.00 | 50.03 | -149.00 | 1444.00 | 57.70 |
| 2021-0... | NORMAL | 281.30 | 299.90 | 2.40 | 3.00 | 698.00 | 915.00 | 1613.... | 237.40 | 3.90 | 49.98 | -2319.00 | -883.00 | 57.70 |
| 2021-0... | NORMAL | 283.10 | 296.80 | 2.40 | 3.20 | 700.00 | 961.00 | 1661.... | 234.90 | 3.70 | 49.98 | -2743.00 | -846.00 | 57.50 |
| 2021-0... | NORMAL | 275.20 | 297.30 | 3.30 | 3.90 | 918.00 | 1183.00 | 2101.... | 237.90 | 2.20 | 49.98 | -1448.00 | -402.00 | 57.80 |
| 2021-0... | NORMAL | 270.00 | 297.30 | 4.00 | 4.60 | 1106.00 | 1387.... | 2493.... | 237.30 | 1.10 | 49.96 | -1209.00 | -9.00 | 57.70 |
| 2021-0... | NORMAL | 274.00 | 293.10 | 4.00 | 4.70 | 1111.00 | 1402.... | 2513.... | 236.70 | 0.90 | 49.96 | -2054.00 | 18.00 | 57.60 |
| 2021-0... | NORMAL | 277.90 | 295.50 | 6.70 | 8.20 | 1883.... | 2388.... | 4271.... | 242.80 | 7.20 | 49.98 | 231.00 | 1493.00 | 57.60 |
| 2021-0... | NORMAL | 264.90 | 282.20 | 4.30 | 4.90 | 1134.00 | 1392.... | 2526.... | 238.50 | 0.60 | 50.03 | -935.00 | 5.00 | 57.60 |
| 2021-0... | NORMAL | 245.60 | 289.70 | 9.30 | 9.30 | 2304.... | 2669.... | 4973.... | 237.10 | 10.00 | 50.06 | -219.00 | 2409.00 | 56.50 |
| 2021-0... | NORMAL | 278.50 | 300.50 | 2.20 | 2.80 | 633.00 | 862.00 | 1495.... | 232.10 | 4.40 | 50.08 | -2935.00 | -1010.00 | 54.40 |
| 2021-0... | NORMAL | 278.50 | 294.40 | 3.10 | 3.70 | 864.00 | 1100.00 | 1964.... | 233.50 | 2.70 | 50.08 | -2358.00 | -549.00 | 54.00 |
| 2021-0... | NORMAL | 273.50 | 289.00 | 2.50 | 3.20 | 697.00 | 932.00 | 1629.... | 229.20 | 4.00 | 50.09 | -1749.00 | -882.00 | 54.00 |
| 2021-0... | NORMAL | 251.40 | 296.00 | 5.40 | 5.80 | 1371.... | 1727.... | 3098.... | 233.90 | 2.90 | 50.07 | -735.00 | 548.00 | 53.00 |
| 2021-0... | NORMAL | 273.40 | 298.60 | 6.40 | 7.00 | 1782.... | 2113.00 | 3895.... | 236.30 | 5.50 | 50.04 | -1487.00 | 1310.00 | 51.40 |
| 2021-0... | NORMAL | 259.10 | 294.80 | 3.00 | 3.40 | 779.00 | 1003.... | 1782.... | 235.00 | 3.30 | 50.06 | -1809.00 | -736.00 | 50.10 |
| 2021-0... | NORMAL | 271.00 | 292.10 | 5.10 | 5.90 | 1406.... | 1746.... | 3152.... | 239.20 | 2.90 | 50.01 | -218.00 | 611.00 | 49.30 |
| 2021-0... | NORMAL | 259.20 | 298.50 | 5.20 | 5.70 | 1361.... | 1710.... | 3071.... | 231.70 | 2.70 | 49.99 | -2029.00 | 508.00 | 46.50 |
| 2021-0... | NORMAL | 272.10 | 294.40 | 9.40 | 10.50 | 2500.... | 3039.... | 5539.... | 239.60 | 12.70 | 50.01 | 353.00 | 2825.00 | 46.70 |

Displaying 1 to 20 of 190 items

This is something I would like to have as it gives the client a lot of flexibility with their data and then they can use it where they like, inside or outside my project. The other major functionality is the page that allows the changing of system modes:




Here it displays the current mode selected and a brief description of what each one does. This will be possible to integrate into my project by using their battery API that they provide.

## Computing Tools/Techniques

*Authentication with HTTP requests in Java*
https://stackoverflow.com/questions/3283234/http-basic-authentication-in-java-using-httpclient

This webpage gives a good example of using authentication before a POST HTTP request to an API. I need to be able to do this when the program sends changes to system modes to the GivEnergy API.

*Working with databases in Java*
https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

This tutorial explains how to connect to and access a database in Java. This is being used in my project for storing data that is used by the model that predicts performance of the panels.

*HTTP Requests in Java*
https://stackoverflow.com/questions/1359689/how-to-send-http-request-in-java

This webpage gives an example of an HTTP request in Java. I am using this a lot in my project in order to collect and send data to and from weather APIs and the GivEnergy API.

*Graphing in Java*
https://knowm.org/open-source/xchart/

This is a library that I am using to graphically display graphs in my desktop application. It allows you to create a very wide range or graphs and charts. A particular use for this in my project is displaying a forecast for the power output of the solar panels.

*Statistical Techniques*
https://www.khanacademy.org/math/ap-statistics/bivariate-data-ap/least-squares-regression/v/calculating-the-equation-of-a-regression-line

This video tutorial describes how to fit a line of best fit to some bivariate data. This is used for linear regression in my program and allows me to predict the performance of the panels with this bivariate data the program has collected into a database.

*GivEnergy API Documentation*
https://kb.givenergy.cloud/category.php?id=8

This site helped me develop my program and allowed me to understand how this API worked and the functionality that it has. This API is used to change settings of the system and to collect data about it.

*Weather Forecasting*
https://docs.solcast.com.au/?_ga=2.217890705.663793041.1631263837-335749219.1631263837#introduction

This website provides an API that gives a weather forecast for a specific Latitude and Longitude. The forecast includes values for GHI which is essential for forecasting performance of solar panels

*Processing SQL*
https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html

This tutorial website explains how to send query SQL statements into a database and then store the response from the database. This is used in my program every time it needs to access data from the databases that it is connected to.

*Variable-Length Arguments*
https://www.geeksforgeeks.org/variable-arguments-varargs-in-java/

This website explains how to use varargs in methods in Java. This is useful in my program as when I have a method that produces a graph, I can have a completely variable amount of data sets that I can add to the graph. This allows separate data sets to be compared on the same graph.

*Tabbed Pane*
https://docs.oracle.com/javase/tutorial/uiswing/components/tabbedpane.html

This tutorial explains the use of the Tabbed Panes in Java Swing. This is the layout that I am using in my Swing application, and it allows for a very structured GUI as different pages of information and user interactions can be spread out in a logical way

*Sending Emails with Java*
Send email using Java Program - GeeksforGeeks

This website explains the process of sending emails using Java. This is useful in my project as the program that runs on the Raspberry Pi server will need to have code that sends an evening report via email to the client.

*Finding the file path of the running program*
https://mkyong.com/java/java-read-a-file-from-resources-folder/

This website shows how to find the file path of the currently running java program. This is useful as when the program is built and distributed it then knows where to find and access data files relative to the currently running program.

*XChart*
https://knowm.org/open-source/xchart/

This is a library I am using to be able to display graphs and charts in my GUI to show data

*jCalendar*
http://www.java2s.com/Code/Jar/j/Downloadjcalendar14jar.htm

This is a library I am using to have another Java Swing component in my program that can be used to select dates graphically.

*java-json*
https://jar-download.com/artifacts/org.json

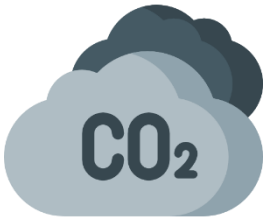
This is a library I am using to handle data in JSON format.

*Derby*
https://db.apache.org/derby/derby_downloads.html

This library is used to handle database connections.

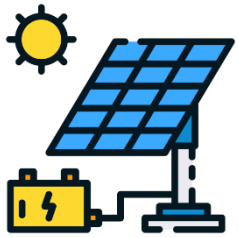
## Images/Icons

### CO2 clouds image



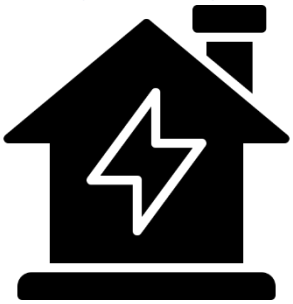

This image was sourced from https://cdn-icons.flaticon.com/png/512/573/premium/573027.png?token=exp=1650219193~hmac=060eecc8f309a2237167ee67f123f640 . This image is used in the Live Values tab in the box that displays the total CO2 saved today.

### Solar panel image



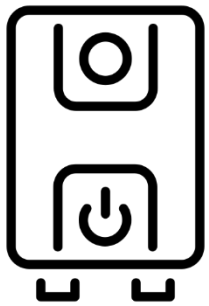
This image was sourced from https://cdn-icons-png.flaticon.com/512/2933/2933980.png . This image is used in the Live Values tab in the box that displays the live solar panel power output.

### House power image



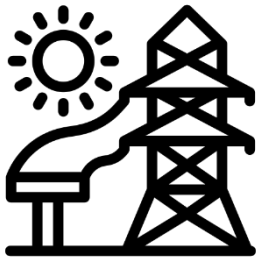
This image was sourced from https://cdn-icons.flaticon.com/png/512/3444/premium/3444114.png?token=exp=1650219302~hmac=3306031bf738f3f72ff5ca0f8dcaa3ed . This image is used in the Live Values tab in the box that displays the live power demand from the house.

### Inverter power image



This image was sourced from https://cdn-icons.flaticon.com/png/512/5006/premium/5006285.png?token=exp=1650219415~hmac=64bf0bfe11ff5e85c395fff996d61e60 . This image is used in the Live Values tab in the box that displays the live power through the inverter.

### Grid power image



This image was sourced from https://cdn-icons-png.flaticon.com/512/6875/6875839.png . This image is used in the Live Values tab in the box that displays the live power coming from or going to the grid

### Battery Image

This image was sourced from https://cdn-icons.flaticon.com/png/512/664/premium/664883.png?token=exp=1650219620~hmac=df99232c23501eb86572d1e980e18659 . This image is used in the Live Values tab in the box that displays the live battery SOC (state of charge).

#### *Loading animation GIF*

Here is a screenshot of the loading spinner animation GIF that the program uses to display when the program is doing intense processing:

This GIF was sourced from https://icons8.com/preloaders/ and was built using the customizer on the website.

## General Research

### *How the Solar Panels work*

The solar panels that are in this system are PV (Photovoltaic). This means they convert solar radiation from sunlight directly into electricity. The panels contain a semi-conducting material that is most likely silicon. When there is light on the panels, electrons are knocked free from the semi-conductor by the incident photons. The solar cell has positive and negatively charged semiconductors very close together and this helps to create an electric field around the cells. This electric field then provides a force on any charged particles such as the loose electrons that were previously freed from the semiconductor. This force initiates DC electricity through the panels and this current will flow through wires to get to the Inverter.

Source: https://solect.com/the-science-of-solar-how-solar-panels-work/

### *Calculating Solar Radiation on Tilted Surfaces*

https://www.pveducation.org/pvcdrom/properties-of-sunlight/solar-radiation-on-a-tilted-surface

This website explains how to calculate the component of solar radiation that acts on a surface (i.e the solar panels) at a specific tilt and azimuth. This is needed in the calculations that I have to do when working out how much of the GHI (radiation that falls onto a horizontal surface) is actually incident to the plane of the panels.

### *Modelling the Position of the Sun*

The position of the sun in the sky from any location on earth can be represented by two values: elevation and azimuth. Both these values can be measured in degrees. Elevation is the apparent 'height' of the sun above the horizon measured in degrees. Azimuth is the angle of the sun measured from due north in the horizontal plane (source: https://www.sciencedirect.com/topics/engineering/solar-azimuth-angle#:~:text=Solar%20azimuth%20angle%20is%20defined,plane%20and%20due%20south%20direction.)

These two values are needed when the program needs to calculate the proportion of GHI incident to the plane of the surface of the solar panels. The program performs calculations to determine the position of the sun at any time of the day and any day of the year. These calculations take into

account the number of days into the year, the time of day and the exact latitude and longitude. The formula for the sun's elevation is given below:

$$\alpha = \sin^{-1}[\sin\delta\sin\varphi + \cos\delta\cos\varphi\cos(HRA)]$$

The formula for the Sun azimuth is given below:

$$Azimuth = cos^{-1}[\frac{sin\delta cos\varphi - cos\delta sin\varphi cos(HRA)}{cos\alpha}]$$

Where: φ is the latitude in degrees, δ is the declination angle in degrees (https://www.pveducation.org/pvcdrom/properties-of-sunlight/declination-angle) and HRA is the Hour Angle (https://en.wikipedia.org/wiki/Hour_angle#:~:text=In%20astronomy%20and%20celestial%20navigation,rotations%20depending%20on%20the%20application.)

Full calculations for these formulas can be found at https://www.pveducation.org/pvcdrom/properties-of-sunlight/the-suns-position . The program will perform these calculations to get a highly accurate position of the sun which it can then use in the calculations of incident light intensity on the panels.

### *Factors Affecting Solar Generation*

Factors other than weather which affect the performance of solar panels include location, tilt, azimuth, and potential shading effects on the panels. All of these are major parameters for the effectiveness of the system and if these factors do not have optimal values, then the system will be compromised. The location of the solar panel site affects performance as at different latitudes the incident light onto the earth will be at different angles and provide less power, which is why it is warmer at the equator. The tilt of the panels also makes a significant impact. During the summer, the sun is higher in the sky so a smaller angle between the panel and the ground would be preferable. During the winter, the sun tends to be lower in the sky so it would be preferable to have a large tilt so that the panel is more perpendicular to the incident rays. The Azimuth angle is the angle from 000° North and the direction that the panel is facing. In the northern hemisphere it is preferable for a panel to point south as this is where the sun will be in the sky. Also, if the panels are affected by shading affects by trees or buildings then this will act

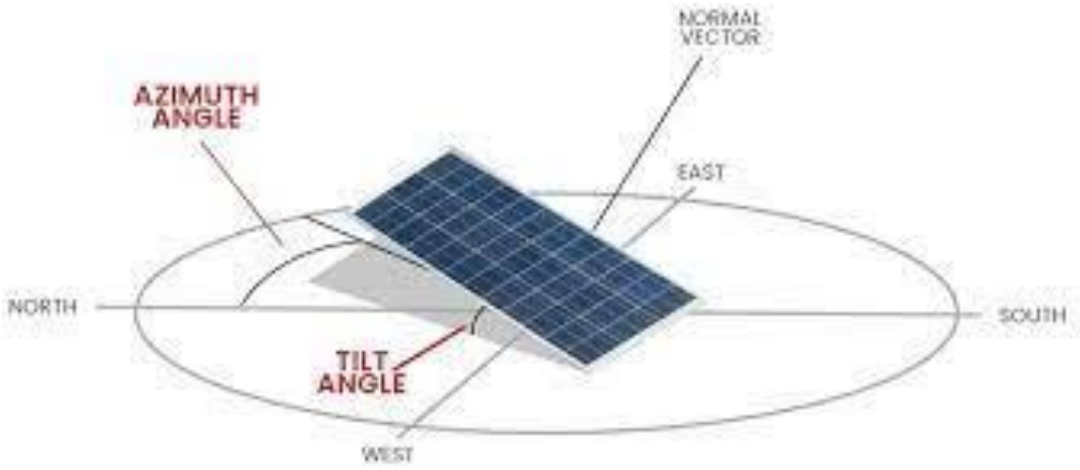


Image source: https://www.prostarsolar.net/support/blog/how-to-set-solar-panel-angle-to-sun.html

### *GHI (Global Horizontal Irradiation)*

GHI is the total amount of shortwave radiation incident to a horizontal surface on the ground. GHI is crucial to all calculations of predicting solar panel performance. The calculation of this value involves two other values – DNI (Direct Normal Incidence) and DHI (Diffuse Horizontal Irradiation). The DNI is the amount of solar radiation received per unit area by a surface that is perpendicular to the rays that come from the exact angle of the sun (tilted directly towards the sun). DHI is the amount of radiation received per unit area by a surface but does not arrive via a direct path from the sun as it has been scattered by molecules and particles in the atmosphere and comes equally from all directions. The relationship between GHI, DNI, and DHI is given by Global Horizontal (GHI) = Direct Normal (DNI) X cos(θ) + Diffuse Horizontal (DHI).

### *The Inverter and Battery*

The purpose of an inverter is to convert DC electricity into AC electricity. In the context of solar panels, the panels will only produce DC, but buildings always need AC to power appliances. This means that an inverter must be used in order for the electricity to be of the correct form. The inverter in my client's system is a Hybrid inverter. This means that is has the capability control the charge of battery that can store the charge from the panels. The inverter is connected to the internet via Wi-Fi, and this means that the user can change the modes of the system. The modes change the behaviour of the inverter, and the inverter has the capability to control the SOC of the battery and charge/discharge it at specific times of the day.

https://www.fallonsolutions.com.au/solar/information/what-is-a-solar-inverter-and-how-does-it-work

### *GivEnergy API*

GivEnergy is the company that make the Inverters and the battery for the system. Their website and mobile app provide controls for the system and present all the data collected from the system. The inverter is able to collect all data from the system including data from the panels and battery. This data is then sent to the GivEnergy servers so that it can be accessed by the GivEnergy website and app. Also, when the user uses the website to change the mode of the system the inverter receives this request and makes those changes. GivEnergy also provide access to all this data via their API. This API provides a JSON response and uses Session Cookies to allow retention between API calls. This means that a login API must be sent when first accessing the API to initialise the connection and after that you save the session cookie and use it in all other API calls that you may need in that session. There are two main APIs that they provide – the main GivEnergy API with the Base URL of http://www.givenergy.cloud/GivManage/api/ and the Battery API with the Base URL of https://api.givenergy.cloud/. The main API provides all data collected from the inverter and panels and the battery API provides specific and detailed data about the battery and is also the API that is used to send requests to change the modes of the system. The battery API is therefore very essential in my project.

Battery API documentation: https://kb.givenergy.cloud/article.php?id=54

Main GivEnergy API documentation: https://kb.givenergy.cloud/article.php?id=5

### *Octopus Agile Tariff*

The Agile Tariff provides variable half-hourly energy prices for importing and exporting energy from the grid. These prices are published by Octopus Energy and are based on the wholesale price of electricity which changes due to weather and how much energy the country is producing. For

example, if it happened to be a particularly windy day then wind turbines would be generating lots of free energy – this would cause the wholesale prices to drop and therefore Octopus Energy would have periods of cheaper electricity prices. If there is excess energy across the UK then the Octopus Agile prices would decrease, it is even possible for the importing prices to be negative, so you effectively get paid to import electricity. This illustrates why my program needs to properly analyse the prices so that it can make smart decisions about when to import electricity when the prices are cheap.

#### *Octopus API*

The Octopus API is used to collect data about electricity sourced from the grid. The most important endpoint of this API is the Agile Prices. This endpoint returns agile prices between a specified period for any point since records started. At the end of every day, usually between 4pm and 8pm, Octopus Energy publish the prices for the next day.

https://www.guylipman.com/octopus/api_guide.html

#### *Mathematical Matrices Operations*

There are 3 complex mathematical Matrix operations that I need to use in my program. The first is calculating the inverse of a matrix. This is needed so that I can solve matrix equations by using matrix multiplication. The process of finding the inverse of a matrix is explained here: https://www.mathsisfun.com/algebra/matrix-inverse.html . Matrix multiplication is the second matrix operation that I am using in my program and it is used to find the matric product of multiple matrices. The process of matrix multiplication is explained here: https://www.mathsisfun.com/algebra/matrix-multiplying.html . The last operation the program performs is finding the determinant of a matrix. The determinant is a calculated value that can represent a matrix. It is needed when finding the inverse of a matrix. The process of calculating the determinant of a matrix is explained here: https://www.mathsisfun.com/algebra/matrix-determinant.html

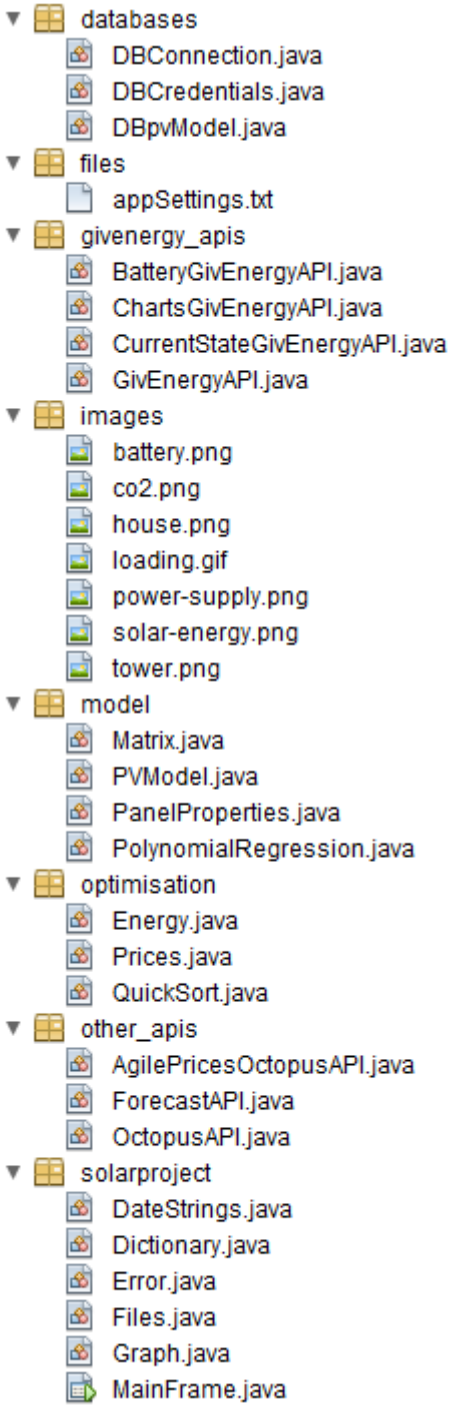
# Design

## Technical Skills

### Classes

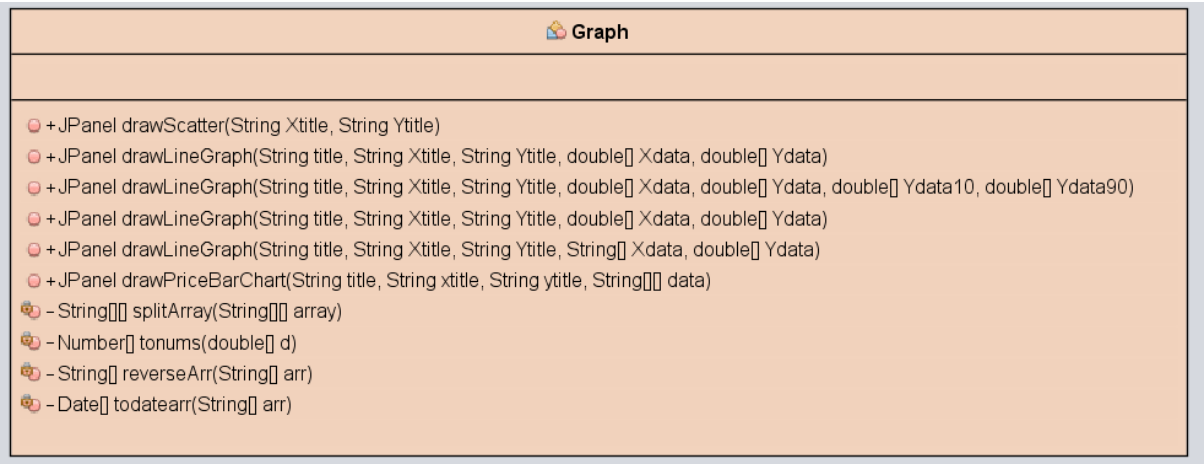
I have used classes to logically structure my program. Classes are sorted into packages by their general use, for example all classes that deal with Databases are stored in the same package. See a

screenshot of all classes sorted into their packages below:



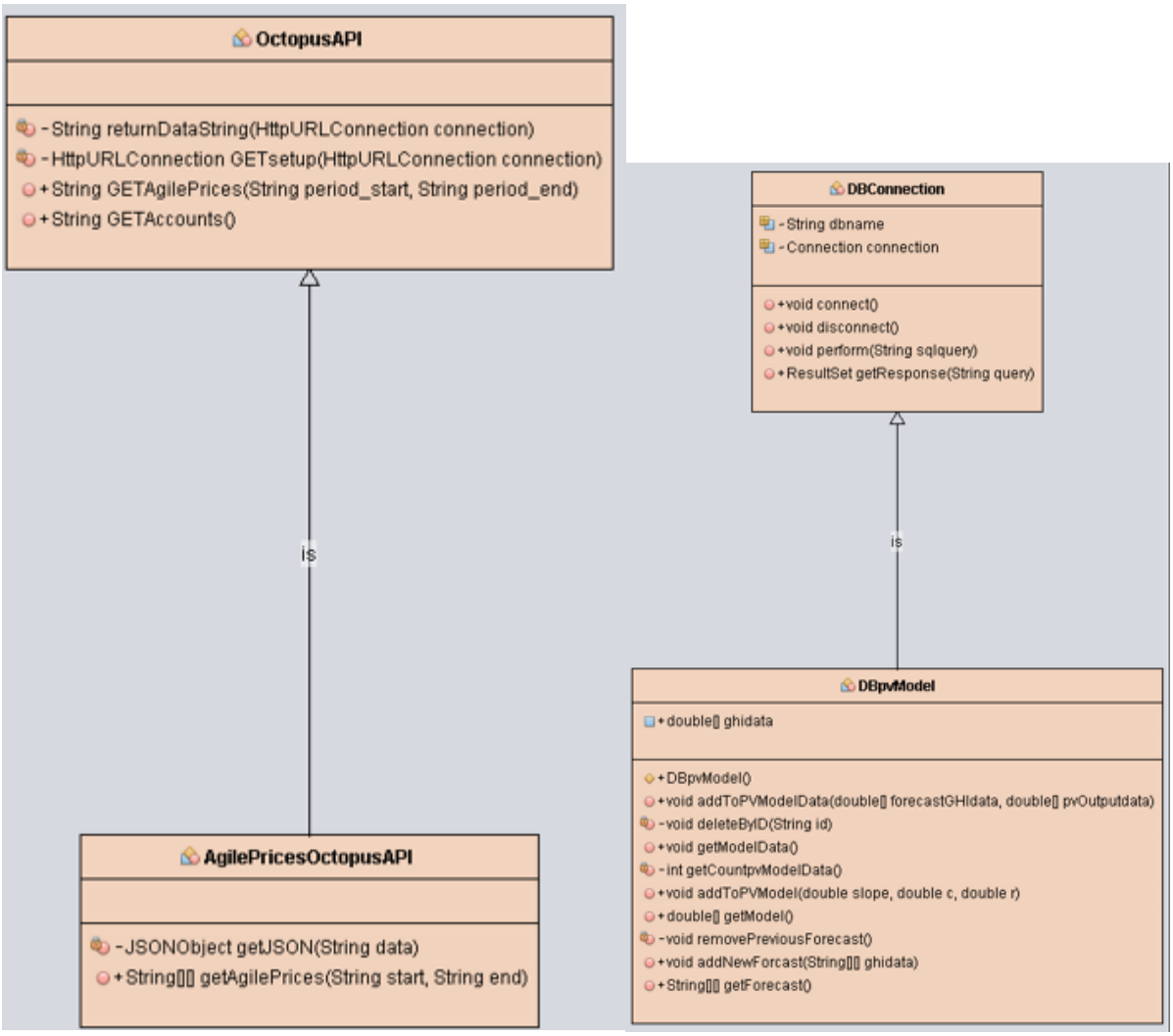

## Overloading

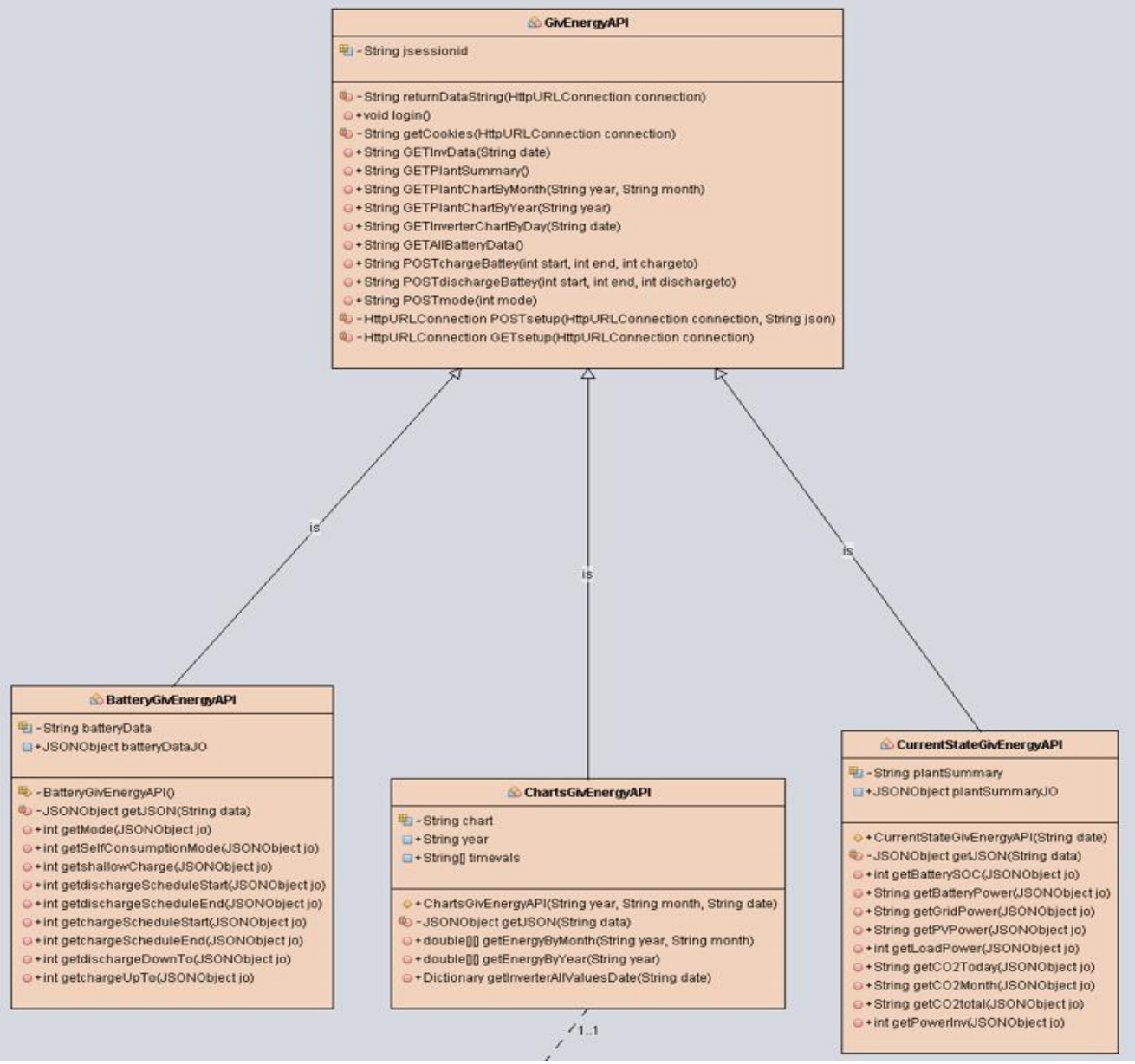
I have used overloading in my Graph class. I am generating many graphs that display many different datasets including graphs that have multiple datasets on the same axis. I have overloaded versions of methods to accommodate different variations in graphs the program might need to produce. The UML diagram for this class shows how the drawLineGraph() method is overloaded (the third drawLineGraph() uses varargs for the last parameter but this does not show on the diagram).




## Inheritance

There are certain classes in my program that are very closely related and have functionality/properties that are a subset of another class's functionality/properties. To address this I have implemented inheritance between certain classes. An example is the classes that deal with GivEnergy API requests. The parent class is GivEnergyAPI.java and BatteryGivEnergyAPI, ChartsGivEnergyAPI and CurrentStateGivEnergyAPI are all child classes to this class. The child classes all collect slightly different data but they all access it through the same API – with different endpoints. This relationship works very well with inheritance. The following UML diagrams represent the relationships between all uses of inheritance in my program:


OctopusAPI
- String returnDataString(HttpURLConnection connection)
- HttpURLConnection GETsetup(HttpURLConnection connection)
+ String GETAgilePrices(String period_start, String period_end)
+ String GETAccounts()
DBConnection
- String dbname
- Connection connection
+ void connect()
+ void disconnect()
+ void perform(String sqlquery)
+ ResultSet getResponse(String query)
is
is
AgilePricesOctopusAPI
- JSONObject getJSON(String data)
+ String[][] getAgilePrices(String start, String end)
DBpvModel
+ double[] ghidata
+ DBpvModel()
+ void addToPVModelData(double[] forecastGHIdata, double[] pvOutputdata)
- void deleteByID(String id)
+ void getModelData()
- int getCountpvModelData()
+ void addToPVModel(double slope, double c, double r)
+ double[] getModel()
- void removePreviousForecast()
+ void addNewForcast(String[][] ghidata)
+ String[][] getForecast()




## Overriding

I have used Overriding in my project so that I can multithread methods. When the program is initially opening up a lot of methods are called because the program needs to collect data from APIs and databases, and it needs to generate graphs etc. so that when the project is opened the user has full functionality. However, this means that there are a few seconds of processing that causes the GUI to be unresponsive. I have multithreaded all these processes so that they run seamlessly while a loading animation displays. When all the processing is complete the loading spinner disappears, and the user has full access to the GUI. The run() method is overridden with all my code and that method is originally from the Runnable class.

```
@Override
public void run(){
```

```
implements Runnable{
```

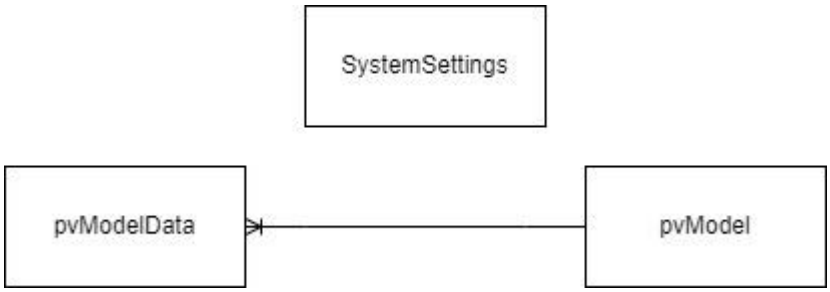
## Complex database model

I am using a relational embedded SQL database in my project. To access the database, I need to use SQL functions. I have methods within DBpvModel class that form and send requests using the DBConnection class, which is its parent class. An example of a such method is below:

```
public void addToPVModelData(double[] forecastGHIdata, double[] pvOutputdata) throws SQLException{
    LocalDate date = LocalDate.now();
    String query;
    for(int i =0;i<forecastGHIdata.length;i++){
        if((pvOutputdata[i]==0)|(forecastGHIdata[i]==0)){
            continue;
        }
        query = "INSERT INTO pvModelData(forecastGHI,pvOutput,Date)"+"VALUES ("+forecastGHIdata[i]+","+pvOutputdata[i]+",'"+date+"')";
        perform(query);
    }
}
```

This method adds data to a table in the database. The query String is the SQL that is used to add the data.

Here is an entity relationship diagram for my database:




pvModelData stores every data point I have collected, and each point has a foreign key that matches with the primary key of pvModel. pvModel stores records of the key parameters used in my model, meaning my program does not have to recalculate the model each time it needs to use it – it only needs to fetch the necessary model values from this table.

SystemSettings is a table that serves a completely separate purpose than the other 2 tables and it is used to temporarily store the charging and discharging periods for the next day as they are always calculated the day before using the forecast for the next day. This table would be updated every day for the next day so the values in it are always correct.

## Calling parameterised Web service APIs and parsing JSON to APIs

My project accesses three main APIs: The GivEnergy API, the Octopus API and the Solcast API. The GivEnergy API has many endpoints and provides data about the solar panels, inverter, and battery. The Octopus API provides data about the agile price of importing electricity. The Solcast API provides solar radiation forecasts and estimated actuals of solar radiation which I can use to build a model. I have made use of inheritance for the GivEnergy API and the Octopus API due to the large number of endpoints that they contain. All the APIs return JSON responses and so I am using a library that can read values from JSON format in order to efficiently make use of the response. Here is a screenshot of all methods needed to collect data from an endpoint:

```
public void login() throws MalformedURLException, IOException{
    HttpURLConnection connection;
    URL url = new URL("https://www.givenergy.cloud/GivManage/api/login?account=                &password=                ");
    connection = (HttpURLConnection) url.openConnection();
    connection.setRequestMethod("POST");
    jsessionid = getCookies(connection);
    connection.setConnectTimeout(5000);
    connection.setReadTimeout(5000);
    System.out.println(jsessionid);
}
```

```
private HttpURLConnection GETsetup(HttpURLConnection connection) throws ProtocolException{
    connection.setRequestMethod("POST");
    connection.addRequestProperty("Cookie", "JSESSIONID="+jsessionid);
    connection.setConnectTimeout(5000);
    connection.setReadTimeout(5000);
    return connection;
}
```

```
public String GETInvData(String date) throws MalformedURLException, I
    HttpURLConnection connection;
    URL url = new URL("https://www.givenergy.cloud/GivManage/invData/                -date+""+"+?page=1&rows=1000");
    connection = (HttpURLConnection) url.openConnection();
    connection = GETsetup(connection);
    String response = returnDataString(connection);
    return response;
}
```

## Dictionary Data Structure

I am implementing a Dictionary data structure to allow easy and fast access to arrays that contain many values to represent on a graph. When I need to display many graphs on the same axis I have a dictionary that pairs a name for a dataset with the dataset. The Dictionary class uses the inbuilt hash-map in Java and acts as a layer of abstraction on top of that so that only the functionality needed by the rest of the program is available with the class.

```
dict.add("pacExport", Arrays.stream(pacExportList.toArray()).mapToDouble(num -> Double.parseDouble(num.toString())).toArray());
dict.add("pacImport", Arrays.stream(pacImportList.toArray()).mapToDouble(num -> Double.parseDouble(num.toString())).toArray());
dict.add("ppv", Arrays.stream(ppvList.toArray()).mapToDouble(num -> Double.parseDouble(num.toString())).toArray());
dict.add("loadpower", Arrays.stream(loadPowerList.toArray()).mapToDouble(num -> Double.parseDouble(num.toString())).toArray());
dict.add("batpoweractual", Arrays.stream(batPowerActualList.toArray()).mapToDouble(num -> Double.parseDouble(num.toString())).toArray());
return dict;
```

The data sets can be accessed through .get(key):

```
chartdata.get("pacExport"),chartdata.get("pacImport"),chartdata.get("ppv")
```

## Matrix Data Structure

The matrix data structure is central to the complex mathematical model. It is used to calculate the coefficients of the mathematical function for the model. In a matrix, data is stored in an n*m sized table. I have represented this with a multi-dimensional array which holds all the data within the matrix. Matrices are needed as they allow complicated mathematical tasks to be largely simplified and the processing behind matrices are suited to a computer as there tends to be many steps. The matrix data structure is implemented through a class and matrix objects can be created, representing a matrix with any dimensions. Multiplying matrices and finding the inverse of matrices are the two main operations of the Matrix class. However, both tasks have many mathematical sub-steps along the way. This website explains the process I am using with matrices to perform polynomial regression: http://polynomialregression.drque.net/math.html

This website explains finding the inverse of a matrix (although their example uses a small matrix which needs many less steps than the matrices used in my program):

https://www.mathsisfun.com/algebra/matrix-inverse.html

This website explains matrix multiplication:

https://www.mathsisfun.com/algebra/matrix-multiplying.html

## Complex Mathematical Model

The complex mathematical model I am using in my project aims to predict the power output of the solar panels from a given weather forecast, over the course of the next few days. It achieves this using a statistical technique called polynomial regression on a dataset which the program is collecting. As more data is collected the model will be refined, forming a basis for the machine learning aspect of the program. The model represents the relationship between two variables: solar radiation incident on the panels, and solar panel power output. The program is building a dataset that contains hundreds of bivariate data pairs, forecasted GHI and power output for a given point in time. These data points are stored in a table in the database. The model is a function of forecasted GHI, so you can input any forecasted GHI and get an output for the expected power output of the solar panels. This is how the model is used to predict power output – a forecast is collected and passed straight into the model which returns a power output prediction. When the model needs to be calculated the program performs polynomial regression on all the data points. This effectively creates a curve of best fit through the data set that best describes the relationship. Since the relationship is not entirely linear it needs to be polynomial regression and not linear regression. In order to perform polynomial regression, the program uses matrices. Here is an image of the matrices required for the main calculation (source: http://polynomialregression.drque.net/math.html):

$$\begin{bmatrix} n & \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 & \cdots & \sum_{i=0}^{n} x_i^m \\ \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 & \sum_{i=0}^{n} x_i^3 & \cdots & \sum_{i=0}^{n} x_i^{(m+1)} \\ \sum_{i=0}^{n} x_i^2 & \sum_{i=0}^{n} x_i^3 & \sum_{i=0}^{n} x_i^4 & \cdots & \sum_{i=0}^{n} x_i^{(m+2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{n} x_i^m & \sum_{i=0}^{n} x_i^{(m+1)} & \sum_{i=0}^{n} x_i^{(m+2)} & \cdots & \sum_{i=0}^{n} x_i^{2m} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} y_i \\ \sum_{i=0}^{n} x_i y_i \\ \sum_{i=0}^{n} x_i^2 y_i \\ \vdots \\ \sum_{i=0}^{n} x_i^m y_i \end{bmatrix}$$

This is a matrix multiplication, the first matrix contains sums of squares, cubes etc of all the x values in the data points i.e., the GHI forecast data. The second matrix contains one column and the elements within it are the coefficients of the model – these values are what is ultimately calculated. The final matrix on the right-hand side is another single column matrix that contains sums of x and y data. The full derivation of this matrix equation can be found on this website: http://polynomialregression.drque.net/math.html

In order to calculate the values of the coefficients in the second matrix the left most matrix needs to be “left multiplied” onto the other side of the equation. This involves finding the inverse of the left most matrix and multiplying it with the right most matrix. The result will be a matrix only containing the values of the coefficients of the polynomial function, which is used directly in the model. Finding the inverse of the matrix and multiplying it with another matrix is quite complicated and is handled by my user defined Matrix class.

## Multithreading

I am using multithreading in my program to improve the efficiency and user experience when using my program. It is used so that while processing is happening in the background, loading spinner animations can be displayed over the project. This prevents the GUI seeming as if it is ‘frozen’ while lots of processing is happening in the background. I have also used multithreading to handle the server that is used to run the embedded database with my project. This server needs to be started when the program is started and stopped when the program is closed. This requires multithreading as these processes can take several seconds and would clearly impact on the performance and user experience when the program is opening or closing. Another use for multithreading is to repeatedly update the live system values on the first tab of the GUI. This tab displays live values from the system for example the current power output of the solar panels. This is updated every two minutes on a multithreaded timer.

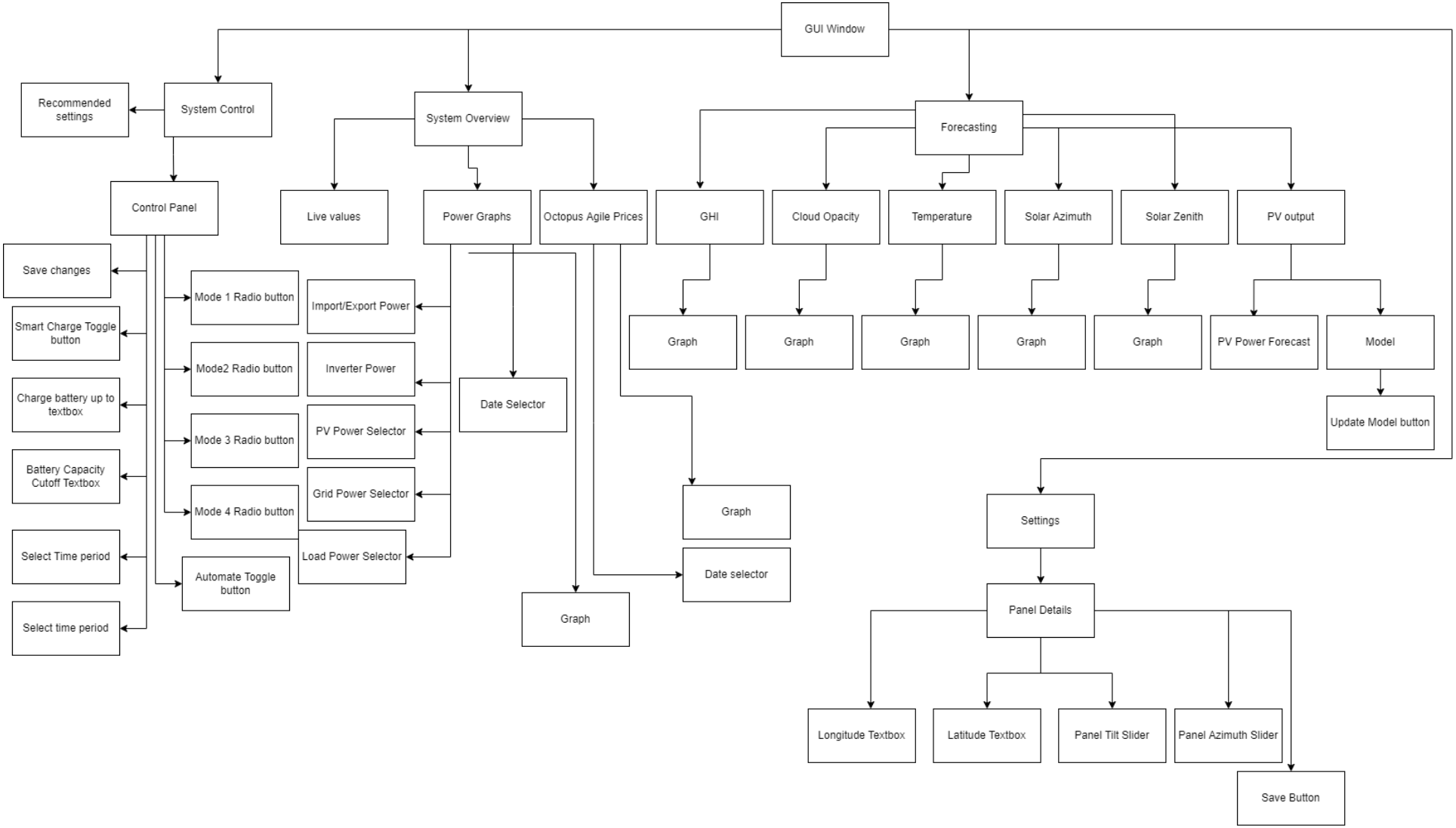
## Recursive Algorithms

I have used a recursive method to calculate the determinant of a matrix. This is a step that is needed when calculating the complex mathematical model. The nature of matrices means that as you get higher dimensional n*n matrices the calculations get longer and longer as the determinant of each minor needs to be calculated, but this minor will probably be larger than a 2*2 matrix so a matrix of minors will have to be created for that minor and so on. This problem is very much suited to a recursive algorithm due to the nature of the calculations required. Unfortunately, the problem has a time complexity of O(n!) so for matrices bigger than a 12*12 it tends to take far too long. I only ever need to use 5*5 matrices at most so the processing time is not a problem. Here is a screenshot of the method I am using to calculate the determinant. The exit condition is when the matrix has been refined down to a 2*2:

```
private double determinant(Matrix matrix){

    if (matrix.rows == 2){
        return matrix.matrix[0][0] * matrix.matrix[1][1] - matrix.matrix[0][1] * matrix.matrix[1][0];
    }
    double det = 0;
    for(int i=0;i<matrix.rows;i++){
        det+= Math.pow(-1, i)*matrix.matrix[0][i]*determinant(submatrix(matrix,0,i));
    }
    return det;
}
```

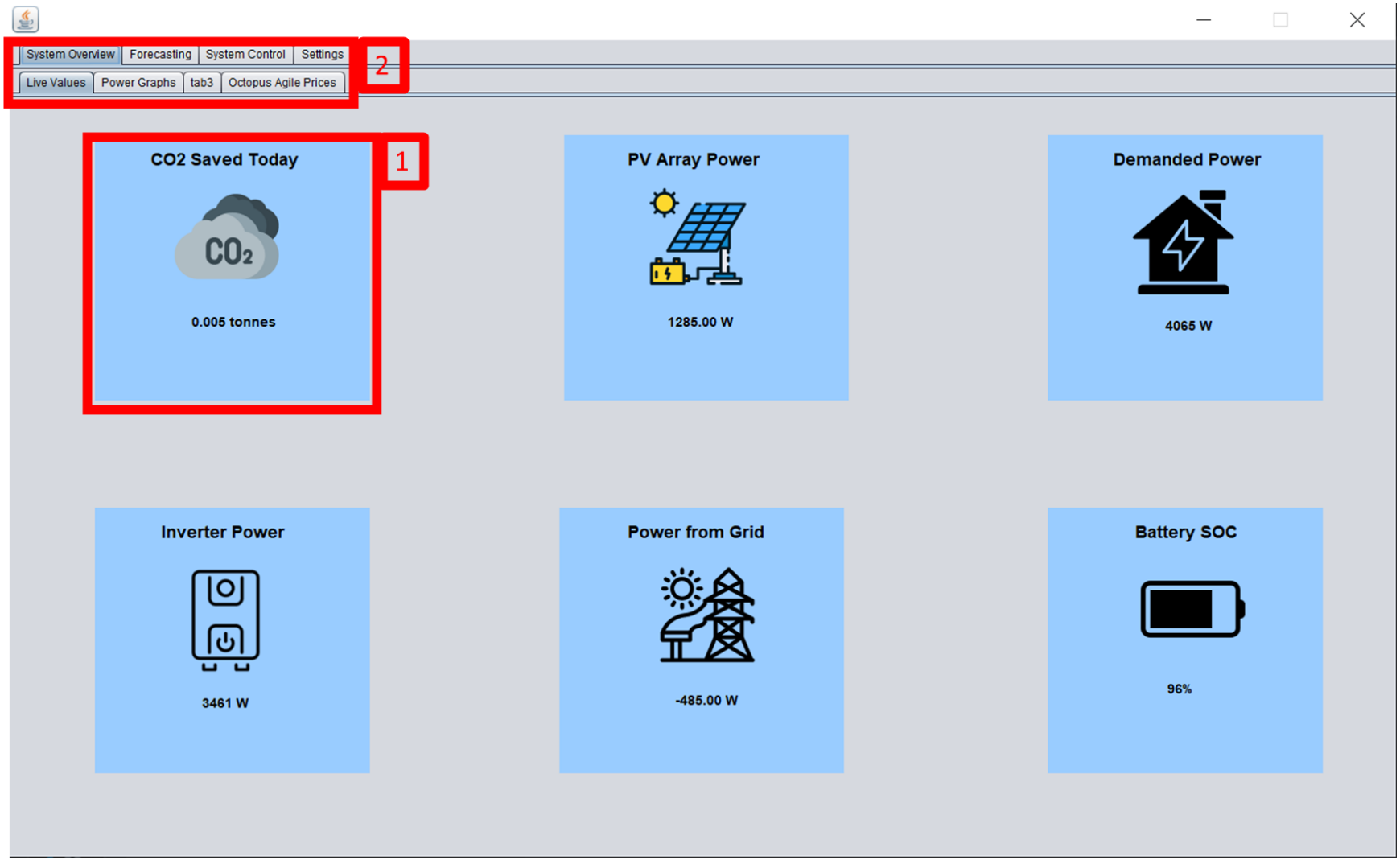
## UI Design Overview



# UI Screenshots

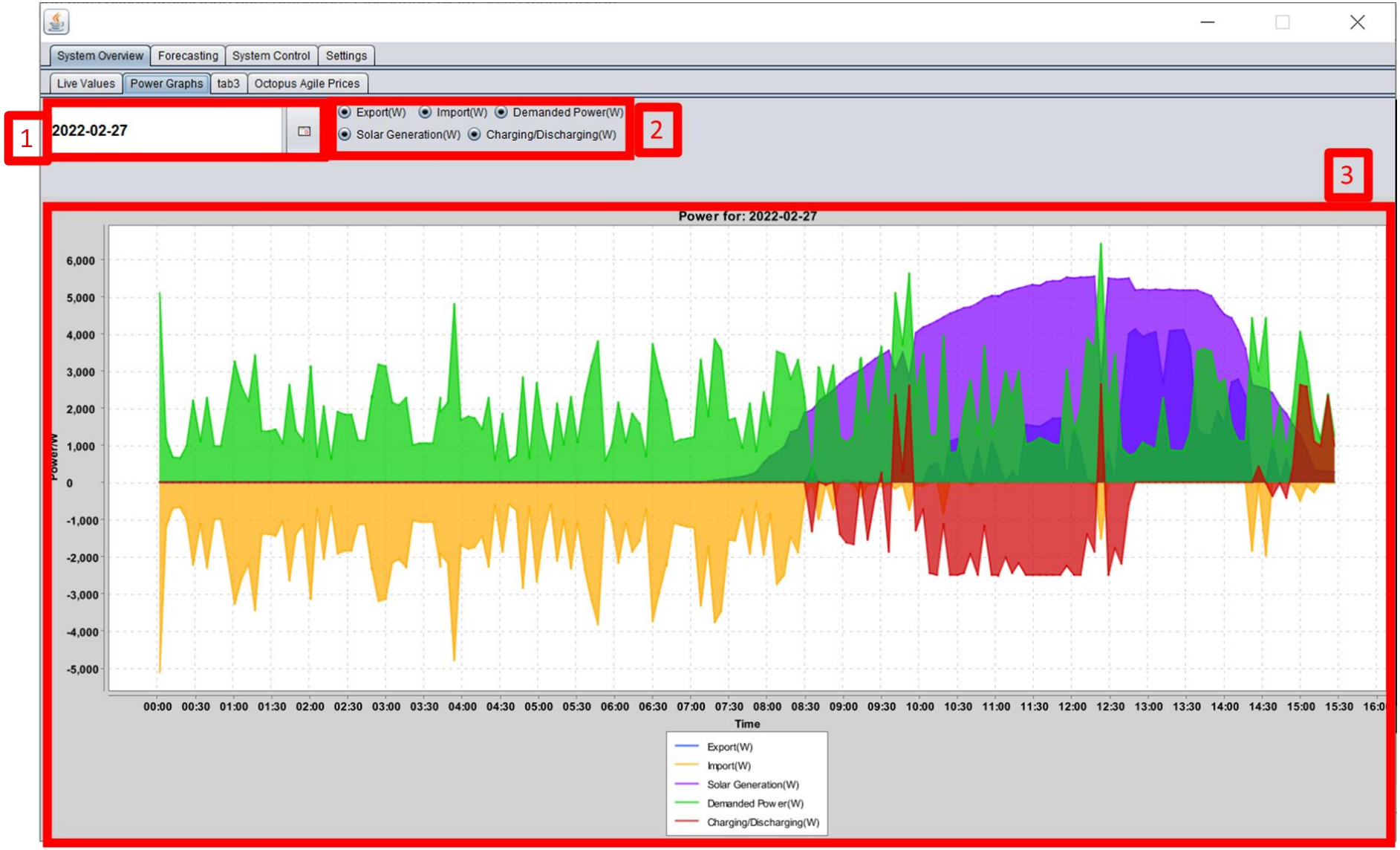
## System Overview tab

### *Live Values tab*



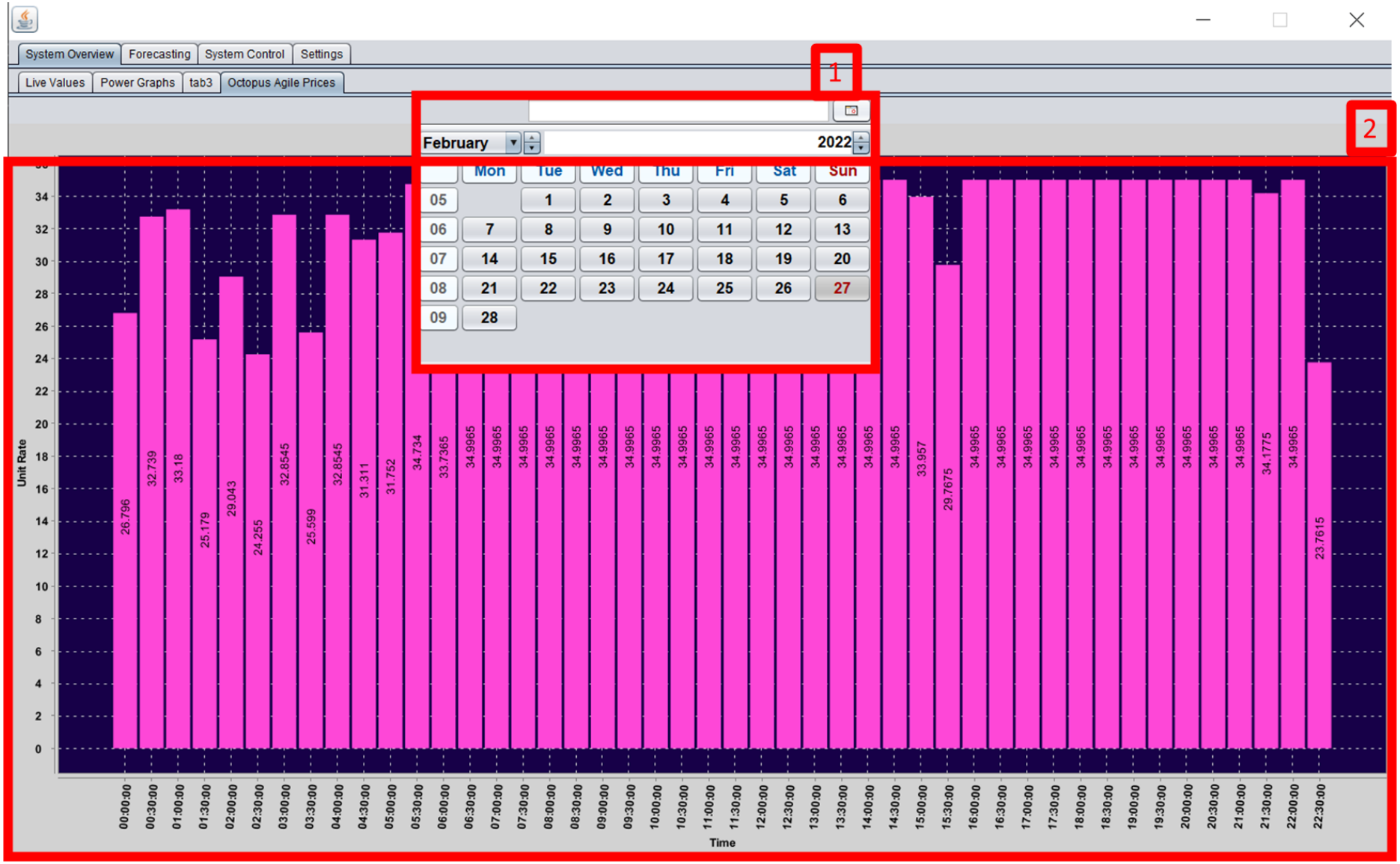
1) This blue panel along with the other 5 show live values from the system and are updated every 2 minutes
    a. These values are collected using the GivEnergy API with the CurrentStateGivEnergyAPI class and displayed onto the GUI with MainFrame.java class. The CurrentStateGivEnergyAPI class inherits from GivEnergyAPI and GivEnergyAPI runs a method called GETPlantSummary() which returns a JSON of the live data values. CurrentStateGivEnergyAPI has methods which return individual values from this JSON data response such as battery SOC or solar PV power.
2) This is a tab selector where the user can navigate the whole GUI

*Power Graphs tab*



1) This is a date picker component that is used to select from what date the data should be from on the graph
    a. When a new date is selected is runs the method drawPowerGraph() in the MainFrame which uses the Graph class to produce the correct graph from the right date and with the right data sets on it
    b. The data is collected from the GivEnergy API using inverterValuesDictionary() method in the ChartsGivEnergyAPI class which adds all the data to a Dictionary object for easier access to the data
2) These radio buttons control which datasets are shown on the axis
    a. These radio buttons are checked for their state when a new graph needs to be generated
3) This is the graph that is displayed and controlled by the other highlighted features

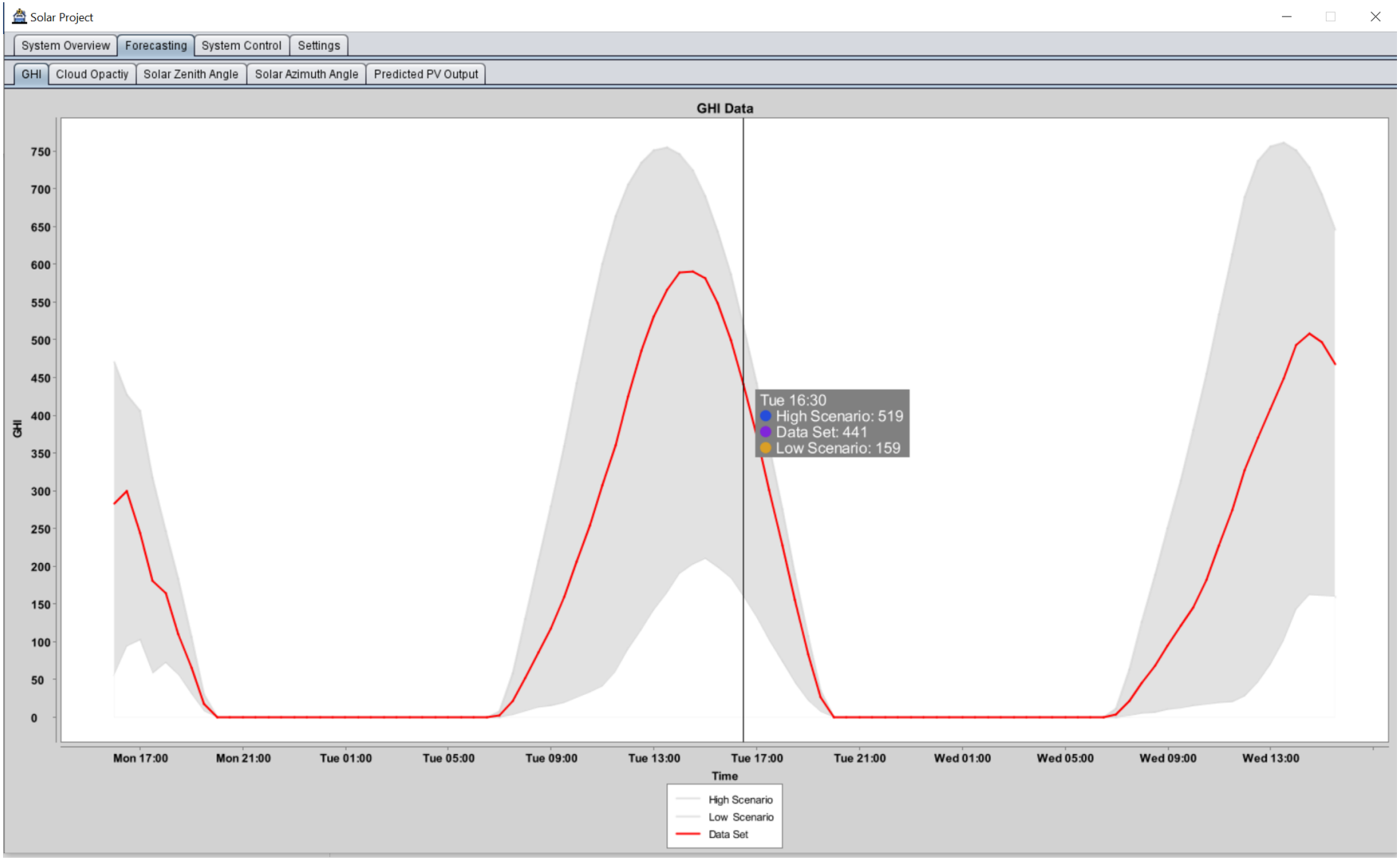
*Octopus Agile Prices tab*



1) Date picker component so that the user can select what date of electricity prices they want to see
    a. When the value of this component is changed a method called octopusDatePickerPropertyChange() is run and it uses objects of the Graph class and the AgilePricesOctopusAPI class to collect the data from the Octopus Energy API from the chosen date and display it as a bar chart.
2) This is the bar chart that displays the electricity prices for the selected date. The chart is a JPanel object and is produced by the drawBarChart() method in the Graph class.

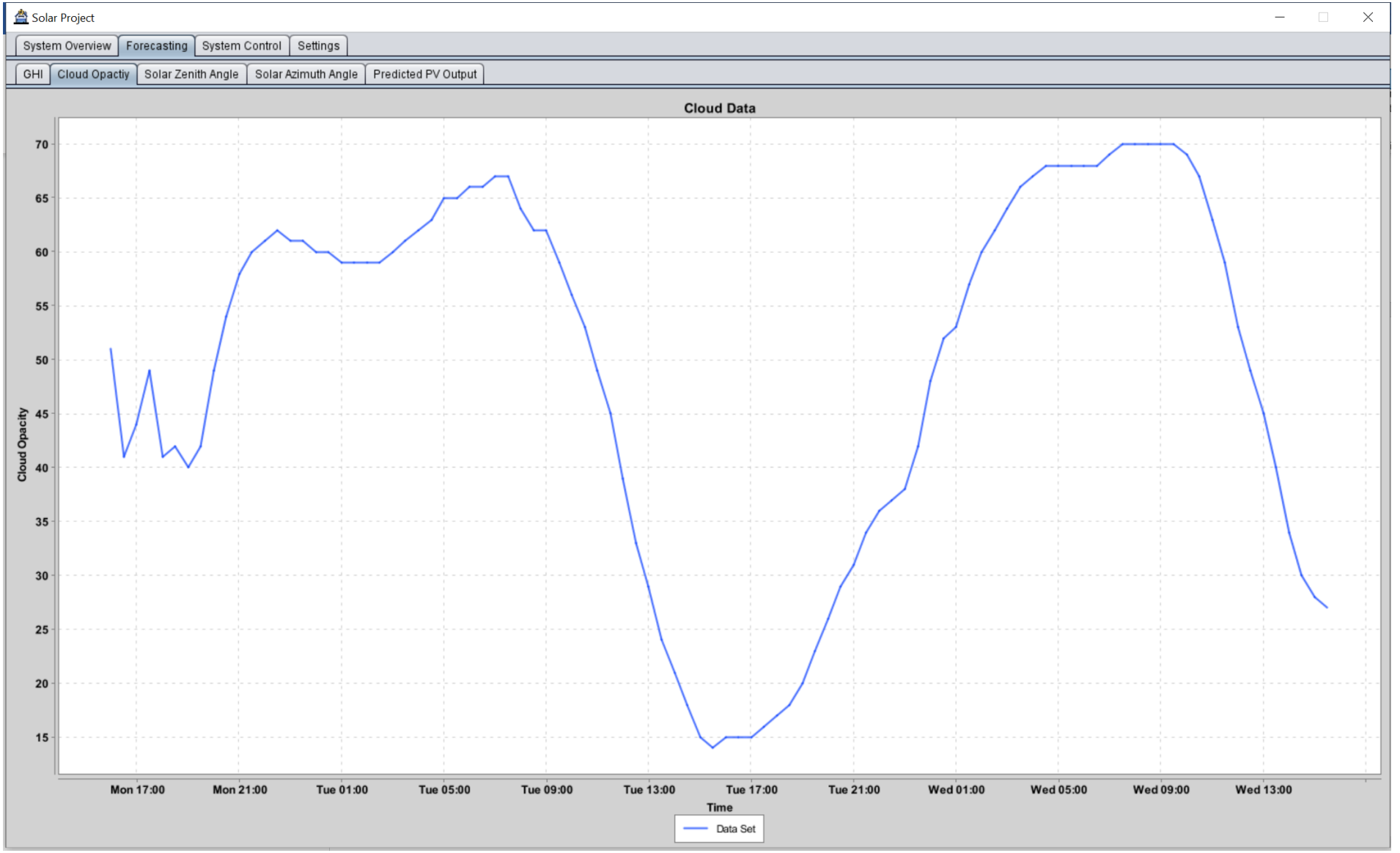
## Forecasting

*GHI tab*


Solar Project
System Overview
Forecasting
System Control
Settings
GHI
Cloud Opactiy
Solar Zenith Angle
Solar Azimuth Angle
Predicted PV Output
GHI Data
GHI
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
Tue 16:30
High Scenario: 519
Data Set: 441
Low Scenario: 159
Mon 17:00
Mon 21:00
Tue 01:00
Tue 05:00
Tue 09:00
Tue 13:00
Tue 17:00
Tue 21:00
Wed 01:00
Wed 05:00
Wed 09:00
Wed 13:00
Time
High Scenario
Low Scenario
Data Set

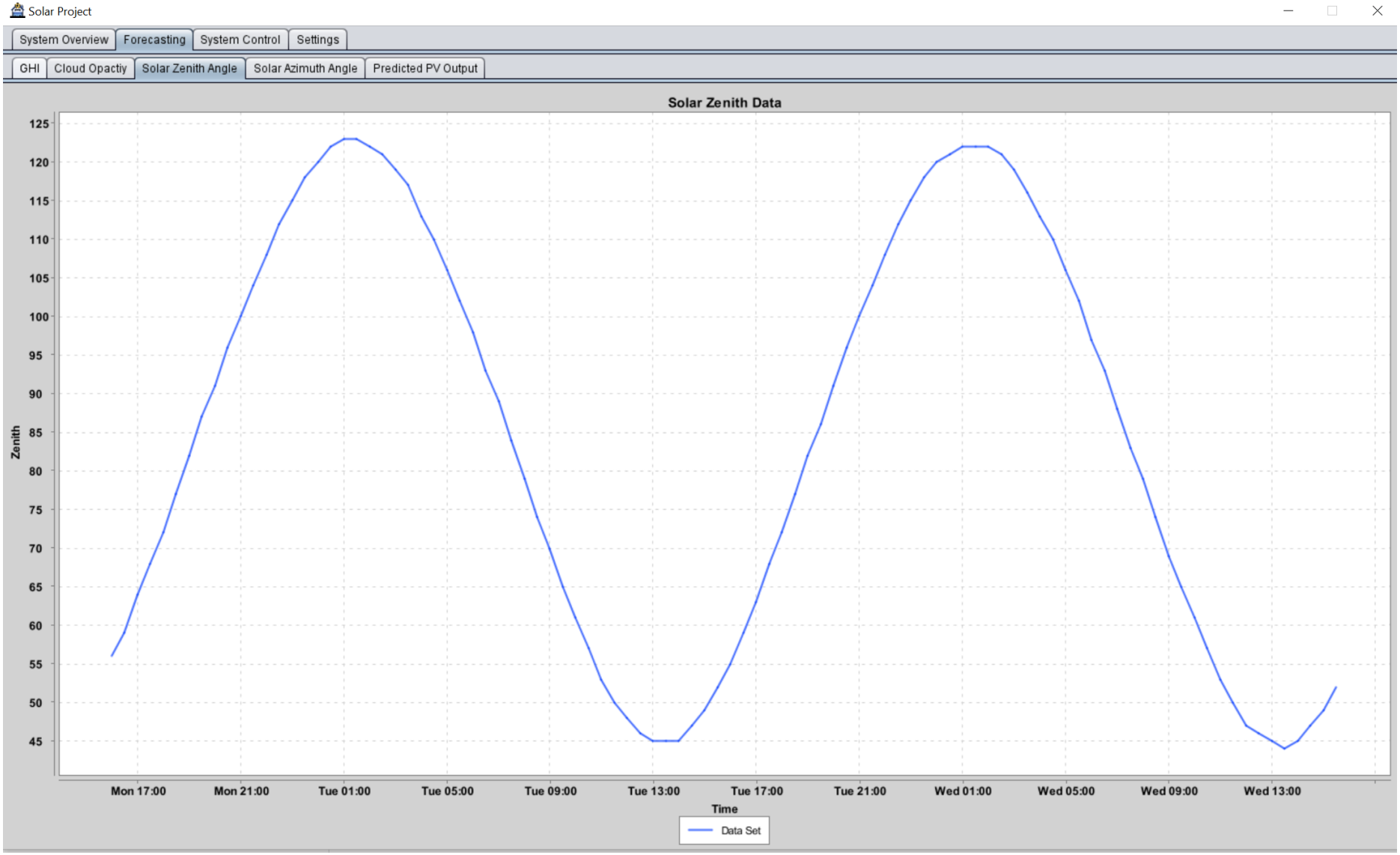
This tab displays a single graph for the forecasted GHI (solar radiation) for the next 48 hours. This graph is added when the program is first loaded up with the initForecastGraphs() in the MainFrame class. The method drawLineGraphGHI() from the Graph class is run within initForecastGraphs() to specifically produce this graph and it is returned as a JPanel that can be added to the tab. The data for the graph is from the GETGHI(), GETGHI10() and GETGHI90() methods in the ForecastAPI class.

*Cloud Opacity tab*



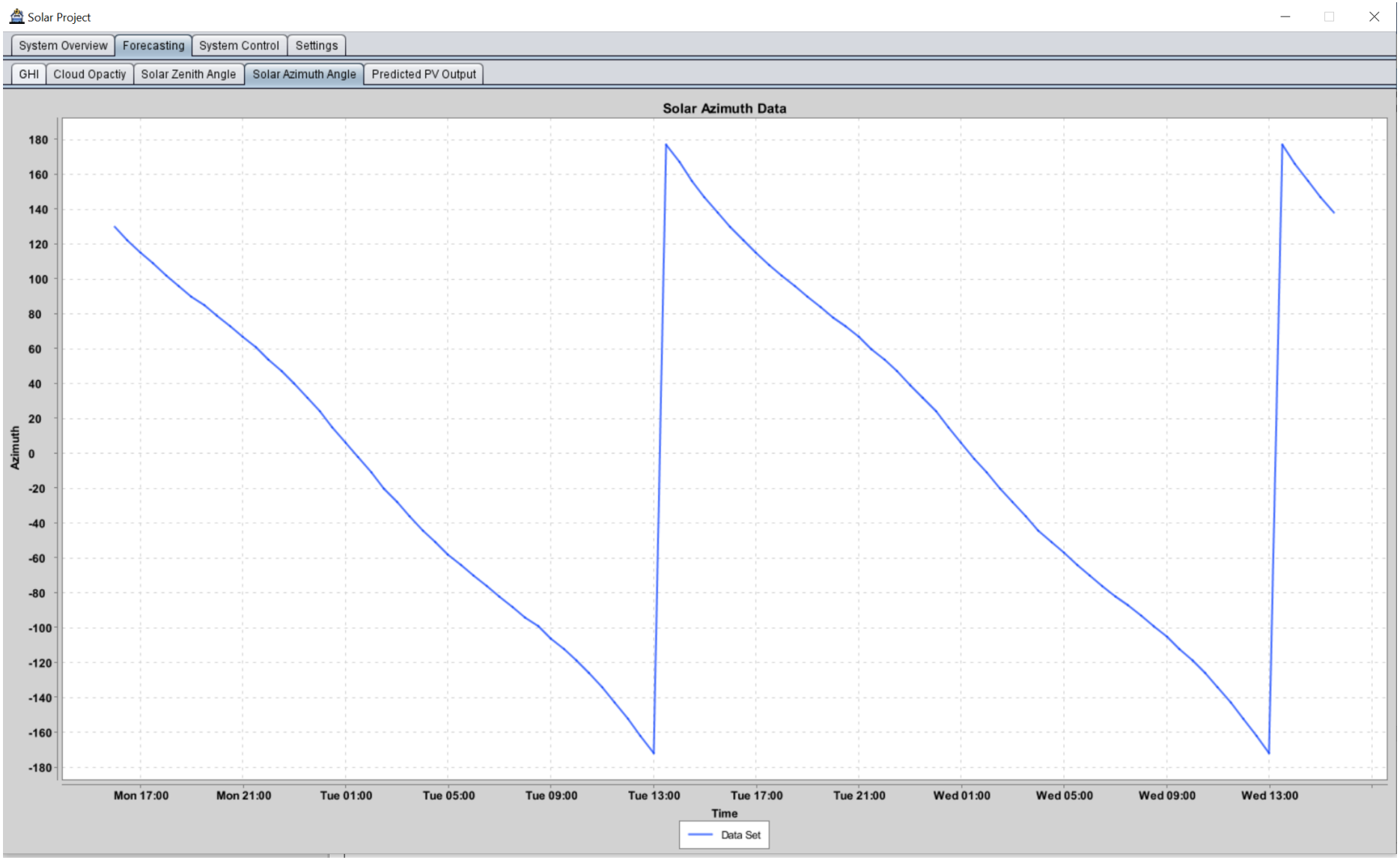
This tab displays a single graph for the forecasted cloud opacity for the next 48 hours. This graph is added when the program is first loaded up with the initForecastGraphs() in the MainFrame class. The method drawLineGraph() from the Graph class is run within initForecastGraphs() to specifically produce this graph and it is returned as a JPanel that can be added to the tab. The data for this graph is from GETCloudOpacity() method in the ForecastAPI class.

*Solar Zenith Angle tab*



This tab displays a single graph for the solar zenith angle for the next 48 hours. This graph is added when the program is first loaded up with the initForecastGraphs() in the MainFrame class. The method drawLineGraph() from the Graph class is run within initForecastGraphs() to specifically produce this graph and it is returned as a JPanel that can be added to the tab. The data for this graph is from GETZenith() method in the ForecastAPI class.

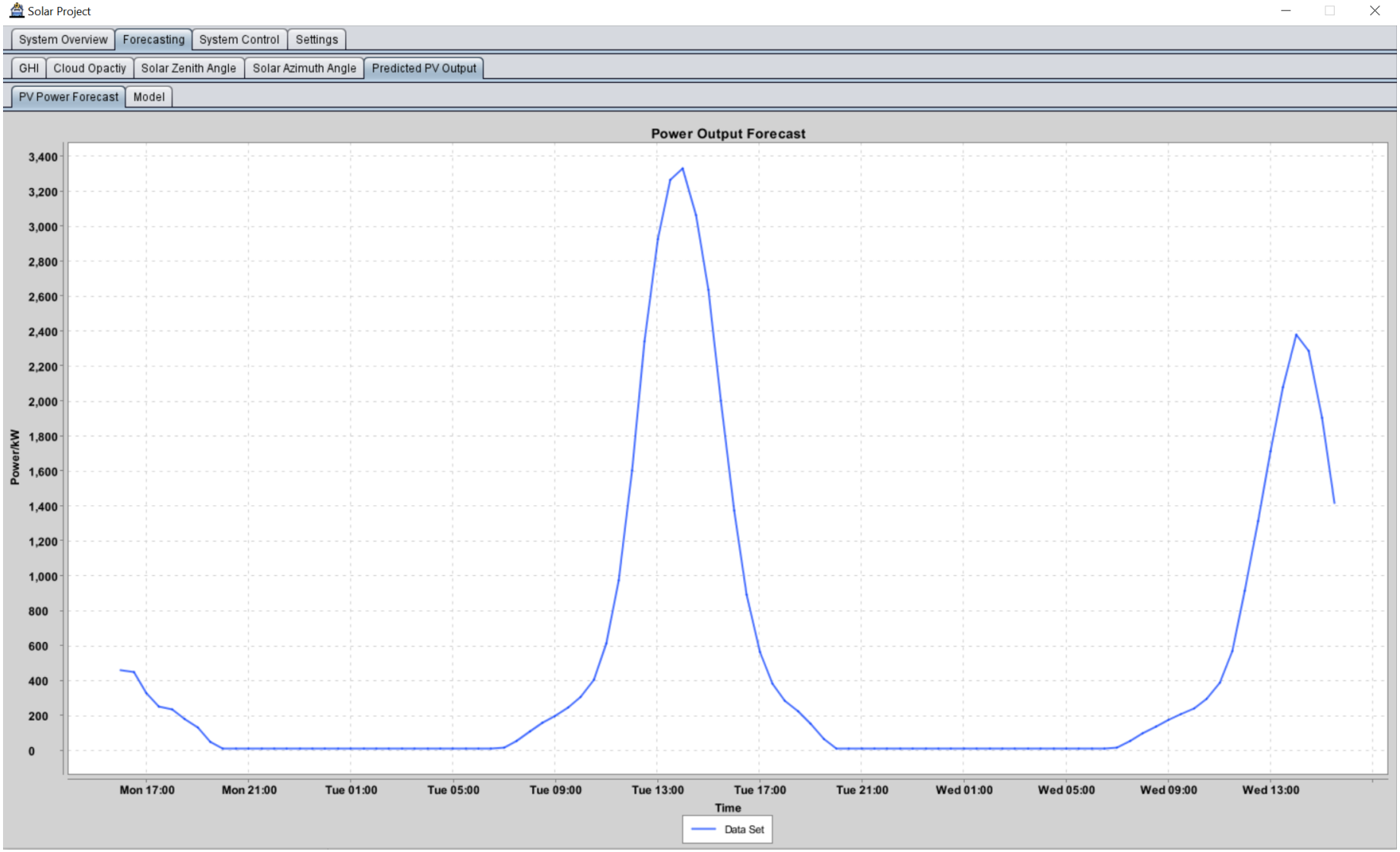
*Solar Azimuth Angle tab*



This tab displays a single graph for the solar azimuth angle for the next 48 hours. This graph is added when the program is first loaded up with the initForecastGraphs() in the MainFrame class. The method drawLineGraph() from the Graph class is run within initForecastGraphs() to specifically produce this graph and it is returned as a JPanel that can be added to the tab. The data for this graph is from GETAzimuth() method in the ForecastAPI class.

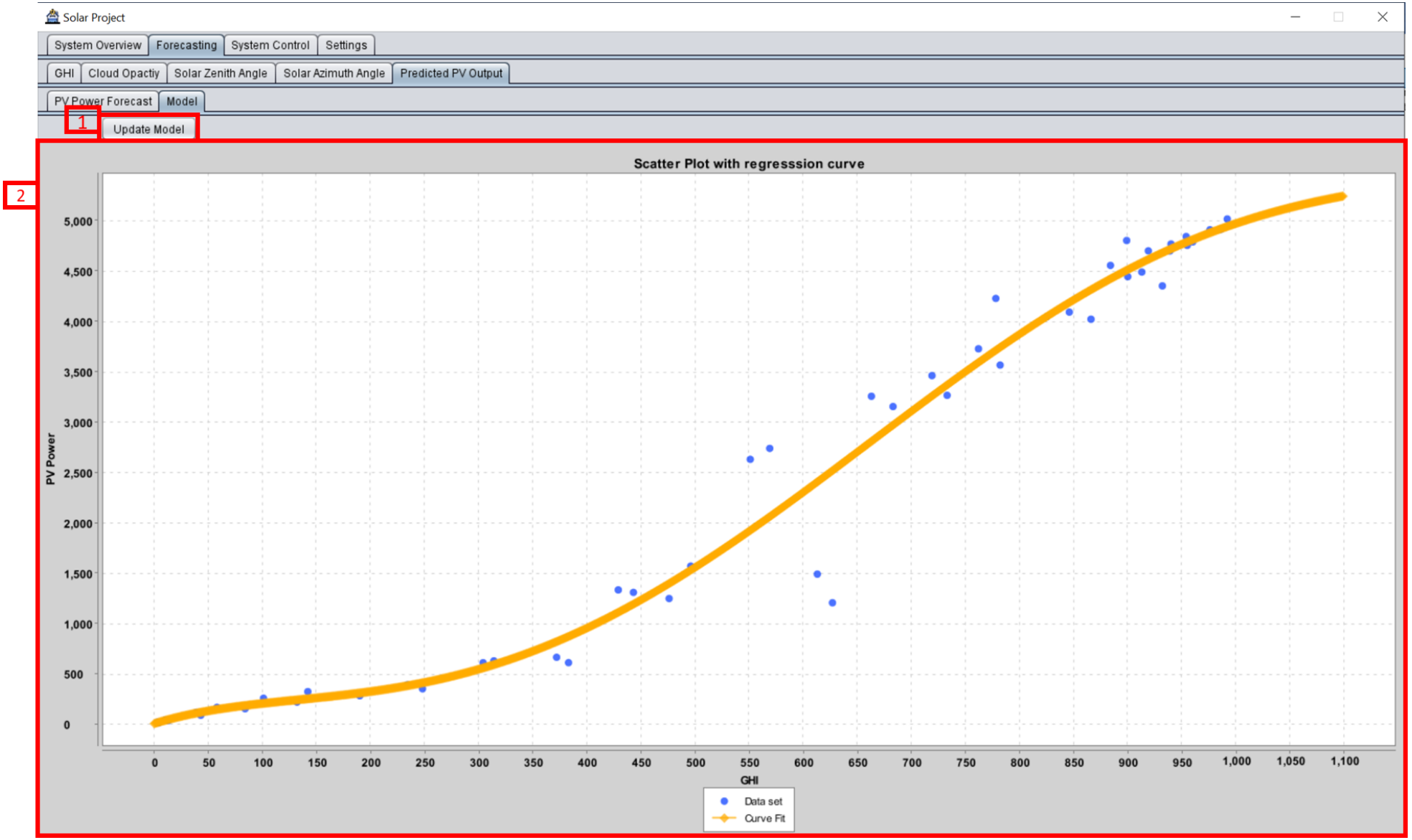
*Predicted PV Output tab*

## PV Power Forecast tab



This tab displays a single graph that shows the predicted solar PV output for the next 48 hours. This data is calculated by the program using the mathematical model and a weather forecast. This graph is added when the program is first loaded up with the initForecastGraphs() in the MainFrame class. The method drawLineGraph() from the Graph class is run within initForecastGraphs() to specifically produce this graph and it is returned as a JPanel that can be added to the tab. The data for this this graph is collected using the GETPPV() method in ForecastAPI. This method uses the Solcast API to get solar radiation data for the next 48 hours. Then it recalculates this data to take into account the tilt, location, and orientation of the panels. Then it uses an object of the PVModel class to run this data through the complex mathematical model is the useModel() method. This returns the data for the predicted solar PV output which is then added to this graph.

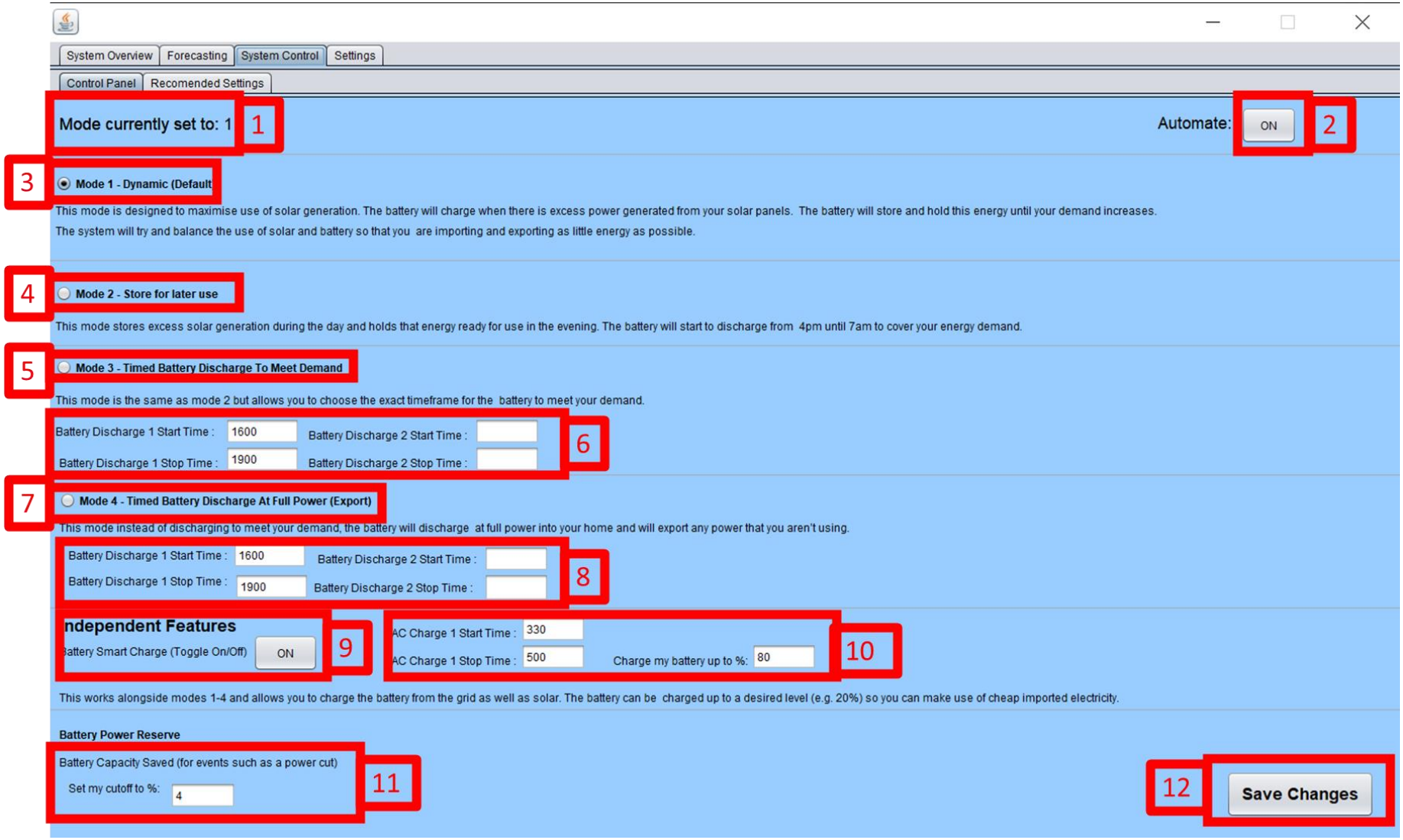
## Model tab


Solar Project
System Overview
Forecasting
System Control
Settings
GHI
Cloud Opactiy
Solar Zenith Angle
Solar Azimuth Angle
Predicted PV Output
PV Power Forecast
Model
1
Update Model
2
Scatter Plot with regresssion curve
PV Power
5,000
4,500
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1,000
1,050
1,100
GHI
Data set
Curve Fit

1) This button can be pressed by the user to update the complex mathematical model. When it is clicked it runs a method called updateModelActionPerformed(). In this method it creates an object of the PVModel class. This object can then run a method called updateModel(). The updateModel() method collects data from the GivEnergy API about the solar PV power output over the last day and also data from the Solcast API about estimated solar radiation over the last day. With this new data along with previous it uses the PolynomialRegression class to re-evaluate the complex mathematical model. Then this scatter graph with update with new values. It will also update the PV power output graph using the new mathematical model.
2) This scatter graph is created and added to the tab when the program is started. The method drawScatterGraph() in the Graph class is used to produce it. The data points for the graph are collected from the pvModelData table in the database with the getGHIData() and getPPVData() methods in the PVModel class. The mathematical function that represents the model is shown by the yellow curve and it is collected from the pvModel table in the database. The useModel() method in PVModel is used to display the curve.

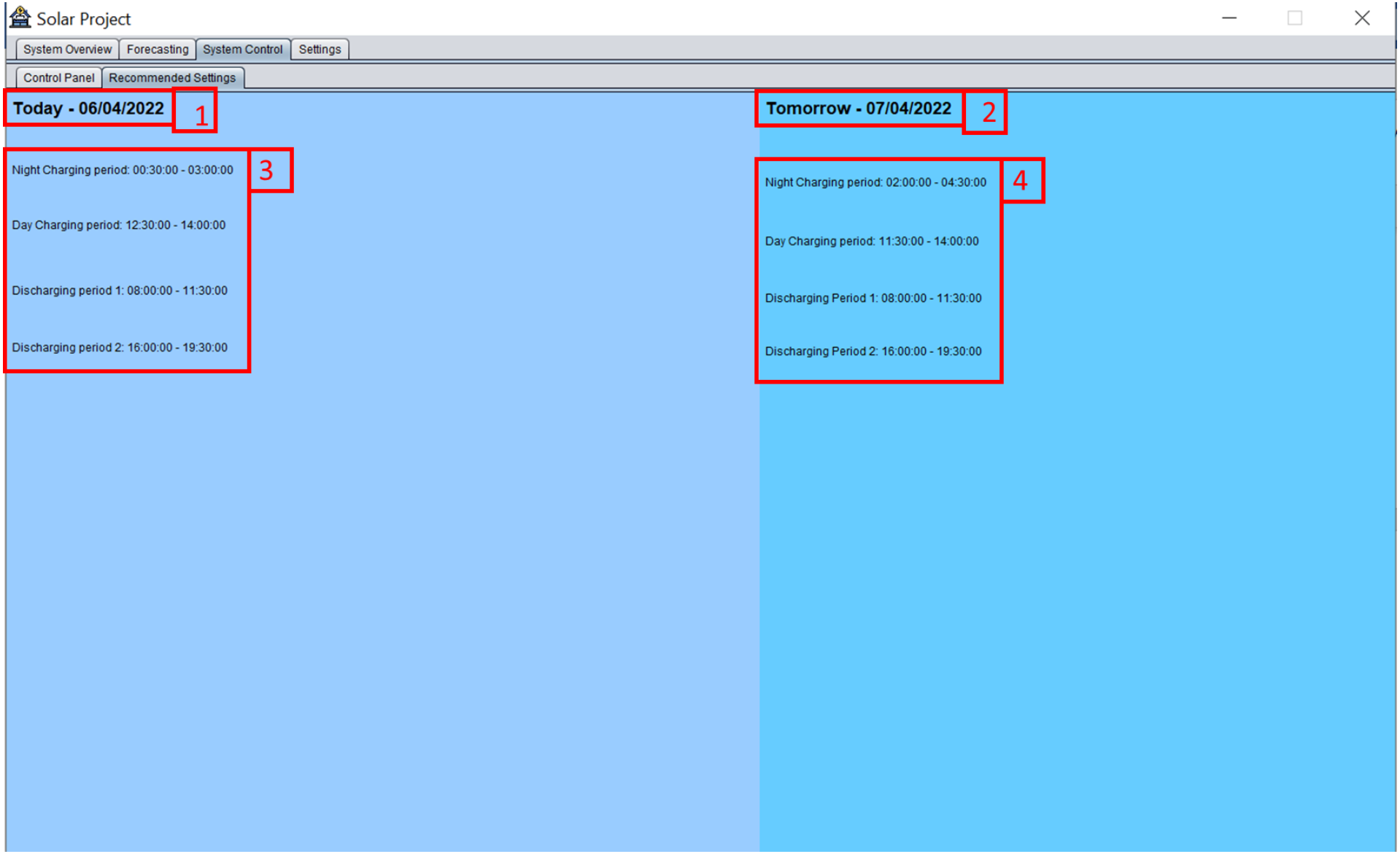
## System Control tab

### *Control Panel tab*



1) Displays the current mode of the system. Gets this data from the GivEnergy API with the GETMode() method in the BatteryGivEnergyAPI class
2) A toggle button that controls whether the program will automatically control and perform mode changes to the system. This button is an extension objective, so its functionality is not fully implemented
3) Radio button that selects mode 1. Selecting this will unselect all other radio buttons.
4) Radio button that selects mode 2. Selecting this will unselect all other radio buttons.
5) Radio button that selects mode 3. Selecting this will unselect all other radio buttons.
6) Textboxes that allow the user to input times of the day when the battery will discharge. The user must enter times in 'HHmm' format.
7) Radio button that selects mode 4. Selecting this will unselect all other radio buttons.
8) Textboxes that allow the user to input times of the day when the battery will discharge
9) A toggle button that controls whether the battery will be in smart charge mode
10) Text fields that allow the user to input when the battery will charge and how far it will charge
11) Text field that sets a cut-off value for the battery SOC
12) Button that will save the changes made to these settings and send API requests so that the changes are implemented
    a. A method called saveChangesActionPerformed() is run. This method checks through all the components of the form. It uses the BatteryGivEnergyAPI class to send mode changes to GivEnergy which will change the real system settings. The method modeChange() in BatteryGivEnergyAPI is used to change the mode to a number which is given in the argument of the method. The discharge() method is used to tell the API between which times to discharge the battery. The charge() method is used to tell the API between which times to charge the battery.
    b. The saveChangesActionPerformed() method also runs a method in the MainFrame class called validateDateFormat(). This method is used to check the user input is the correct time format and if it is not then an error box shows to tell the user it is wrong

*Recommended Settings tab*



1)This label displays the current date

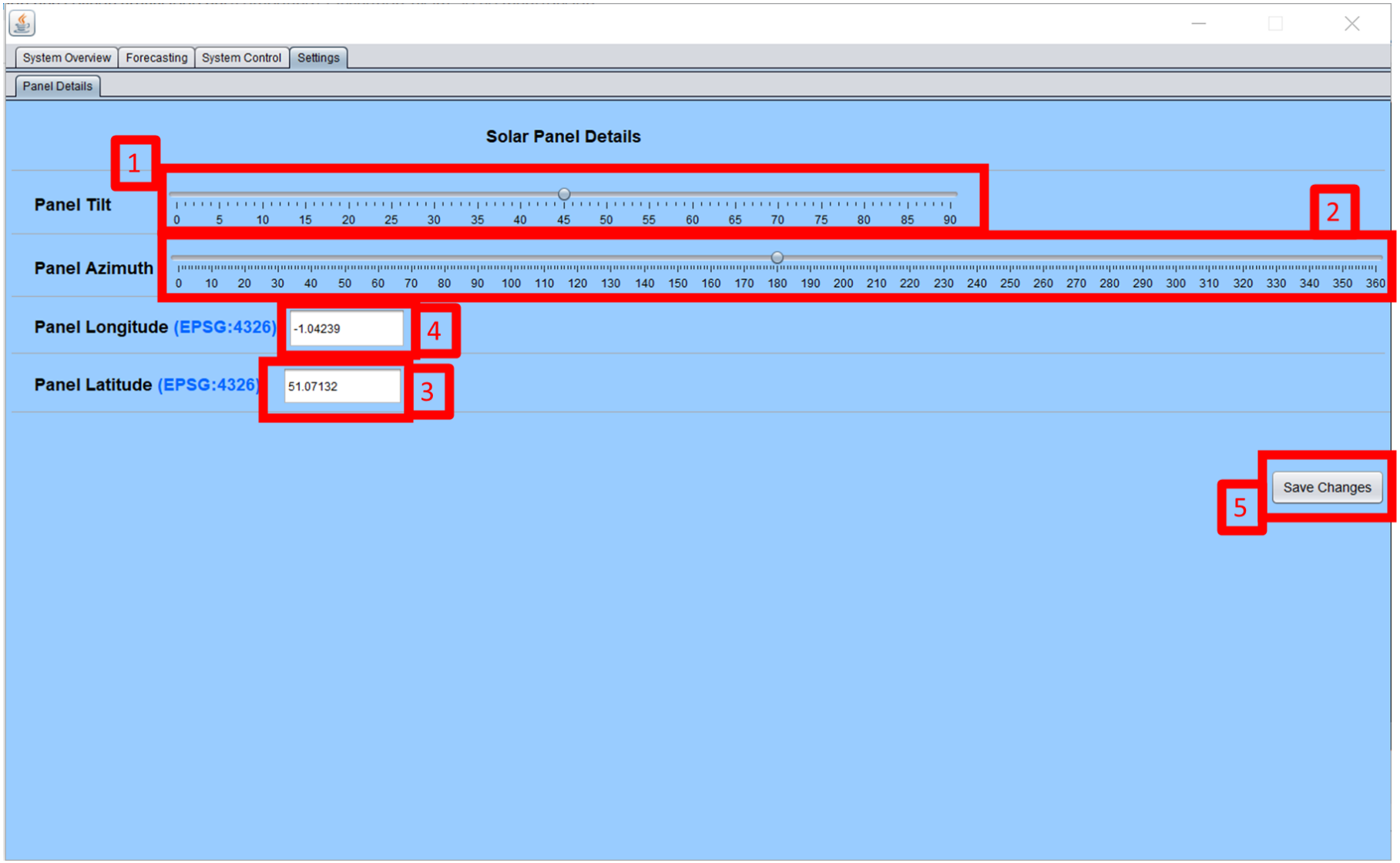
2) This label displays tomorrow's date

3) The time periods in this section are fetched from the SystemSettings database table. The setChargingPeriods() method in the MainFrame class uses the DBpvModel class to run the getChargingPeriods() method to collect these periods from the database.

4) The times periods in this section are calculated in the program with the Energy and Prices classes.

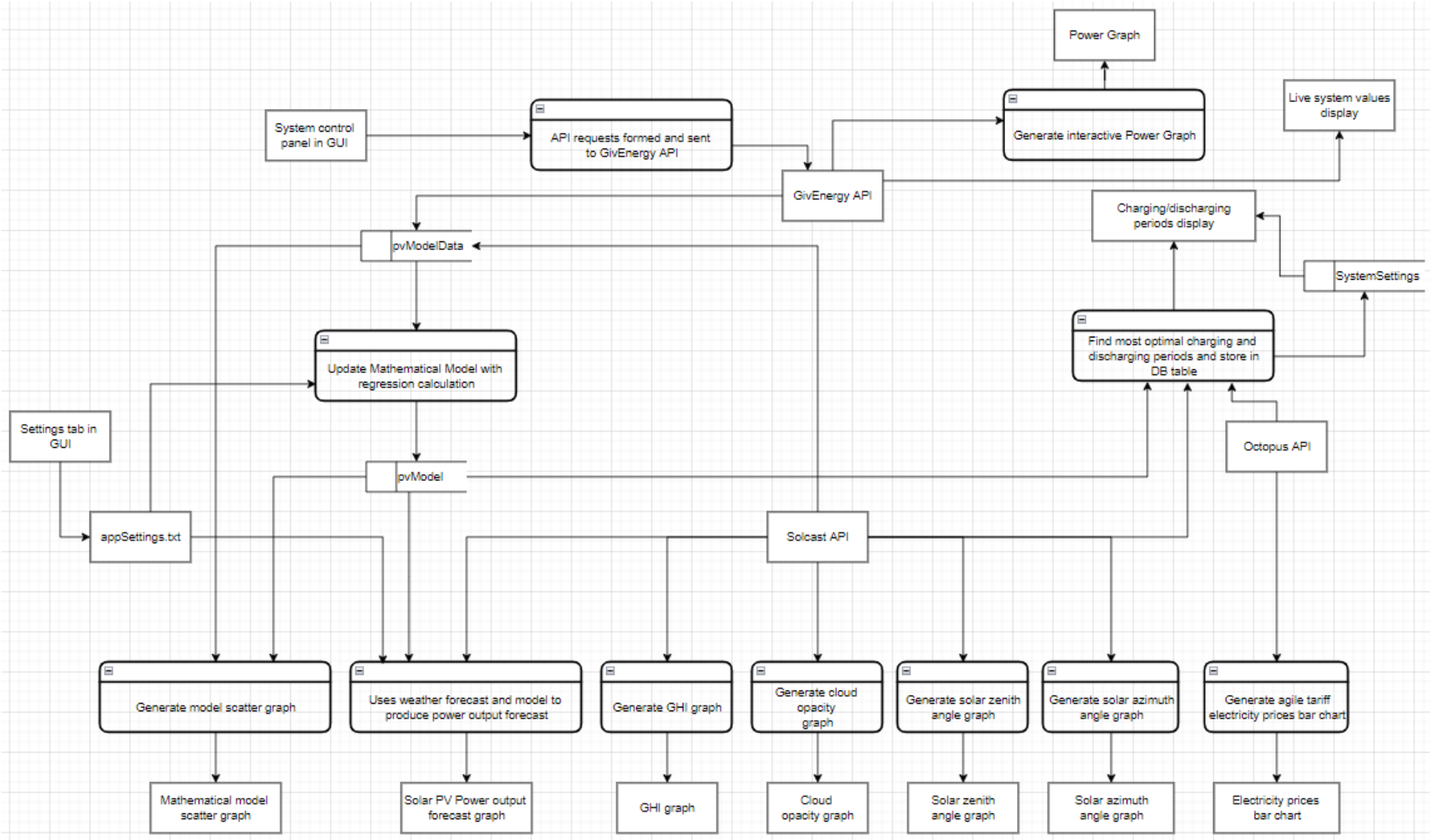
## Settings tab

### *Panel Details tab*

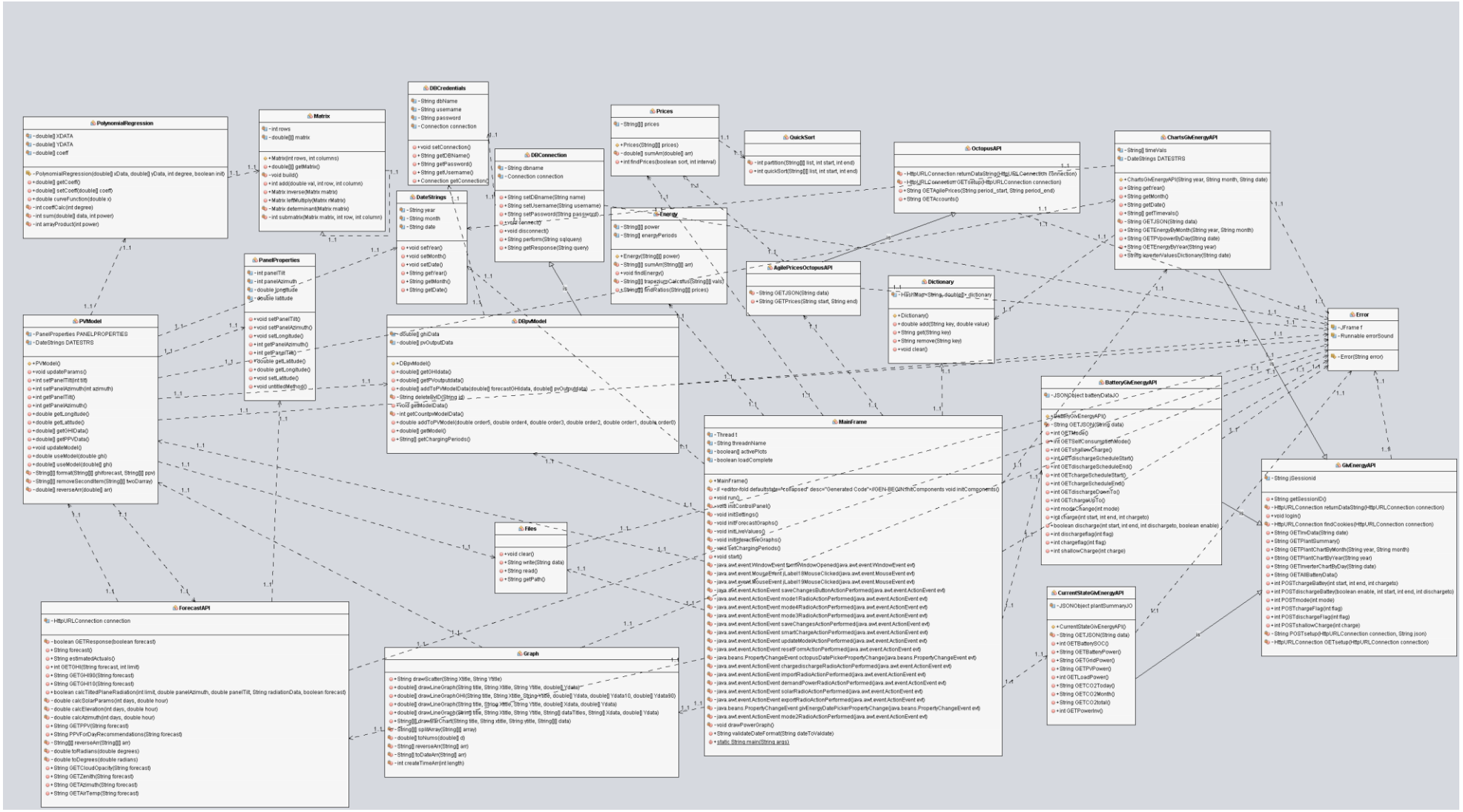


1) Slider that can be used to set the panel tilt values so that the program can make more accurate predictions
2) Slider that can be used to set the panel azimuth values so that the program can make more accurate predictions
3) Text field that allows the user to enter the latitude of the solar panels
4) Text field that allows the user to enter the longitude of the solar panels
5) Button that saves the changes made by storing new values in a file
    a. Pressing the button runs the saveChangesButtonActionPerformed() method. This method uses an object of the Files class and the JSONObject library class to save the data in JSON format in the appSettings.txt file. Then the new values are stored in attributes in the PVModel class using the updateParams() method.

## Data Flow Diagram



# Class Diagram



## Class Summary

| Class | Description |
|---|---|
| MainFrame | This class is the main class that contains the main method. It contains all the attributes needed by the GUI and it initialises the GUI and all the data it needs to display. All the other methods in all other classes are run from this class in some way. |
| Error | Any class can make an object of this class and pass a String into the constructor. When an instance of this class is made an error box shows up to inform the user of an error with a custom message. |
| DateStrings | Holds variables that are string representation of dates such as year, month, and a date. |
| Files | This class has methods that can read and write to text files. |
| Graph | Has methods that generate various graphs and charts to display data in the GUI. The data needed to create these classes is passed into the parameters of the methods. |
| Dictionary | A data structure class that acts as an abstracted HashMap to store keys against their values. |
| OctopusAPI | A parent class that has methods that can collect responses from all the Octopus API endpoints. |
| AgilePricesOctopusAPI | A child class to OctopusAPI that forms a request with specific parameters to collect data from the Octopus Agile endpoint and receives the response from its parent class Octopus API. |
| ForecastAPI | A class that can collect weather forecast data from an API. |
| Prices | This class is used during the optimisation process when the best charging and discharging periods are determined. The agile electricity prices are crucial in this calculation and this class is used to handle this data. |
| Energy | This class is similar to the Prices class, but it performs calculations using the forecasted power output to find how much energy will be produced by times of the day. The most optimum times to charge the battery are found from this as this will determine when the solar panels are the most effective. |
| QuickSort | This class is used to sort a list of values from lowest to highest really quickly using the quicksort algorithm. |
| PolynomialRegression | This class performs least squares regression on a set of bivariate data to determine a polynomial function that summarises the set of data. This is used in the model for prediction of power output. |
| Matrix | This class represents a matrix data type. It can perform operations on matrices, and it is used to perform the least squares regression. |

| | |
|---|---|
| PVModel | Controls the flow of data from databases and APIs and uses other mathematical classes to build the solar forecasting model. It has attributes that can be set in the GUI that can affect how the model is evaluated such as the tilt, azimuth, and location of the solar panels. |
| PanelProperties | Holds variables about the orientation and location of the solar panels. |
| DBpvModel | This class interacts with the tables in the database that hold values to do with the solar forecasting model. It has methods to update and collect data from multiple tables. |
| DBConnection | This class is a parent class to all the other database classes. It provides the actual connection to the database by logging in and establishing a connection with the MySQL database. |
| DBCredentials | This class is used to hold values that are needed for the connection such as the database username and password. This class also holds the Connection object. These values are accessed with getters and setters. |
| GivEnergyAPI | This is a parent class to all the API classes that deal with the GivEnergy API. It contains methods that send the requests to every endpoint needed in the API. The API also requires the client to log in and this class handles that. The API uses cookie retention to remain logged in, so this class also finds that cookie and uses it to make all the requests so that it does not have to log in every time it needs to use the API. |
| BatteryGivEnergyAPI | This is a child class to the GivEnergyAPI and it specifically has methods that can pass in data to the Battery endpoints of the API. It contains specific methods to do with the battery and inverter control and this is the class that is used when the user wants to change the settings of the system. |
| ChartsGivEnergyAPI | This is a child class of GivEnergyAPI and it has methods to collect data from the Charts endpoints of the GivEnergy API. This data is returned from the API in a format that is very easy to pass into a graph and so when the program needs to generate a graph to display some information this class is used to collect that data from the API. |
| CurrentStateGivEnergyAPI | This class is a child class of GivEnergyAPI. It collects live values such as battery state of charge or PV generation that can be displayed on the GUI. The program uses the methods in this class every few minutes so that when the program is running it can display live values from the solar panel system. |

## Method/Property Design

### MainFrame

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Private | Void | initSettings | n/a | Reads from the app settings files and sets attributes to those values. |
| Private | void | initControlPanel | n/a | Uses the GivEnergy APIs to find the current system settings of the solar panel and inverter and loads these settings into the control panel tab of the GUI so that the correct settings can be displayed. |
| Private | Void | drawPowerGraph | n/a | This method uses the graph class to produces a special type of graph where datasets can be added and removed from the axis and multiple dataset can be displayed over each other. |
| Private | Void | octopusDatePickerPropertyChange | PropertyChangeEvent evt | This method is run when the Octopus energy prices date picker component is changed to a new date. It updates the bar chart with data from the day selected. |
| Private | Void | formWindowOpened | WindowEvent evt | This method is called when the program is started and the window is opened. It runs other methods to |

| | | | | |
|---|---|---|---|---|
| | | | | initialise the GUI and it uses multithreading to continually update the live system values tab while the program is running. |
| Private | void | initForecastGraphs | n/a | Uses the Solcast API and Graph class to add graphs of weather forecasts to the GUI. |
| Private | Void | initLiveValues | n/a | Uses the GivEnergyAPI to add live values to the GUI live values tab . |
| Private | Void | initInteractiveGraphs | n/a | Creates the interactive power graph and Octopus Agile tariff bar chart to be displayed in the GUI. |
| Private | Void | setChargingPeriods | n/a | Uses optimisation classes to calculate the optimum charging/discharging periods and then displays those in the GUI. Also fetches previously calculated time periods for the current day from the database. |
| Public | Void | run | n/a | This method overrides a run method in Runnable so that the program can multithread loading the components into the GUI as well as performing calculations and displaying a loading spinner while it is executing. |

| | | | | |
|---|---|---|---|---|
| Public | Boolean | validateDateFormat | String dateToValidate | Tests to see if a date String is in a particular format by parsing that string into the required format and if no error is thrown it returns true as the date is in the correct format. |
| Private | Void | updateModelActionPerformed | ActionEvent evt | When the update model button is pressed this method is called and uses methods in pvModel to update the mathematical model and then replaces old model graphs with new updated ones. |
| Private | Void | saveChangesActionPerformed | ActionEvent evt | When the save changes button is pressed then this method is called and it uses the GivEnergy API to update the system settings from what the user inputted into the form. |

*Attributes*

| Access Modifier | Type | Name | Description |
|---|---|---|---|
| Private | Thread | thread | The thread object that is used when the program needs to multithread operations when loading the program. |
| Private | Boolean | activePlots | This holds Boolean values for whether each data set in the power |

| | | | |
|---|---|---|---|
| | | | graph is displayed or not displayed on the graph. |
| Private | Boolean | loadComplete | Value that stores whether the program has finished loading to prevent processes from running if loadComplete is false. |

## Graph

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | JPanel | drawScatter | String xTitle, String yTitle | Produces a scatter graph from set of data and also adds a curve of best fit using the complex mathematical model. |
| Public | JPanel | drawLineGraph | String title, String xTitle, String yTitle, double[] yData | Produces a line graph plot of a set of data. This method is overloaded to take into account if multiple datasets were represented on the same axis or if the area under the graph was shaded in. |
| Public | JPanel | drawLineGraphGHI | String title, String Xtitle, String Ytitle, double[] Ydata, double[] Ydata10, double[] Ydata90 | Draws a line graph with shaded high and low scenarios from 3 sets of data in the parameters. |
| Public | JPanel | drawBarChart | String title, String xTitle, String yTitle, String[][] data | Produces a bar chart from inputted data. |
| private | Date[] | createTimeArr | Int length | Creates an array of Date objects which represent every half an hour from the |

| | | | | |
|---|---|---|---|---|
| | | | | point in time when the method is run up to a limit in the parameter. |
| Private | Date[] | toDateArr | String[] arr | Converts a String array of dates into a Date array in the HH:mm:ss format. |
| Private | String[] | reverseArr | String[] arr | Reverses the order of an array. |
| Private | String[][] | splitArray | String[][] arr | Splits a 2d array with many arrays that hold 2 values into a 2d array with 2 sub arrays where the first contains all the values that were first in each pair and the second contains all the values that were second in the original pairs. |

## Files

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | Void | clear | n/a | Clears all data from a text file. |
| Public | Void | write | String data | Overwrites all data in file with data in parameter of method. |
| Public | String | read | n/a | Reads and returns all data from text file. |
| String | String | getPath | n/a | Finds the path that the running jar file is stored in so it can find the directory of the text file. |

## Error

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | (constructor) | Error | String error | Creates a JOptionPane that can display to the user that there has been an error. The parameter holds a string with a message to display on the error window. |

*Attributes*

| Access Modifier | Type | Name | Description |
|---|---|---|---|
| Private | JFrame | FRAME | A constant that holds a JFrame object that is used to display the error message |
| Private | Runnable | ERROR_SOUND | A Runnable object that will play a windows error sound when the program has to display the error window. |

## Dictionary

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | Void | add | String key, double value[] | Adds a key and value to a hashmap. |
| Public | Double[] | get | String key | Returns data from a key from the hashmap. |
| Public | Void | remove | String key | Removes data using a key. |
| Public | Void | clear | n/a | Clears the hashmap. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | HashMap | DICTIONARY | This object of HashMap holds data and is abstracted by the Dictionary class. |

## DateStrings

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | String | year | Holds a string representation of a month. |
| Private | String | month | Holds a string representation of a month. |
| Private | String | date | Holds a string representation of a date. |

## OctopusAPI

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | String | GETAgilePrices | String periodStart, String periodEnd | Uses the Octopus API to get the agile prices for a given time period and returns a string representation of a JSON response. |
| Private | String | returnDataString | HttpURLConnection connection | Establishes a connection with the endpoint and forms the response using an input stream. This data is then returned and it will be of JSON format and returned as a String. |
| Private | HttpURLConnection object | GETsetup | HttpURLConnection connection | Sets up a GET API request and returns the HttpURLConnection object. |

## AgilePricesOctopusAPI

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | String[][] | GETPrices | String start, String end | Uses the Octopus API methods from the OctopusAPI class it inherits from to get the agile prices for an inputted day. |

## ForecastAPI

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Private | String | GETResponse | boolean forecast | Uses the Solcast API to get a weather forecast for the next 48 hours. Returns data as a string representation of a JSON response. |
| Private | String[][] | calcTiltedPlaneRadiation | int limit, double panelAzimuth, double panelTilt, String radiationData, boolean forecast | Performs calculations on the solar radiation GHI data from the forecast to calculate the actual intensity of light that is incident to the solar panels (the API only returns the solar radiation intensity for horizontal ground). It used values of panel tilt angle and azimuth and location coordinates to perform these calculations. |
| Public | Double[] | GETPPV | String forecast | Uses the forecast response and mathematical model to return an array of predicted power output from the solar panels. |
| Public | String[] | PPVForDayRecommendations | String forecast | Uses the mathematical model to return an array of predicted power output and time. |
| Private | Double[] | calcSolarParams | int days, double hour | Calculates the current hour angle and declination based |

| | | | | |
|---|---|---|---|---|
| | | | | upon the current date and time and location. |
| Private | Double | calcElevation | int days, double hour | Uses the calcSolarParams method and trigonometry to calculate the elevation of the sun above the horizon at a location and time. |
| Private | Double | calcAzimuth | int days, double hour | Uses the calcSolarParams method and trigonometry to calculate the azimuth of the sun at a location and time. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | HttpURLConnection | connection | Holds the HttpURLConnection that is used when using the API. |

## Prices

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | String[][] | findPrices | boolean sort, int interval | Finds the total cost of importing energy for every 2.5 hour period over a day. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | String[][] | PRICES | Holds the price and time for each half hour across a day. |

## Energy

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | Void | findEnergy | n/a | Calculates a forecast for the energy that is produced by the solar panels based upon the expected power output of the panels. |
| Private | Double | trapeziumCalculus | String[][] vals | Uses the Trapezium Rule to perform integration on a set of data. |
| Public | String[] | findRatios | String[][] prices | Finds the ratios between electricity price and energy generated at time periods over a day. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | String[][] | POWER | Holds power values and a time for that value in a 2d array. |
| private | String[] | energyPeriods | String that holds a value of energy produced for each time period. |

## QuickSort

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | Void | quickSort | String[][] list, int start, int end | A recursive method that can order an array of items very fast. |

| | | | | |
|---|---|---|---|---|
| Private | Int | partition | String[][] list, int start, int end | Splits array around a value then puts that value in sorted array and put all smaller elements before that value and all greater elements after that value. |

## PolynomialRegression

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | Double | curveFunction | double x | Returns a value based on an input value using the polynomial function stored in that object. |
| Private | Void | coeffCalc | Int degree | Uses matrices and arrays of x and y data to perform least squares regression to produces a polynomial function that represents a best fit for a set of data. |
| Private | Double[] | arrayProduct | Int power | Combines the x data and y data by multiplying each y term by the corresponding x term raised to the power of their index. |
| Private | Double | sum | Double[] data, int power | Sums data in an array where each data item is raised to the power in the parameter. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | Double[] | X_DATA | Holds double values for the x data in a data set. |

| | | | |
|---|---|---|---|
| Private | Double[] | Y_DATA | Holds double values for the y data in a data set. |
| Private | Double[] | coeff | Holds a set of values that are the coefficients to the polynomial function. |

## PVModel

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | Void | updateParams | n/a | After the user changes settings in the GUI such as the location coordinates of the solar panels or tilt of the panels, this method can update the attributes in the class that are holding information. |
| Public | Double[] | getGHIData | n/a | Returns all the solar radiation data that is stored in the database that is being used by the complex mathematical model. |
| Public | Double[] | getPPVData | n/a | Returns all the Solar Panel power output data that is stored in the database that is being used by the complex mathematical model. |
| Public | Void | updateModel | n/a | A key function that will update and refine the complex mathematical model |

| | | | | |
|---|---|---|---|---|
| | | | | with new data that has been collected over the last day. |
| Public | Double | useModel | Double ghi | From an inputted value of GHI (solar radiation) in the parameter it will return a value for the expected power output of the solar panels based on the mathematical model. |
| Public | Double[] | useModel (overloaded) | Double[] ghi | From an inputted array of values of GHI (solar radiation) in the parameter it will return an array of values for the expected power output of the solar panels based on the mathematical model. |
| Private | Double[] | Format | String[][] ghiEstimatedValues, String[][] ppv | Formats data that is collected from temporary storage in database and data that is collected from weather forecast API. It calculated average power outputs over half hour periods of the forecasted time frame according to the times returned from the API. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| Private | PanelProperties | PANEL_PROPERTIES | An object that holds variables about the orientation and location of the solar panels. |
| Private | DateStrings | DATE_STRS | An object that holds variables that represent a year, month and date. |

## PanelProperties

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | String | panelAzimuth | Holds an integer value for panel azimuth. |
| Private | String | panelTilt | Holds an integer value for panel tilt. |
| Private | String | latitude | Holds a double value for a latitude. |
| Private | Double | longitude | Holds a double value for a longitude. |

## Matrix

(More detail on some of these methods can be found in the Algorithm Design section)

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Private | Void | build | n/a | Creates an n*m dimensional array that is used to fundamentally hold the data stored in a matrix object. |
| Public | Void | add | double val, int row, int column | Adds a value to a specific position in the matrix. |

| | | | | |
|---|---|---|---|---|
| Public | Matrix | inverse | Matrix matrix | Performs an inversing operation on the matrix that is passed into the method. The method then returns the inversed version of that matrix. |
| Public | Matrix | leftMultiply | Matrix rMatrix | Performs matrix multiplication and returns the result of the operation as a matrix. |
| Private | Double | determinant | Matrix matrix | Calculates the determinant of a given inputted matrix |
| Private | Matrix | submatrix | Matrix matrix, int row, int column | Finds the particular submatrix for a given row and column of a matrix. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | Double[][] | matrix | Holds values stored in the matrix object. |
| Private | Int | ROWS | Holds the number of rows in the matrix. |
| Private | Int | COLUMNS | Holds the number of columns in the matrix. |

## GivEnergyAPI

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| public | Void | login | n/a | Method that is called when the program needs to connect to the GivEnergy API. It also sets a value to the jsessionid attribute so the same connection can be retained. |
| private | String | returnDataString | HttpURLConnection connection | Uses an HttpURLConnection object to get the API response from any GivEnergy API endpoint and returns it as a string. |
| private | HttpURLConnection | POSTSetup | HttpURLConnection connection, String json | Method that sets up a connection that can send data in JSON format to a POST endpoint so that the program is able to send data to the GivEnergy system. |
| private | HttpURLConnection | GETSetup | HttpURLConnection connection | Method that sets up the connection to a GET request. |
| Private | String | findCookies | HttpURLConnection connection | Gets the cookie string from the API connection so that the program can remain logged into the API with cookie retention. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| private | String | jSessionID | String representation of the session cookie used by the GivEnergyAPI used to retain the connection to the API. |

## CurrentStateGivEnergyAPI

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | (constructor) | CurrentStateGivEnergyAPI | n/a | Uses its super class to get an API response for the current live values of the system. Also ensures that the program is logged into the API and if not uses the login() method to do so. |
| Private | JSONObject | GETJSON | String data | Converts a String of data into a JSONObject of data in JSON format that can be more easily accessed. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| Private | JSONObject | PLANT_SUMMARY_JO | Holds a JSONObject object of an API response that is used in the rest of the class. |

## ChartsGivEnergyAPI

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | Double[][] | GETEnergyByMonth | String year, String month | Uses a GivEnergyAPI response to return a set of data representing different energy uses such as generation and demand over |

| | | | | |
|---|---|---|---|---|
| | | | | the course of a month during that year. |
| Public | String[][] | GETPVPowerByDay | String date | Uses a GivEnergy API response to return a set of data about the power generation by the solar panels for a specific date. |
| Public | Double[][] | GETEnergyByYear | String year | Uses a GivEnergyAPI response to return sets of data representing various energy uses such as generation and demand over the course of each month over the given year. |
| public | Dictionary | inverterValuesDictionary | String date | Uses the Dictionary class to create a new Dictionary and store array objects against a key. This is an effective way to return a single object that holds data sets used in graphs that show multiple sets of data on the same axis. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| private | DateStrings | DATE_STRS | An object that holds variables of dates such as year, month and a date. |
| private | String[] | timeVals | An array that holds String representations of times that are used when generating graphs of power or energy against time. |

## BatteryGivEnergyAPI

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | (constructor) | BatteryGivEnergyAPI | n/a | Uses its super class to get a API response JSON. This is stored in the batteryDataJO attribute of this class. |
| Private | JSONObject | GETJSON | String data | Converts a String of data into a JSONObject of data in JSON format that can be more easily accessed. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| private | JSONObject | BATTERY_DATA_JO | Holds a JSON data response that contains all the battery data from the system after an API response has been received. |

## DBConnection

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| public | void | connect | n/a | Makes the connection to the database so that data can be stored or collected. |
| Public | Void | Disconnect | n/a | Closes the connection to the database. |
| public | Void | perform | String sqlQuery | Prepares and performs a MySQL query. |
| public | ResultSet | getResponse | String query | Method that can execute a query specifically when collecting data and returns it as a ResultSet. |

*Attributes*

| Access Modifiers | Type | Name | Description |
|---|---|---|---|
| private | DBCredentials | dbCreds | DBCredentials object that holds all the values needed for the connection to the remote MySQL database. |

## DBCredentials

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| Public | Connection | getConnection | n/a | A getter for the Connection object. |
| Public | String | getDBName | n/a | A getter for the database name. |

| | | | | |
|---|---|---|---|---|
| Public | String | getUsername | n/a | A getter for the database username. |
| Public | String | getPassword | n/a | |
| Public | Void | setConnection | Connection connection | A setter for the Connection object. |

*Attributes*

| Access Modifier | Type | Name | Description |
|---|---|---|---|
| Private | String | dbName | String object that holds the DB name. |
| Private | String | username | String object that holds the DB username. |
| Private | String | password | String object that holds the DB password. |
| Private | Connection | connection | Connection object that holds the connection to the remote database. |

## DBpvModel

*Methods*

| Access Modifiers | Returns | Name | Parameters | Description |
|---|---|---|---|---|
| public | void | addToPVModelData | double[] forecastGHIdata, double[] pvOutputdata | Inserts new data that can be used in the model into a database table. |
| private | void | deleteByID | String id | Can delete a set of data points that have a specific primary key date. |

| | | | | |
|---|---|---|---|---|
| public | void | getModelData | n/a | Gets all the model data points from the database and stores them into arrays that can be used elsewhere in the program. |
| private | int | getCountpvModelData | n/a | Performs SQL statement to find how many records are in the model data table. |
| public | void | addToPVModel | double order5, double order4, double order3, double order2, double order1, double order0 | Adds calculated model parameters to a table in the database. |
| public | double[] | getModel | n/a | Looks in the model table to get the model parameters that can be used to make forecast predictions. |
| public | String[] | getChargingPeriods | n/a | Returns a list of previously calculated charging and discharging periods to display in the GUI that are stored overnight in a table. |

*Attributes*

| Access Modifier | Type | Name | Description |
|---|---|---|---|
| Private | Double[] | ghiData | Array that contains all the forecasted GHI data stored in the database. |
| private | Double[] | pvOutputData | Array that contains all the power output data from the solar panels that is stored in the database. |

## Algorithm Design

### Curve Fitting Algorithm

```
def coeffCalc(degree): //Uses matrices and arrays of x and y data to perform least squares regression to produces a polynomial function that represents a best fit for a set of dat
        m = degree + 1
        coeff = []
        Matrix Xmatrix = new Matrix(m,m)
        for i in range m:
                for j in range m:
                        Xmatrix.add(sum(xdata, i+j), i, j) //sum of Xs matrix
        Matrix XYmatrix = new Matrix(m,1)
        XYdata = []
        for i in range m:
                XYdata = arrayProduct(i)
                XYmatrix.add(sum(XYdata, 1), i, 0)//sum of XYs matrix
        Matrix inverse = Xmatrix.inverse(Xmatrix) //inverses matrix
        Matrix coeffMatrix = inverse.leftMultiply(XYmatrix) //multiplys xMatrix and xyMatrix to get vector matrix with coefficient values in
        for i in range m:
                coeff[i] = coeffMatrix.getMatrix()[i][0]
```

This sub routine calls many methods and uses other classes to calculate the coefficients of a polynomial function to represent a best fit for a set of data. This function is used when predicting the power output of the panels from a given solar radiation forecast. This method effectively builds two matrices and then performs a matrix multiplication on them to produce a final column matrix that contains the values of the coefficients. As per the pseudocode the first Matrix to be generated is called Xmatrix. This matrix has m*m dimensions and contains sums of increasing powers of data. For example, on element could contain a sum of all the data where each item in the data set has been squared, and another element in the matrix could be the same but the data is all cubed then summed. Here is an image to illustrate this matrix:

$$\begin{bmatrix} n & \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 & \cdots & \sum_{i=0}^{n} x_i^m \\ \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 & \sum_{i=0}^{n} x_i^3 & \cdots & \sum_{i=0}^{n} x_i^{(m+1)} \\ \sum_{i=0}^{n} x_i^2 & \sum_{i=0}^{n} x_i^3 & \sum_{i=0}^{n} x_i^4 & \cdots & \sum_{i=0}^{n} x_i^{(m+2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{n} x_i^m & \sum_{i=0}^{n} x_i^{(m+1)} & \sum_{i=0}^{n} x_i^{(m+2)} & \cdots & \sum_{i=0}^{n} x_i^{2m} \end{bmatrix}$$

After the algorithm has generated this matrix using all the x data, which in my projects case is values of GHI, the algorithm generates another matrix of m*1 dimensions. This matrix also contains sums of powers of x data but each term in each sum is multiplied by a corresponding value of y data for that x value. Y data in the context of my project is the power output of the solar panel system. Here is an image to illustrate that matrix:

$$\begin{bmatrix} \sum_{i=0}^{n} y_i \\ \sum_{i=0}^{n} x_i y_i \\ \sum_{i=0}^{n} x_i^2 y_i \\ \vdots \\ \sum_{i=0}^{n} x_i^m y_i \end{bmatrix}$$

The first matrix is then inversed and stored in the Matrix 'inverse' as per the pseudocode. This inverse is then left multiplied to the second matrix and the resulting matrix is stored in coeffMatrix. The values stored in coeffMatrix are the coefficients to the polynomial approximation and they are then added to a coefficients array and then the algorithm is complete.

## Algorithm for most Optimal Charging Periods

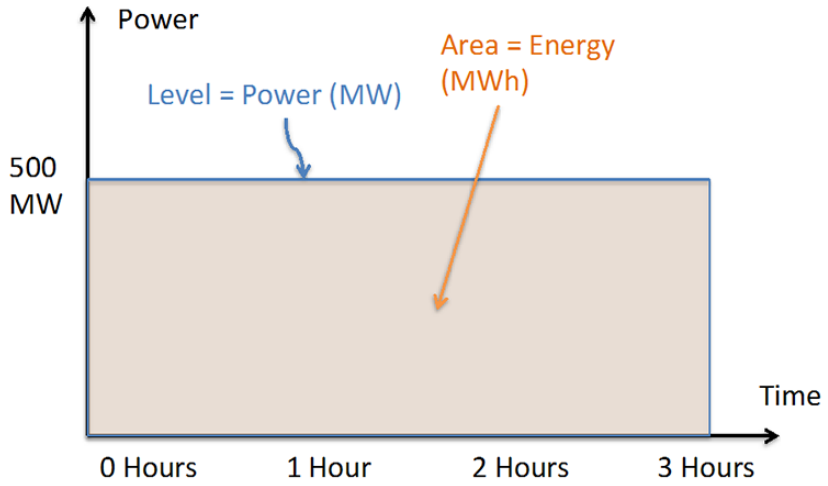
```
def findRatios(prices): //finds the ratios between price and energy generated over the day
        ratios -> [][]
        for i in range (prices.length):
                try:
                        ratios[i][0]-> (prices[i][0])/(energy[i]) //ratio of price and energy
                catch:
                        ratios[i][0] -> infinity //incase the energy produced is equal to 0 then ratio set to very high value
                ratios[1](1) -> prices[i](1)
                ratios[1][2] -> prices[i][2]
        return ratios

def findEnergy(): //Calculates a forecast for the energy that is produced by the solar panels based upon the expected power output of the panels.
        energyPeriods -> []
        subarr -> []
        energy -> 0.0
        for i in range (power.length-5):
                for j in range 5:
                        subarr[j] -> power[i+j]
                energy -> trapeziumCalculus(subarr) //calcluates energy from power values
                energyPeriods[1] => energy

def trapeziumcalculus (values): //uses the Trapezium rule to perform numerical integration on a power-time data set to find energy produced
        energy -> values[0][0] + values[values.length-1][0]
        energy -> 0.25 * (energy + (2 * SumArr(values)))
        return energy
```

This algorithm is used to determine which period of the day is the most optimal to charge the battery based on the energy that will be produced during each time period and the price of importing energy in that period. It does this by calculating the ratio of price:energy for each 2.5 hour period of the day (where price is the total price of importing only energy from the grid in that period and energy is the predicted energy that the solar panels will generate over that period). The forecasting model I have developed only predicts power output at each time of the day and for this algorithm I need the energy produced over prolonged periods. In order to find the values of energy I am performing a type of numerical integration called the Trapezium Rule on the

power-time dataset which will find the value of energy between 2 time periods. This is equivalent to finding the area under a power-time graph which is equal to the energy as shown by this graph (https://www.e-education.psu.edu/ebf483/node/644 ):




After the energy produced is calculated for every 2.5 hour period of the day (only 'daytime' hours), the algorithm uses the findRatios() method to find the ratio of price to energy for every period. The reason for this is that the time period that is the best will have the smallest value for its ratio as the price will be small and the energy produced will be large. The period with the smallest ratio can then be found from the ratios array and then the program will recommend that time period to the user.

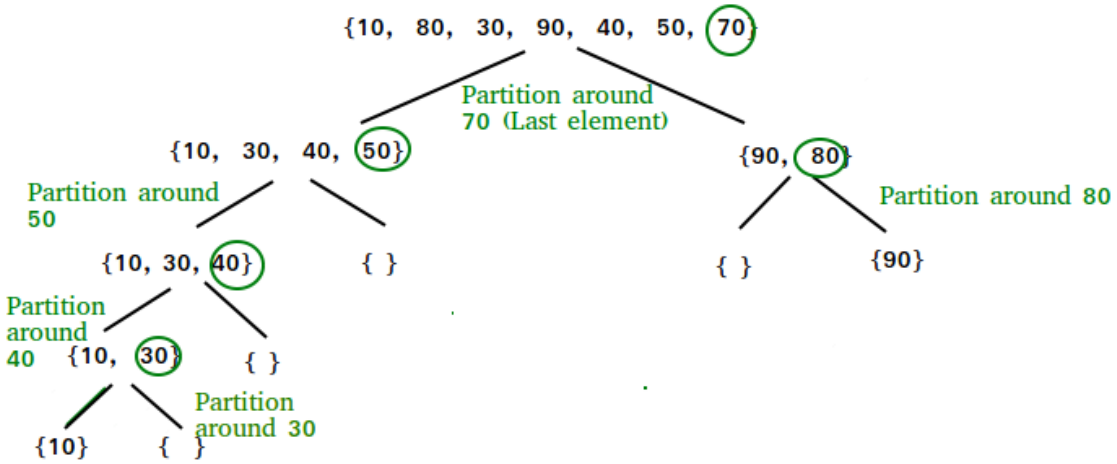
## Quicksort Algorithm

```
def partition(list, start, end):
        //splits array around a value then puts that value in sorted array and put all smaller elements before that value and all greater elements after that value
        pivot = list[end][0]
        i = start - 1
        for j in range(start, end-1):
                if(list[j][0]<pivot):
                i = i + 1
                temp = list[i]
        list[i] = list[j]
        list[j] = temp
        temp = list[i+1]
        list[i+1] = list[end]
        list[end] = temp
        return (i+1)

def quicksort(list, start, end): //A recursive method that can order an array of items very fast
        if(start<end):
                part = partition(list, start, end)
                quicksort(list, start, part-1)
                quicksort(list, part+1, end)
```

This algorithm was sourced from: https://www.geeksforgeeks.org/quick-sort/

The QuickSort works by picking an element of a list as a 'pivot' value. This value is placed in the sorted list at the correct position and then all elements that are smaller than that are put before that pivot value and all elements that are greater than it are put after it. The image below from https://www.geeksforgeeks.org/quick-sort/ illustrates the process if the pivot was chosen as the last element in each list:




This algorithm is used to sort an array of numbers from lowest to highest. This is used in my program because during the optimisation process where the program is calculating the best periods on which to charge and discharge the battery an array of ratios between the energy produced and total price over each period is calculated and the period with the smallest ratio is most beneficial. In order to find the smallest ratio value in the array the program uses the quicksort to sort the array and then the smallest item will be at index 0 in the array.

## Inversing Matrix Algorithm

```
def inverse(Matrix matrix): //Performs an inversing operation on the matrix that is passed into the method. The method then returns the inversed version of that matrix.
        rows -> getRows()
        columns ->getColumns()
        Matrix inverse -> new Matrix(rows, columns) //new matrix object created to hold inverse matrix
        for i in range matrix.getRows():
                for j in range matrix.matrix[i].length:
                        //uses determinant and submatrix method to calculate value of element in inverse array and then depending on position makes it negative
                        inverse.matrix[i][j] -> (-1**(i+j)) * determinant(submatrix(matrix,i ,j))
        multiplier -> 1/determinant(matrix)
        for i in range inverse.getRows():
                for j in range i:
                        temp -> inverse.matrix[i][j]
                        inverse.matrix[i][j] -> inverse.matrix[j][i] * multiplier //mulitplies every element by multiplier
                        inverse.matrix[j][i] -> temp * multiplier
        return inverse
```

This algorithm determines and returns the inverse of a matrix. The purpose for inversing a matrix is so that it can be multiplied onto the other side of a matrix equation. There are many steps to invert a matrix and there is a recursive nature to it as well. To perform an inverse the matrix of minors must be found for all elements in the matrix. Then the determinant of each minor is multiplied by the original element and every other value is multiplied by -1. These values are all added to the new 'inverse' matrix. This is all represented by this line in the algorithm:

```
inverse.matrix[i][j] -> (-1**(i+j)) * determinant(submatrix(matrix,i ,j))
```

Then the determinant of the original matrix is found and the reciprocal of this is stored in 'multiplier.' Every value in the inverse matrix is then multiplied by 'multiplier' in the next nested for loop. The inverse matrix is then returned from the method and the inverse has been found.

## Finding Determinant of Matrix

```
def determinant(Matrix matrix): //Calculates the determinant of a given inputted matrix
        if(matrix.matrix.length != matrix.matrix[0].length):
                return -1
        if(matrix.getRows() == 2):
                return (matrix.matrix[0][0] * matrix.matrix[1][1]) - (matrix.matrix[0][1] * matrix.matrix[1][0]) //special case if matrix is only 2x2
        det -> 0
        for i in range matrix.getRows():
                det -> det + (-1**i) * matrix.matrix[0][i] * determinant(submatrix(matrix, 0, i)) //recursion as determinant of each submatrix needs to be found, every other value multiplied by -1
        return det
```

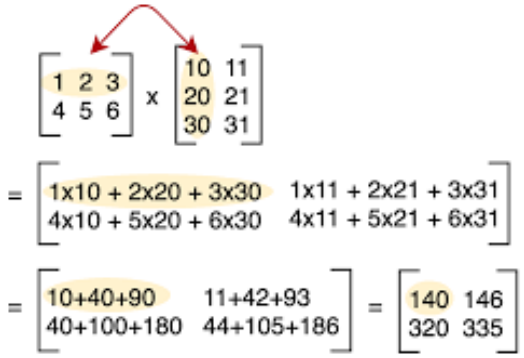
The determinant of a matrix is a calculated value from that matrix. It is used when calculating the inverse of a matrix. The determinant is effectively the sum of every element along the top row where each element is multiplied by the determinant of its submatrix minor. This is what gives this algorithm its recursive nature as in order to find the determinant you need to find determinants of smaller submatrices of that matrix. The first if statement is used to return the method early if the matrix is a non-square matrix. The second matrix is a special case where the matrix could be 2x2 which is very easy to calculate in one line of code. The loop after that is used to run through each element and then for each element recursively find determinants of submatrices to find the determinant.

## Multiplying Matrices

```
def leftMultiply(Matrix Rmatrix): //Performs matrix multiplication and returns the result of the operation as a matrix.
        Matrix product -> new Matrix(Rmatrix.rows, Rmatrix.columns)
        for row in range Rmatrix.rows: //iterates over every row in matrix
                sum -> 0
                for column in range columns: //nested loop to cover each value in the row
                        sum -> sum + Rmatrix.matrix[column][0]*matrix[row][column]
                product.add(sum, row, 0)
        return product
```

This method calculates each element value of each row and then adds it to the new product matrix in the line product.add(sum, row, 0). There is a nested for loop in order to cover every element in all the rows and columns.

In order to multiply two matrices each row of the left hand matrix is flipped vertical and then each element of that row is multiplied to a corresponding element in a column in the right hand matrix. This is repeated for all values in the right hand matrix until a whole new matrix has been formed which will be the product of the two matrices. To illustrate this I added a diagram (https://towardsdatascience.com/a-complete-beginners-guide-to-matrix-multiplication-for-data-science-with-python-numpy-9274ecfc1dc6 ):



## Finding the Incident Radiation Intensity on a Solar Panels

```
def calcTiltedPlaneRadiation(limit, panelAzimuth, panelTilt, radiationData, forecast):
        //Performs calculations on the solar radiation GHI data from the forecast to calculate the actual intensity
        //of light that is incident to the solar panels (the API only returns the solar radiation intensity for horizontal ground).
        //It used values of panel tilt angle and azimuth and location coordinates to perform these calculations.
        solarRadiation = [][]
        JSONObject response = new JSONObject(radiationData) //JSONObject that holds API response
        dayOf Year = 0
        hour = 0
        arrayName = ""
        date = ""
        if forecast:
                arrayName = "forecasts"
        else:
                arrayName = "estimatedActuals"
        for i in range(limit):
                date = response.getJSONArray(arrayName).getJSONObject(i).getString("period_end")
                //gets time and date from json response (needed for calculations)
                dayOf Year = date.getDayOfYear() // uses above date to work out day of year
                hour = date.getHour() // uses above date to work hour of day with a decimal to represent minutes
                elevation = calcElevation(dayOfYear, hour) //uses clacElevation method to work out the elevation of the sun at each time
                azimuth = calcAzimuth(dayOfYear, hour) //uses clacAzimuth method to work out the azimuth of the sun at each time
                ghi = response.getJSONArray(arrayName).getJSONObject(i).getDouble("ghi") //gets GHI value fron API response
                incidentRad = ghi/sin(90-elevation)
                //calcuates the radiation intensity in the direction of the direct path from the sun to a point on the ground
                moduleRad = incidentRad*((cos(90-elevation)*sin(panelTilt)*cos(panelAzimuth-azimuth))+(sin(90-elevation)*cos(panelTilt))
                //calculates the radiation intensity that is incident to the plane of the solar panels
                solarRadiation[i][0] => moduleRad
                time = response.getSONArray(arrayName).getJSONObject().getString("period_end").substring(11, 19)
                time2 = time.minusMinutes (30)
                solarRadiation[i](1) = time
                solarRadiation[i][2] = time2
        return solarRadiation
```

```
calcuations sourced from https://www.pveducation.org/pvcdrom/properties-of-sunlight/the-suns-position
def calcSolarParams (days, hour): //Calculates the current hour angle and declination based upon the current date and time and location
        PVModel pvm -> new PVModel()
        longitude -> pvm.getLongitude()
        x-> (360/365)(days-81)
        equationOfTime -> 9.87*(sin(2*x)) - 7.53(cos(x)) - 1.5*(sin(x)) //calculates EoT
        timeCorrection -> 4*(longitude)-equationOfTime //calculates timeCorrection from EoT
        localSolarTime -> hour + (timeCorrection/60) //calculates the local solar time
        hourAngle -> 15 (localSolarTime-12)   //calculates the hour angle
        declination -> toRadians (3.45 * sin(x)) //calculates the declination of the sun
        return [declination, hour Angle] //returns the hour angle and declination

def calcElevation(days, hour): //Uses the calcSolarParams method and trigonometry to calculate the elevation of the sun above the horizon at a location and time
        PVModel pvm -> new PVModel()
        solarParams -> calcSolarParams (days, hour)
        declination -> solarParams[0]
        hourAngle -> solarParams[1]
        latitude -> pvm.getLatitude()
        elevation -> asin(sin(declination)*sin(latitude) + cos(declination)*cos(latitude)*cos(hourAngle))
        return elevation

def calcAzimuth(days, hour): //Uses the calcSolarParams method and trigonometry to calculate the azimuth of the sun at a location and time
        PVModel pvm -> new PVModel()
        solarParams -> calcSolarParams (days, hour)
        declination -> solarParams[0]
        hourAngle -> solarParams[1]
        latitude -> pvm.getLatitude()
        azimuth -> acos(((sin(declination)* sin(latitude))-(cos(declination) sin(latitude)*cos(hourAngle)))/(cos(calcElevation(days, hour)))))
        return azimuth
```

The calcTiltedRadiation() method takes a GHI value and uses that along with the orientation of the solar panels and the position of the sun to calculate the incident radiation on the panels. It iterates through GHI data JSON to get each value and then that value is recalculated using the calcElevation() and calcAzimuth() methods. The method runs the calcElevation() and calcAzimuth() method which do maths in order to work out the position of the sun in the sky. The mathematics behind the algorithm was sourced from https://www.pveducation.org/pvcdrom/properties-of-sunlight/the-suns-position. The azimuth and elevation calculated by these methods are then next used in the clacTiltedRadiation() method in the calculation of incident solar radiation intensity on the plane of the solar panels.

## Updating the Complex Mathematical Model

```
def updateModel(): //A key function that will update and refine the complex mathematical model with new data that has been collected over the last day.
        DBpvModel db = new DBpvModel()
        ChartsGivEnergyAPI cge = new ChartsGivEnergyAPI(year, month, date)
        ForecastAPI fa = new ForecastAPI()
        actuals = fa.estimatedActuals() //array of estimated actual solar radiation values over the last day
        radData[][] = fa.calcTiltedPlaneRadiation(48, panelAzimuth, panelTilt, actuals, false) //recalculated array of solar raditaion values taking panel orientation into account
        ppv[][] = cge.GETPVpowerByDay(date) //array of the actual power output of the solar panels
        newppv[] = format(radData, ppv)
        newradData[] = removeSecondItem(radData)
        db.addToPVModelData(newradData, newppv) //adds the new collected data to the data table in the database
        db.getModelData()
        PolynomialRegression pr = new PolynomialRegression(db.getGHIdata(), db.getPVoutputdata(), 5, true) //re-evaluates the complex mathematical model with all the newly collected data
        coeff[] = pr.getCoeff()
        db.addToPVModel(coeff[5], coeff[4], coeff[3], coeff[2], coeff[1], coeff[0]) //adds the newly calculated model coefficients to the database
```

This algorithm is used to update and refine the model that is used to predict solar generation over the next day. It can update the model every day with new data points and it performs calculation on new data and stores the raw data and model coefficients into the database tables. Firstly, the estimated actual solar radiation data is fetched from the Solcast API and recalculated to take into account panel tilt and sun position and added into the radData array. Then the actual solar generation from the current day is stored into the ppv array. These datasets are then added into a table in the database that stores all the datapoints of the model. Then calculations are performed on all the model datapoints to re-evaluate the model and find the coefficients of the model. These values are added to the database in a table that stores key values for the model. This process can be repeated every day to continually develop the model in a machine learning fashion.

# Testing

## Inputs

| Test No. | Test | Data | Objectives Tested | Expected Result | Evidence | Test Passed? | Evidence Explanation/Extra Info |
|---|---|---|---|---|---|---|---|
| 1 | Can the program communicate with the Solcast API to get weather forecast data | Solcast API response in JSON format. | 1.1 | Data returned from Solcast API response in JSON format will be represented on a graph with the same values present in the raw data response and in the program. | Video 1 | YES | This video evidence shows JSON data from the API on the left window and the program displaying that data on the right. This proves that the test passed as the data is exactly the same. |
| 2 | Can the program communicate with the octopus API to collect data about the agile electricity prices | Published data from the Octopus energy API | 1.2 | The published data on the Octopus Energy website should match the data in the program. | Video 2 | YES | The video evidence shows the program producing a bar chart that is exactly the same as the published bar chart data on the Octopus Energy website, so it proves that the test has passed. |
| 3 | Can the program communicate with the GivEnergy API to collect data about the solar system. This includes live values and historic power data | Data from the GivEnergy API that the program uses | 1.3 | The graphs displayed in the program should display exactly the same data as the graphs on the GivEnergy website. This should be shown for all the types of graphs. | Video 3<br>Video 4 | YES | Video 3 shows the program producing exactly the same graphs displaying the same data as the graphs on the GivEnergy website and this proves that the test passed.<br>Video 4 shows the program updating the live values continuously over about 15 minutes and the video is sped up. This video proves that the test has passed as it can periodically update live values from the system. |
| 4 | Can the program collect data from a database | Data from these tables in database: | 2.1<br>2.2<br>2.3 | Data from the SystemSettings should be displayed | Screensh ot 1 | YES | Screenshot 1 shows data from pvModel in the database table. Screenshot 2 shows data in the |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | • pvModel<br>• pvModelData<br>• SystemSettings | | in the Recommended settings in the Today section. Data from pvModel and pvModelData tables should be displayed in the model graph. | Screenshot 2<br>Screenshot 3<br>Screenshot 4 | | pvModelData table. Screenshot 3 shows a graph that displays all the data from both these database tables. At the bottom of screenshot 3 the program has also printed out the contents of the pvModel table. Every point on the graph in screenshot 3 is a record in screenshot 2. These screenshots prove the test had passed as the graph in screenshot 3 displays the data that is shown in the tables in screenshot 1 and 2, also at the bottom of screenshot 3 the values shown in screenshot 1 are shown printed out in the console in the IDE which is further proof that the test passed<br><br>Screenshot 4 shows the SystemSettings table at the bottom and then in the Today section of the recommended settings tab above those values are also present. This also proves that the test passed as the data is exactly the same as the values in the program and in the database table, are the same. |
| 5 | Can the program store and save data inputted by the user in the panel details settings tab | • Latitude: 51.07132<br>• Longitude: -1.04239<br>• Panel Tilt: 45 | 3.1<br>3.2<br>3.3<br>3.4<br>4.1 | After the save changes button is pressed the new settings from the form should be | Video 5 (1:00 to end for objective 3, 0:00 to 1:00 for | YES | This video shows both sliders and both text fields being changed and then the appSettings.txt file being opened, and the new values are present in this file as seen by a comparison with all the values. This |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | • Panel Azimuth: 181 | | present in the appSettings.txt file. | objective 4) | | proves the test has passed as the values inputted into the GUI were then viewed in the text file, so the write was successful. |
| 6 | Can the program change the solar PV system modes | Different inputs in rows below | | All the inputs in following rows should be accepted and the corresponding changes must be made and can be verified by viewing the system settings on the GivEnergy website. | | | |
| 6.1 | Can the program change the mode to Mode 1 | Mode 1 | 5.1<br>5.2<br>5.6 | | | YES | When the mode 1 radio button was selected and the save changes button was pressed no errors occurred and the mode change was visible on the GivEnergy website and so the test passed. |
| 6.2 | Can the program change the mode to Mode 2 | Mode 2 | 5.1<br>5.2<br>5.6 | | | YES | When the mode 2 radio button was selected and the save changes button was pressed no errors occurred and the mode change was visible on the GivEnergy website and so the test passed. |
| 6.3 | Can the program change the mode to Mode 3 | • Mode 3<br>• Battery Discharge Start time: 1600 | 5.1<br>5.2<br>5.6 | | | YES | When the mode 3 radio button was selected with discharging times inputted and the save changes button was pressed no errors occurred and the mode change was |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | • Battery Discharge Stop time: 1930 | | | | | visible on the GivEnergy website and so the test passed. |
| 6.4 | Can the program change the mode to Mode 4 | • Mode 4 Selected<br>• Battery Discharge Start time: 1600<br>• Battery Discharge Stop time: 1930 | 5.1<br>5.2<br>5.6 | | | YES | When the mode 4 radio button was selected with discharging times inputted and the save changes button was pressed no errors occurred and the mode change was visible on the GivEnergy website and so the test passed. |
| 6.5 | Can the program set charging periods for the battery | • Charge start time: 1000<br>• Charge Stop time: 1200<br>• Charge battery up to: 80% | 5.4<br>5.6 | | | YES | When the charging times were inputted and the save changes button was pressed no errors occurred and the mode change was visible on the GivEnergy website and so the test passed. |
| 6.6 | Can the program set minimum battery reserve value | • Cut-off value: 4% | 5.5<br>5.6 | | | YES | When minimum battery reserve value was inputted and the save changes button was pressed no errors occurred and the mode change was visible on the GivEnergy website and so the test passed. |
| 6.7 | Can the program revert all changes made in the form back to the original settings before the user made | n/a | 5.7 | | | YES | The form can be reset back to previous values after the “Reset form” button is pressed. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | any adjustments using a “Reset Form” button | | | | | | |
| 7 | Can the program display an interactive power graph where multiple data sets can be added and removed from the graph | The following permutations of data sets were added<br>• Only Import<br>• Only Export<br>• Only Demand<br>• Only Solar Generation<br>• Only Charging/ Discharging<br>• All plots | 6.1<br>6.2 | Power graph should be able to display all 5 of the plots (import, export, demand, solar generation, charging/discharging )<br>Direct comparison to published data on GivEnergy website should show each data plot is correct. | Video 3 | YES | This test is proved to have passed as in Video 3 the graphs displayed in the program are exactly the same as the ones published on the GivEnergy website by direct comparison. |
| 8 | Does the program’s GUI use a structured tabbed pane layout | All tabs will be pressed to test if they work | 7.1<br>7.2<br>7.3<br>10.1<br>10.2 | Each tab in the GUI should be grouped by data displayed and purpose. Each main tab should have sub-tabs that display more specific things. The user should be able to navigate the GUI by pressing on the tabs. | | YES | This test passed well as the GUI design is intuitive and all relevant parts of the GUI are grouped together. There are sub-tabs for each main tab to display more specific information or controls. The user is able to navigate the entire GUI using the tabs. |

## Outputs

| Test No. | Test | Data | Objectives Tested | Expected Result | Evidence | Test Passed? | Extra Info |
|---|---|---|---|---|---|---|---|
| 9.1 | Are all buttons and components in tabs aligned in a logical order and grouped by purpose | n/a | 10.3 | Each form should be designed so that buttons and other input and output components are in a logical layout and grouped by purpose. | | YES | This test passed as the GUI is designed to be grouped in a logical order with similar tabs placed close to each other based on the type of data they display and purpose. |
| 9.2 | Does a loading animation show when the program is starting up | n/a | 10.5 | A loading spinner animation should show for a few seconds as the program is starting up. | Can be seen at 0:36 in Video 5 | YES | At 0:36 video 5 shows the loading spinner which proves that the test has passed. |
| 10 | Does the program display data from Solcast, GivEnergy, Octopus Energy and calculated data such as predicted solar panel power output. | n/a | 11.1<br>11.2<br>11.3<br>11.4<br>11.6 | These data should be displayed in various forms of graphs and charts. | Video 6:<br>• 0:00 objective 11.1<br>• 0:09 objective 11.2<br>• 0:11 objective 11.3<br>• 0:15 objective 11.4<br>• 0:17 objective 11.6.4 | YES | All these objectives were met and so the test passed. Video 6 shows a run through of all the graphs in the program to show that they are all there and display data. This proves the test passed as the video shows all the graphs from the objectives at the |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | • 0:20 objective 11.6.3<br>• 0:21 objective 11.6.2<br>• 0:24 objective 11.6.1 | | timestamps given to the left. |
| 11 | Does the program have an overview tab that displays live values | n/a | 12 | The overview page should display live values from the solar PV system. | Screenshot 5<br>Screenshot 6 | YES | Screenshot 5 shows the live values in the program's GUI and screenshot 6 shows the GivEnergy website showing the same values. This verifies that the live values are correct and proves the test passed. |
| 12 | Can the program write to files | • Latitude: 51.07132<br>• Longitude: -1.04239<br>• Panel Tilt: 45<br>• Panel Azimuth: 181 | 14.1<br>14.2 | The program should be able to write to the appSettings.txt file without producing any errors. | Video 5 (1:00 to end) | YES | This Video 5 from 1:00 to the end proves that the test has passed as the values inputted into the GUI form were successfully written to the appSettings.txt file with no errors and the same data can be viewed in that file. |

## Data Processing

| Test No. | Test | Data | Objectives Tested | Expected Result | Evidence | Test Passed? | Extra Info |
|---|---|---|---|---|---|---|---|
| 13 | Can the program update the complex mathematical model | n/a | 8.1<br>8.2<br>8.3<br>8.4<br>8.5<br>8.6 | After the update model button is pressed the model graph should update with new values added and a new curve of best fit. The power output prediction graph should also change. The database should have also updated. | Video 7 | YES | Video 7 shows the Update Model button being pressed and a new updated graph with more data points being added and a new curve of best fit. This shows that the pvModel table and pvModelData table have updated as well. It proves the test has passed as the graph has been updated with a new curve and more datapoints which shows the database tables have changed and as the curve has changed the model must have been re-calculated. |
| 14 | Can the program calculate recommendations for when the user should charge/discharge the battery. | n/a | 9.1<br>9.2<br>9.3<br>9.4<br>9.6 | This process has worked correctly if the recommended charging and discharging periods for 'Tomorrow' are displayed in the 'Recommended Settings' tab. | Screenshot 7 | YES | This screenshot proves the objective passed as the program is able to calculate and then display the recommended charging and discharging periods for the next day in the 'Tomorrow' section. |

## Error Handling

| Test No. | Test | Data | Objectives Tested | Expected Result | Evidence | Test Passed? | Extra Info |
|---|---|---|---|---|---|---|---|
| 15 | Does the program show an error message if an API response is erroneous | | 1.4 | If an API does not return the expected response for an unknown reason, then the program should display to the user that there has been an issue with that API. | | YES | After inputting a fake API response with incorrect data within it to the program instead of the program actually using the real API, the program showed an error message to the user explaining there was an issue with that API, so the test passed. |
| 16 | Does the program show an error message if the program cannot work fully if there is no internet connection | | 1.5<br>15.4 | If there is no internet connection, then an error message should show to inform the user that the program is unable to connect to certain APIs or databases. | | YES | After disconnecting the device from the internet and loading up the program there were error messages that informed the user that the program was unable to connect. |
| 17 | Does the program show an error message if the inputted values in the system settings tab are outside an acceptable range | | 3.5 | For all the following inputs an error message should show if the longitude is greater than 180 or less than -180. An error should also show if the latitude entered is greater than 90 or less than -90. | | | All the inputs in the following rows for test 17 passed as they produced an error when the inputted coordinate values were outside the permitted range. |
| 17.1 | For the longitude text field | 181 | | | | YES | Test passed as error message showed when longitude values was entered at 181. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17.2 | For the longitude text field | -181 | | | | YES | Test passed as error message showed when longitude values was entered at -181. |
| 17.3 | For the Latitude text field | 91 | | | | YES | Test passed as error message showed when latitude values was entered at 91. |
| 17.4 | For the Latitude text field | -91 | | | | YES | Test passed as error message showed when latitude values was entered at -91. |
| 18 | Does the program show an error message if the time inputted in the system control tab is not of 'HHmm' format | | 5.3 | For all the following inputs an error message should show saying that the format is incorrect. | Video 8 | | All the inputs in the following rows for test 118 passed as they produced an error which told the user the time format was not correct. At 0:44 in video 8 the correct format is entered, and the program does not show an error message. |
| 18.1 | | "19:00" | | | Video 8 (0:00 to 0:11) | YES | This part of Video 8 provides proof that the test passed as the inputted value caused an error message. |
| 18.2 | | "" | | | Video 8 (00:15 to 0:23) | YES | This part of Video 8 provides proof that the test passed as the inputted value caused an error message. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 18.3 | | “19:00:00” | | | Video 8 (0:26 to 0:37) | YES | This part of Video 8 provides proof that the test passed as the inputted value caused an error message. |

## Data Storage

| Test No. | Test | Data | Objectives Tested | Expected Result | Evidence | Test Passed? | Extra Info |
|---|---|---|---|---|---|---|---|
| 19 | Can the program store and access data in a remotely accessed MySQL database. | n/a | 15.1<br>15.2<br>15.3<br>15.5<br>15.6 | The program should be able to access all three database tables (pvModel, pvModelData, SystemSettings) to collect and store data within them without causing any errors. | Screenshot 1<br>Screenshot 2<br>Screenshot 3<br>Screenshot 4 | YES | • Screenshot 1 shows the pvModel table in the database and the values in the highlighted row are also present in screenshot 3 in the terminal output underneath the GUI. This proves that the program is able to access the pvModel table and the pvModelData table as the values in pvModel are derived from the data in pvModelData by the program and so it provides proof that the test passed.<br>• Screenshot 2 shows datapoints in the pvModelData table that are also present in the graph in screenshot 3, this provides further proof that the program is able to access the database as the values directly from the table are present in the graph in screenshot 3.<br>• Screenshot 4 shows the SystemSettings database table at |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | the bottom and the recommended charging and discharging periods shown above. The values in the table are the same as the values in the 'Today' section of the recommendations tab and so that provides further proof that the test passed. |
| 20 | Can the program use a text file to store data | n/a | 16.1 | The program should be able to read and write to a text file to store data such as the system settings and preferences in the app without causing any errors. | Video 5 | YES | Video 5 shows the program successfully reading and writing to the appSettings.txt file and using it to store data about the orientation and location of the solar panels. This proves that the test has passed as the program is able to use a text file to store data. |

## GUI Buttons/Components

Some of these buttons and interactive components may have been an important part of previous tests but those tests where about the processes triggered by using these components not whether the actual components behave correctly. For clarity and to show thorough testing I have tested and included every interactive component in the GUI of my program in the table below.

| Test No. | Button/Component | Objectives Tested | Expected Result | Evidence | Test Passed? | Extra Info |
|---|---|---|---|---|---|---|
| 21 | Power graph date picker | 6.2 | Then the component changes state i.e., the date is changed, then the graph should change which shows the component is working. | Video 9 (0:03) | YES | Video 9 is the only video evidence that explicitly shows this component working and the graph displayed different data for that date and so the evidence proves this test passed. |

| | | | | | | |
|---|---|---|---|---|---|---|
| 22 | Power graph Export radio button | 6.1 | When this radio button is pressed then the export data set should either appear or disappear based on the previous state of the radio button. | Video 3 (0:19) | YES | This radio button was shown working in Video 3 and the export data set was able to be added and removed which proved the test passed. |
| 23 | Power graph Import radio button | 6.1 | When this radio button is pressed then the Import data set should either appear or disappear based on the previous state of the radio button. | Video 3 (0:03) | YES | This radio button was shown working in Video 3 and the Import data set was able to be added and removed which proved the test passed. |
| 24 | Power graph Demanded Power radio button | 6.1 | When this radio button is pressed then the Demanded Power data set should either appear or disappear based on the previous state of the radio button. | Video 3 (0:42) | YES | This radio button was shown working in Video 3 and the Demanded data set was able to be added and removed which proved the test passed. |
| 25 | Power graph Solar Generation radio button | 6.1 | When this radio button is pressed then the Solar Generation data set should either appear or disappear based on the previous state of the radio button. | Video 3 (0:59) | YES | This radio button was shown working in Video 3 and the Solar Generation data set was able to be added and removed which proved the test passed. |
| 26 | Power Graph Charging/Discharging radio button | 6.1 | When this radio button is pressed then the Charging/Discharging data set should either appear or disappear based on the previous state of the radio button. | Video 3 (0:33) | YES | This radio button was shown working in Video 3 and the Charging/Discharging data set was able to be added and removed which proved the test passed. |
| 27 | Octopus Agile prices bar chart date picker | 11.2.1 | Then the component changes state i.e., the date is changed, then the bar chart should change which shows the component is working. | Video 9 (0:11) | YES | This video shows the date picker changing the contents of the bar chart which proves that it works and so the test passed. |
| 28 | Update Model button | 8.1 | This button should cause the model graph underneath it to update with more datapoints and a different curve | Video 7 (0:02) | YES | This video shows the button being pressed and then the graph beneath it updating which proves |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | and if this happens then the button is working correctly. | | | that the button works and so the test passed. |
| 29 | Panel Tilt slider in settings | 3.2 | This slider works correctly if after the save changes button is pressed, the tilt value on the slider is then present in the appSettings.txt file. | Video 5 (1:06) | YES | This video shows the slider being used to set a value and then the same value can be seen in the text file which proves the slider works and the test passed. |
| 30 | Panel Azimuth slider in settings | 3.3 | This slider works correctly if after the save changes button is pressed, the azimuth value on the slider is then present in the appSettings.txt file. | Video 5 (1:10) | YES | This video shows the slider being used to set a value and then the same value can be seen in the text file which proves the slider works and the test passed. |
| 31 | Panel longitude text field in settings | 3.4 | This text field works correctly if after the save changes button is pressed, the longitude value in the text field is then present in the appSettings.txt file. | Video 5 (1:35) | YES | This video shows the text field being used to input a value and then the same value can be seen in the text file which proves the text field works and the test passed. |
| 32 | Panel latitude text field in settings | 3.4 | This text field works correctly if after the save changes button is pressed, the latitude value in the text field is then present in the appSettings.txt file. | Video 5 (1:24) | YES | This video shows the text field being used to input a value and then the same value can be seen in the text file which proves the text field works and the test passed. |
| 33 | Save Changes button in settings | 14.1<br>14.2 | This button works correctly if the appSettings.txt file is updated after it is pressed. | Video 5 (1:42) | YES | This video shows the save changes button being pressed and then the contents of the text file change. This proves the button works and so the test passed. |
| 34 | Automate toggle button in control panel | 5.8 [Extension] | This toggle button is working if when it is in the ON state the program should be viewed to automatically update the modes of the solar PV system. | | NO | This was a test for extension objective 5.8 and the objective was not met as when the toggle button is switched to ON state |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | then the program is not able to change the modes automatically, so the toggle button does not work. The test did not pass. |
| 35 | Mode 1 radio button in control panel | 5.1 | This radio button is working correctly if it unselects all the other mode radio buttons and when the save changes button is pressed, a change to mode 1 can be viewed on the GivEnergy website. | | YES | This test passed as when the mode 1 radio button is selected all the other mode radio buttons unselect and mode 1 is shown on the GivEnergy website. |
| 36 | Mode 2 radio button in control panel | 5.1 | This radio button is working correctly if it unselects all the other mode radio buttons and when the save changes button is pressed, a change to mode 2 can be viewed on the GivEnergy website. | | YES | This test passed as when the mode 2 radio button is selected all the other mode radio buttons unselect and mode 2 is shown on the GivEnergy website. |
| 37 | Mode 3 radio button in control panel | 5.1 | This radio button is working correctly if it unselects all the other mode radio buttons and when the save changes button is pressed, a change to mode 3 can be viewed on the GivEnergy website. | | YES | This test passed as when the mode 3 radio button is selected all the other mode radio buttons unselect and mode 3 is shown on the GivEnergy website. |
| 38 | Mode 3 battery discharge start time text field in control panel | 5.3 | This text field is working correctly if when the save changes button is pressed then the discharge start time inputted into this text field can also be viewed on the GivEnergy website. | Video 8 (0:02) | YES | This video shows a date being inputted into the text field. Then an error shows saying that the date is in the wrong format. This proves that the text field works correctly as the value in it was able to be analysed about its format. Therefore, the test passed. |
| 39 | Mode 3 battery discharge stop time | 5.3 | This text field is working correctly if when the save changes button is | Video 8 (0:04) | YES | This video shows a date being inputted into the text field. Then |

| | | | | | | |
|---|---|---|---|---|---|---|
| | text field in control panel | | pressed then the discharge stop time inputted into this text field can also be viewed on the GivEnergy website. | | | an error shows saying that the date is in the wrong format. This proves that the text field works correctly as the value in it was able to be analysed about its format. Therefore, the test passed. |
| 40 | Mode 4 radio button in control panel | 5.1 | This radio button is working correctly if it unselects all the other mode radio buttons and when the save changes button is pressed, a change to mode 4 can be viewed on the GivEnergy website. | | YES | This test passed as when the mode 4 radio button is selected all the other mode radio buttons unselect and mode 4 is shown on the GivEnergy website. |
| 41 | Mode 4 battery discharge start time text field in control panel | 5.3 | This text field is working correctly if when the save changes button is pressed then the discharge start time inputted into this text field can also be viewed on the GivEnergy website. | | YES | This test passed because the discharge start time in this text field appeared on the GivEnergy website after the save changes button was pressed. This means the text field functioned correctly. |
| 42 | Mode 4 battery discharge stop time text field in control panel | 5.3 | This text field is working correctly if when the save changes button is pressed then the discharge stop time inputted into this text field can also be viewed on the GivEnergy website. | | YES | This test passed because the discharge stop time in this text field appeared on the GivEnergy website after the save changes button was pressed. This means the text field functioned correctly. |
| 43 | Battery Smart Charge toggle button in control panel | 5.4.1 | This toggle button is working if when it is in the OFF state then the text fields in the Independent Features section are disabled and if it is in the ON state then the text fields are enabled. | | YES | This test passed as when this toggle button is pressed to change its state the text fields in the Independent Features do enable/disable based on the state of the toggle button. |
| 44 | AC Charge start time in control panel | 5.4 | This text field is working correctly if when the save changes button is | | YES | This test passed as when the save changes button was pressed the |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | pressed then the AC charge start time inputted into this text field can also be viewed on the GivEnergy website. | | | value in this text field was also viewable on the GivEnergy website. |
| 45 | AC Charge stop time in control panel | 5.4 | This text field is working correctly if when the save changes button is pressed then the AC charge stop time inputted into this text field can also be viewed on the GivEnergy website. | | YES | This test passed as when the save changes button was pressed the value in this text field was also viewable on the GivEnergy website. |
| 46 | Charge battery up to text field in control panel | 5.4 | This text field is working correctly if when the save changes button is pressed, then the value inputted into this text field is also viewable on the GivEnergy website. | | YES | This test passed as when the save changes button was pressed the value in this text field was also viewable on the GivEnergy website. |
| 47 | Battery power reserve text field in control panel | 5.5 | This text field is working correctly if when the save changes button is pressed, then the value inputted into this text field is also viewable on the GivEnergy website. | | YES | This test passed as when the save changes button was pressed the value in this text field was also viewable on the GivEnergy website. |
| 48 | Reset form button in control panel | 5.7 | This text field is working correctly if when it is pressed then the form which is partially edited will return to the state before anything was changed. | | YES | This test passed as when the reset form button was pressed, all the components returned to their original states before any changes were made. |
| 49 | Save changes button in control panel | 5.6 | This button is working correctly if after it is pressed then the setting changes made in the GUI are then visible on the GivEnergy website. | | YES | This test passed as when the button was pressed the settings and mode changes that had been made were then viewable on the GivEnergy website and so the test passed. |

## Video Links

Video 1: https://youtu.be/kN10Gt_Uxlk

Video 2: https://youtube.com/shorts/VrMbyr11Qas?feature=share

Video 3: https://youtu.be/84aG_fa83ps

Video 4: https://youtu.be/H38q6lvpaHE

Video 5: https://youtu.be/-h_NxOfmb3Y

Video 6: https://youtu.be/g1Lvw7ezM48

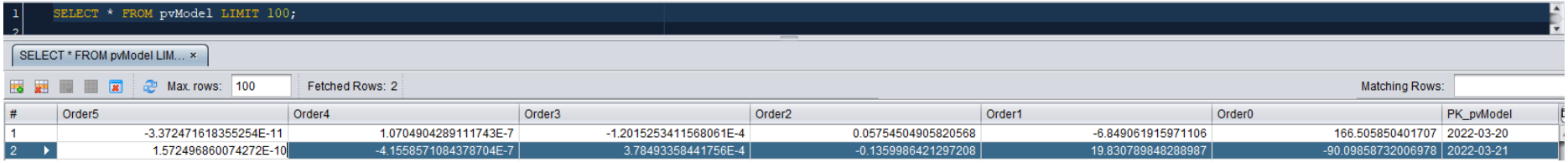
Video 7: https://youtu.be/1XV0uox6k1M

Video 8: https://youtu.be/ol_xo6IZljw

Video 9: https://youtu.be/_t3iDFMwpQs

## Screenshots

### Screenshot 1



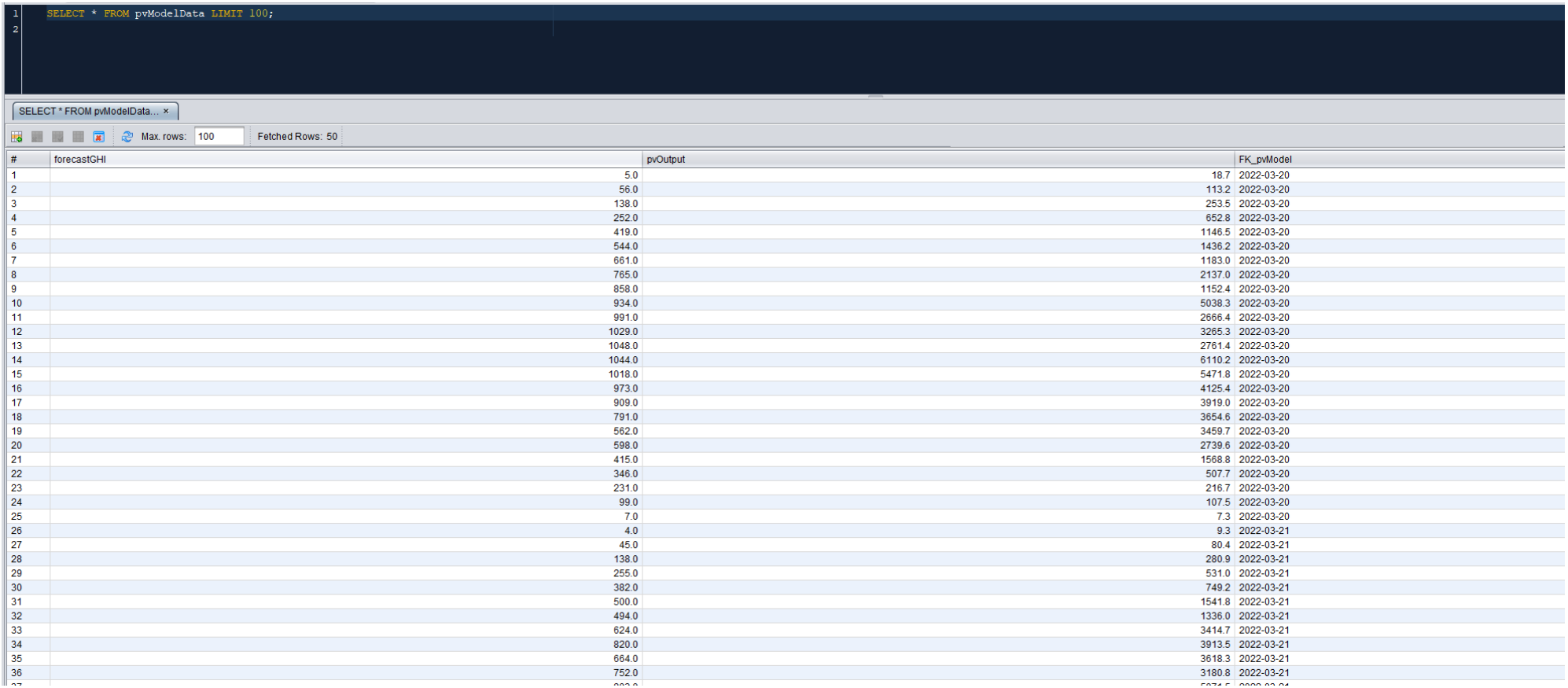
```
1  SELECT * FROM pvModel LIMIT 100;
2
```

SELECT * FROM pvModel LIM... ×

Max. rows: 100 Fetched Rows: 2 Matching Rows:

| # | Order5 | Order4 | Order3 | Order2 | Order1 | Order0 | PK_pvModel |
|---|---|---|---|---|---|---|---|
| 1 | -3.372471618355254E-11 | 1.0704904289111743E-7 | -1.2015253411568061E-4 | 0.05754504905820568 | -6.849061915971106 | 166.505850401707 | 2022-03-20 |
| 2 | 1.572496860074272E-10 | -4.155857108437870 4E-7 | 3.78493358441756E-4 | -0.1359986421297208 | 19.830789848288987 | -90.09858732006978 | 2022-03-21 |

## Screenshot 2



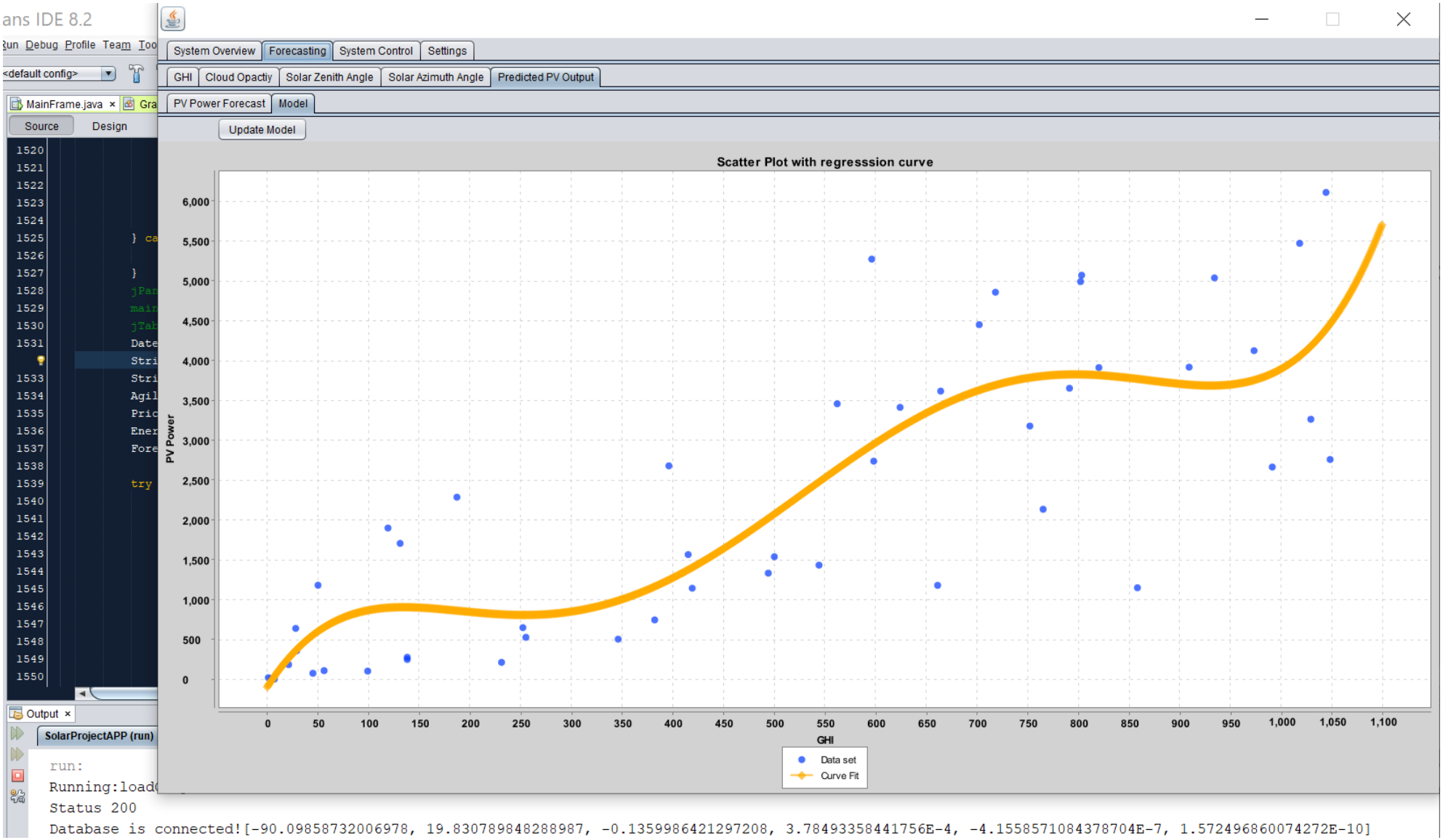
```
1  SELECT * FROM pvModelData LIMIT 100;
2
```

SELECT * FROM pvModelData... ×

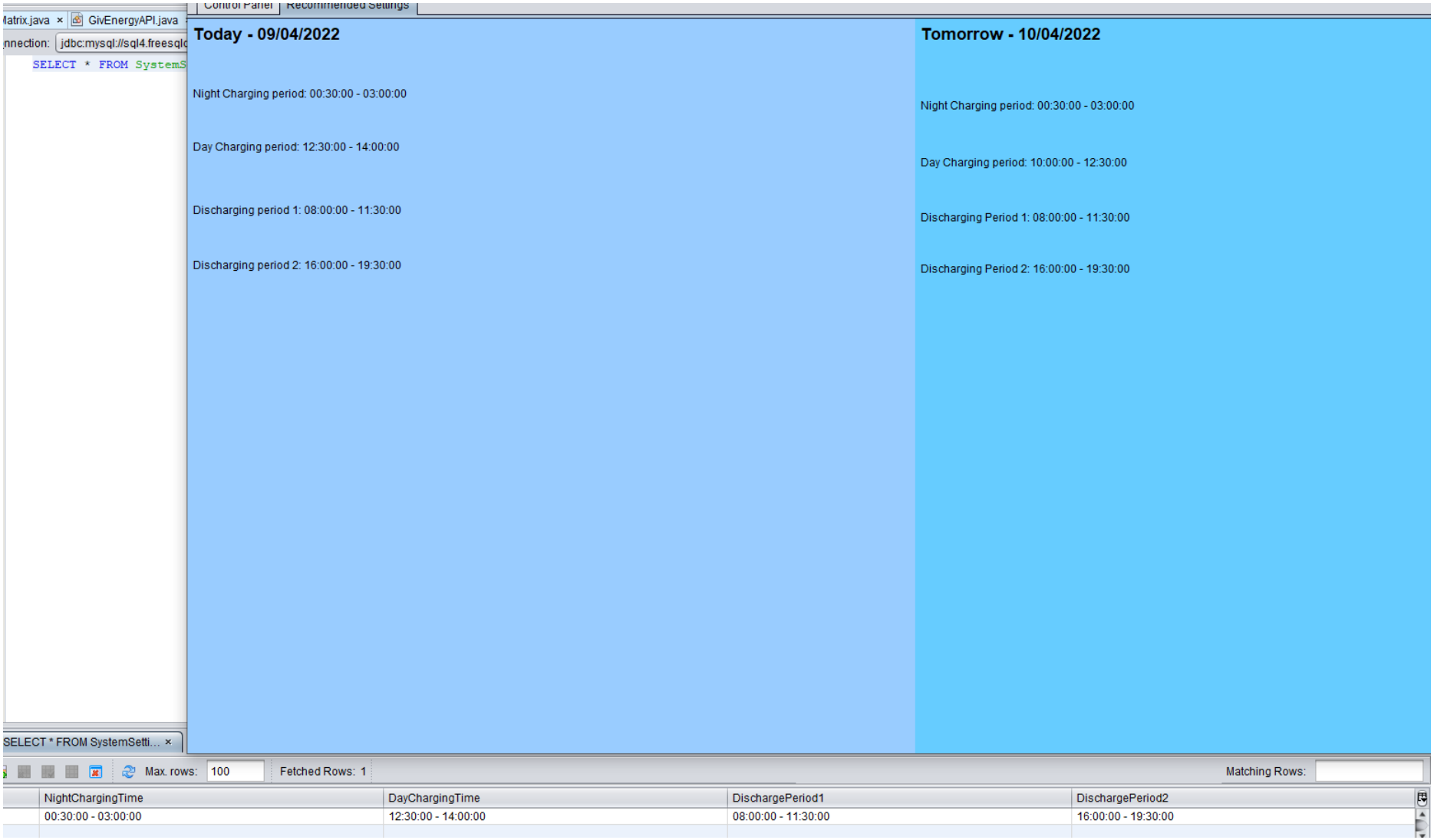
Max. rows: 100 Fetched Rows: 50

| # | forecastGHI | pvOutput | FK_pvModel |
|---|---|---|---|
| 1 | 5.0 | 18.7 | 2022-03-20 |
| 2 | 56.0 | 113.2 | 2022-03-20 |
| 3 | 138.0 | 253.5 | 2022-03-20 |
| 4 | 252.0 | 652.8 | 2022-03-20 |
| 5 | 419.0 | 1146.5 | 2022-03-20 |
| 6 | 544.0 | 1436.2 | 2022-03-20 |
| 7 | 661.0 | 1183.0 | 2022-03-20 |
| 8 | 765.0 | 2137.0 | 2022-03-20 |
| 9 | 858.0 | 1152.4 | 2022-03-20 |
| 10 | 934.0 | 5038.3 | 2022-03-20 |
| 11 | 991.0 | 2666.4 | 2022-03-20 |
| 12 | 1029.0 | 3265.3 | 2022-03-20 |
| 13 | 1048.0 | 2761.4 | 2022-03-20 |
| 14 | 1044.0 | 6110.2 | 2022-03-20 |
| 15 | 1018.0 | 5471.8 | 2022-03-20 |
| 16 | 973.0 | 4125.4 | 2022-03-20 |
| 17 | 909.0 | 3919.0 | 2022-03-20 |
| 18 | 791.0 | 3654.6 | 2022-03-20 |
| 19 | 562.0 | 3459.7 | 2022-03-20 |
| 20 | 598.0 | 2739.6 | 2022-03-20 |
| 21 | 415.0 | 1568.8 | 2022-03-20 |
| 22 | 346.0 | 507.7 | 2022-03-20 |
| 23 | 231.0 | 216.7 | 2022-03-20 |
| 24 | 99.0 | 107.5 | 2022-03-20 |
| 25 | 7.0 | 7.3 | 2022-03-20 |
| 26 | 4.0 | 9.3 | 2022-03-21 |
| 27 | 45.0 | 80.4 | 2022-03-21 |
| 28 | 138.0 | 280.9 | 2022-03-21 |
| 29 | 255.0 | 531.0 | 2022-03-21 |
| 30 | 382.0 | 749.2 | 2022-03-21 |
| 31 | 500.0 | 1541.8 | 2022-03-21 |
| 32 | 494.0 | 1336.0 | 2022-03-21 |
| 33 | 624.0 | 3414.7 | 2022-03-21 |
| 34 | 820.0 | 3913.5 | 2022-03-21 |
| 35 | 664.0 | 3618.3 | 2022-03-21 |
| 36 | 752.0 | 3180.8 | 2022-03-21 |

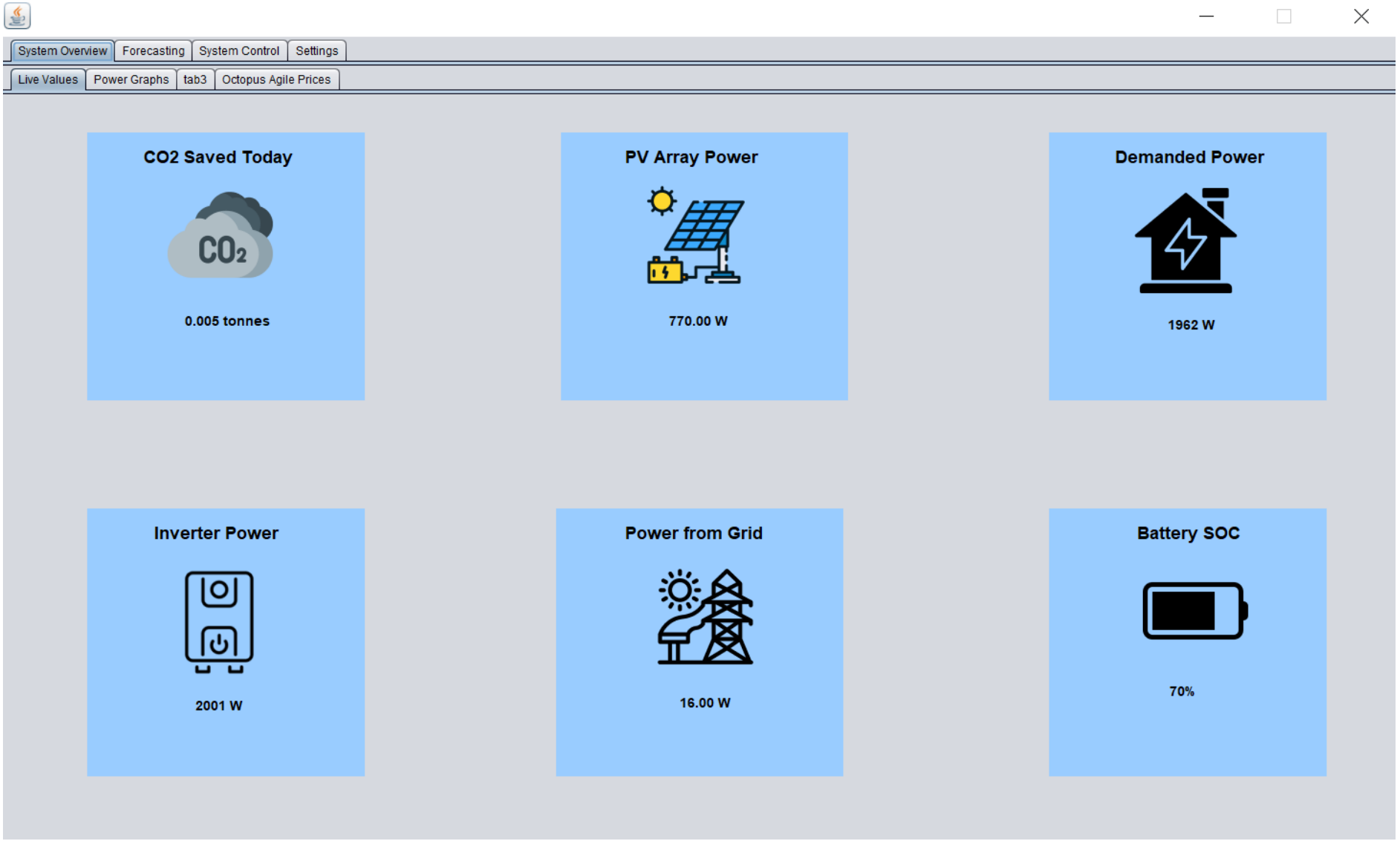
## Screenshot 3



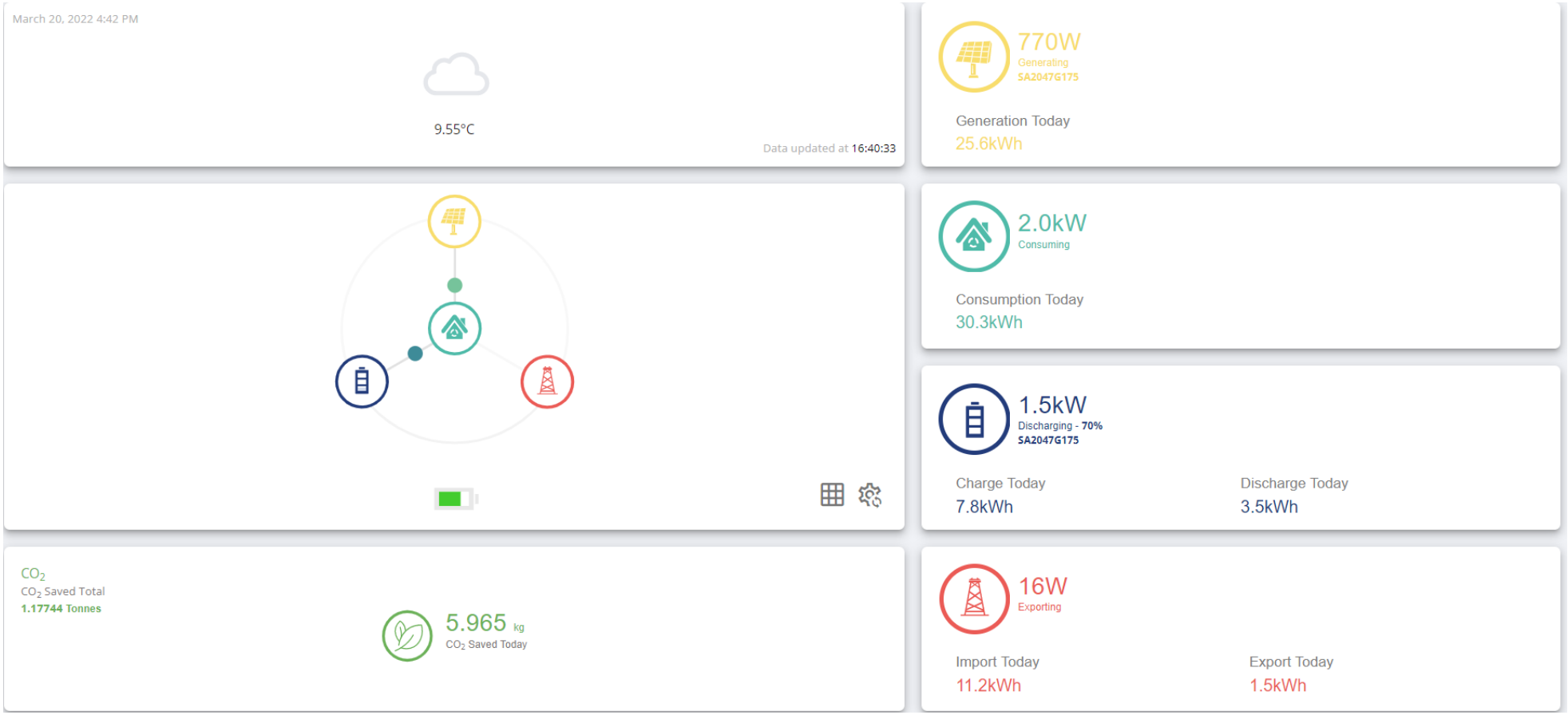
## Screenshot 4



## Screenshot 5



## Screenshot 6



## Technical Solution

### Program Code

MainFrame.java

```
  1 package solarproject;
  2
  3 import databases.DBpvModel;
  4 import givenergy_apis.BatteryGivEnergyAPI;
  5 import optimisation.QuickSort;
  6 import other_apis.ForecastAPI;
  7 import givenergy_apis.CurrentStateGivEnergyAPI;
  8 import givenergy_apis.ChartsGivEnergyAPI;
  9 import java.awt.Toolkit;
 10 import java.io.IOException;
 11 import java.net.URI;
 12 import java.net.URISyntaxException;
 13 import java.net.URL;
 14 import java.text.ParseException;
 15 import java.text.SimpleDateFormat;
 16 import java.time.LocalDate;
 17 import java.time.format.DateTimeFormatter;
 18 import java.util.Calendar;
 19 import java.util.Date;
 20 import java.util.Locale;
 21 import java.util.Timer;
 22 import java.util.TimerTask;
 23 import javax.swing.BoxLayout;
 24 import javax.swing.JPanel;
 25 import model.PVModel;
 26 import optimisation.Energy;
 27 import optimisation.Prices;
 28 import org.json.JSONException;
 29 import org.json.JSONObject;
 30 import other_apis.AgilePricesOctopusAPI;
 31
 32 /**
 33  *
 34  * @author james
 35  */
 36 public class MainFrame extends javax.swing.JFrame
implements Runnable {
 37
 38     private Thread thread; //The thread object
that is used when the program needs to multithread
operations when loading the program
```

```
  39
  40      private boolean[] activePlots = new
boolean[]{true, true, true, true, true}; //This holds
Boolean values for whether each data set in the power
graph is displayed or not displayed on the graph
  41
  42      private boolean loadComplete = false; //Value
that stores whether the program has finished loading to
prevent processes from running if loadComplete is false
  43
  44      public MainFrame() {
  45          initComponents();
  46      }
  47
  48      @SuppressWarnings("unchecked")
  49
  50
#Auto-generated GUI code removed
1470      @Override
1471      public void run() { //This method overrides a
run method in Runnable so that the program can
multithread loading the components into the GUI as well
as performing calculations and displaying a loading
spinner while it is executing.
1472          URL url =
MainFrame.class.getResource("icon.png");//sets the
application icon to replace default java application
icon
1473
setIconImage(Toolkit.getDefaultToolkit().getImage(url))
;
1474          initSettings(); //creates and displays
interative power graph and octopus energy prices bar
chart
1475          initControlPanel(); //creates and displays
the Swing tab that has all the input boxes and buttons
for changing the system modes and settings
1476          initForecastGraphs(); //creates and
displays all the graphs from the Solcast forecasting
API
1477          initLiveValues(); //initially adds the
live values data from GivEnergy to the GUI (these will
be updated after every 2 minutes in a timer)
```

```
1478            initInteractiveGraphs(); //creates and
displays interative power graph and octopus energy
prices bar chart
1479            jPanel3.setVisible(false);
1480            mainTabbedPane.setEnabled(true);
1481            jTabbedPane1.setEnabled(true);
1482            setChargingPeriods(); //runs algorithm to
determine best charging/discharing periods and displays
them
1483            loadComplete = true;//signifies loading
complete
1484        }
1485
1486        private void initControlPanel() { //Uses the
GivEnergy APIs to find the current system settings of
the solar panel and inverter and loads these settings
into the control panel tab of the GUI so that the
correct settings can be displayed
1487            try {
1488                BatteryGivEnergyAPI battery = new
BatteryGivEnergyAPI();
1489                int mode = battery.GETMode();//gets
the current system mode to display at the top
1490                mode = 3;
1491                modeDisplay.setText("Mode currently
set to: " + mode);
1492                switch (mode) { //pre-selects the
correct radio button according to the actual current
mode
1493                    case 1:
1494                        mode1Radio.setSelected(true);
1495                        break;
1496                    case 2:
1497                        mode2Radio.setSelected(true);
1498                        break;
1499                    case 3:
1500                        mode3Radio.setSelected(true);
1501                        break;
1502                    case 4:
1503                        mode4Radio.setSelected(true);
1504                        break;
1505                }
1506                int dischargeStart =
battery.GETdischargeScheduleStart();//gets the actual
discharge start time so it can be added to the GUI
```

```
1507                int dischargeEnd =
battery.GETdischargeScheduleEnd();//gets the actual
discharge end time so it can be added to the GUI
1508
mode3start1.setText(Integer.toString(dischargeStart));
1509
mode3stop1.setText(Integer.toString(dischargeEnd));
1510                mode3start2.setText("");
1511                mode3stop2.setText("");
1512
mode4start1.setText(Integer.toString(dischargeStart));
1513
mode4stop1.setText(Integer.toString(dischargeEnd));
1514                mode4start2.setText("");
1515                mode4stop2.setText("");
1516
chargeStart.setText(Integer.toString(battery.GETchargeS
cheduleStart()));
1517
chargeStop.setText(Integer.toString(battery.GETchargeSc
heduleEnd()));
1518
chargeUpTo.setText(Integer.toString(battery.GETchargeUp
To()));
1519
cutoff.setText(Integer.toString(battery.GETshallowCharg
e()));
1520                Files f = new Files();
1521                String settings = f.read(); //reads
from appSettings.txt
1522                JSONObject jo = new
JSONObject(settings);
1523                boolean independent =
jo.getBoolean("independent"); //if the user wants to be
able to access battery AC charging then this component
is selected
1524                if (independent) {
1525                    smartCharge.setSelected(true);
1526                } else {
1527                    smartCharge.setSelected(false);
1528                }
1529                boolean auto =
jo.getBoolean("automate"); //if the user wants the
program to automate mode changes then this component
would be selected
```

```
1530                if (auto) {
1531                    automate.setSelected(true);
1532                } else {
1533                    automate.setSelected(false);
1534                }
1535            } catch (JSONException ex) {
1536                new Error("There was a problem reading
data.");
1537            }
1538        }
1539
1540        private void initSettings() { //Reads from the
app settings files and sets attributes to those values.
1541            try {
1542                Files files = new Files();
1543                String settings = files.read();
//reads from appSettings.txt which has data in JSON
form
1544                JSONObject jo = new
JSONObject(settings);
1545
panelTiltSlider.setValue(jo.getInt("tilt"));
1546
panelAzimuthSlider.setValue(jo.getInt("azimuth"));
1547
longitude.setText(Double.toString(jo.getDouble("longitu
de")));
1548
latitude.setText(Double.toString(jo.getDouble("latitude
")));
1549            } catch (JSONException ex) {
1550                new Error("There was a problem reading
data.");
1551            }
1552        }
1553
1554        private void initForecastGraphs() { //Uses the
Solcast API and Graph class to add graphs of weather
forecasts to the GUI
1555            ForecastAPI fa = new ForecastAPI();
1556            String forecast = fa.forecast();
1557            double[] ghidata = fa.GETGHI(forecast,
96);//ghi data
1558            double[] clouddata =
fa.GETCloudOpacity(forecast);//cloud opacity data
```

```
1559            double[] azimuthdata =
fa.GETAzimuth(forecast);//solar azimuth data
1560            double[] zenithdata =
fa.GETZenith(forecast);//solar zenith data
1561            double[] ghi90data =
fa.GETGHI90(forecast);//high scenario GHI data
1562            double[] ghi10data =
fa.GETGHI10(forecast);//low scenario GHI data
1563            double[] ppv = fa.GETPPV(forecast);//power
output prediction data
1564            Graph graph = new Graph();//graphs for all
the above datasets are created and displayed in code
below
1565            JPanel ghiplot =
graph.drawLineGraphGHI("GHI Data", "Time", "GHI",
ghidata, ghi10data, ghi90data);
1566            JPanel cloudplot =
graph.drawLineGraph("Cloud Data", "Time", "Cloud
Opacity", clouddata);
1567            JPanel azimuthplot =
graph.drawLineGraph("Solar Azimuth Data", "Time",
"Azimuth", azimuthdata);
1568            JPanel zenithplot =
graph.drawLineGraph("Solar Zenith Data", "Time",
"Zenith", zenithdata);
1569            JPanel ppvplot =
graph.drawLineGraph("Power Output Forecast", "Time",
"Power/kW", ppv);
1570            JPanel scatter = graph.drawScatter("GHI",
"PV Power");
1571            scatter.setVisible(true);
1572            ppvplot.setVisible(true);
1573            modelPanel.setLayout(new
BoxLayout(modelPanel, BoxLayout.PAGE_AXIS));
1574            pvforecast.setLayout(new
BoxLayout(pvforecast, BoxLayout.PAGE_AXIS));
1575            modelPanel.add(scatter);
1576            pvforecast.add(ppvplot);
1577            ghiplot.setVisible(true);
1578            cloudplot.setVisible(true);
1579            azimuthplot.setVisible(true);
1580            zenithplot.setVisible(true);
1581            ghiPanel.setLayout(new BoxLayout(ghiPanel,
BoxLayout.PAGE_AXIS));
1582            ghiPanel.add(ghiplot);
```

```
1583            cloudOpacityPanel.setLayout(new
BoxLayout(cloudOpacityPanel, BoxLayout.PAGE_AXIS));
1584            cloudOpacityPanel.add(cloudplot);
1585            zenithPanel.setLayout(new
BoxLayout(zenithPanel, BoxLayout.PAGE_AXIS));
1586            zenithPanel.add(zenithplot);
1587            azimuthPanel.setLayout(new
BoxLayout(azimuthPanel, BoxLayout.PAGE_AXIS));
1588            azimuthPanel.add(azimuthplot);
1589        }
1590
1591        private void initLiveValues() { //Uses the
GivEnergyAPI to add live values to the GUI live values
tab
1592            try {
1593                CurrentStateGivEnergyAPI currentState
= new CurrentStateGivEnergyAPI();
1594
jLabel7.setText(currentState.GETCO2Today() + "
tonnes");
1595
jLabel8.setText(currentState.GETPVPower() + " W");
1596
jLabel9.setText(Integer.toString(currentState.GETBatter
ySOC()) + "%");
1597
jLabel10.setText(Integer.toString(currentState.GETLoadP
ower()) + " W");
1598
jLabel11.setText(Integer.toString(currentState.GETPower
Inv()) + " W");
1599
jLabel12.setText(currentState.GETGridPower() + " W");
1600            } catch (IOException | JSONException ex) {
1601                new Error("There was a problem
updating live values.");
1602            }
1603        }
1604
1605        private void initInteractiveGraphs() {
//Creates the interactive power graph and Octopus Agile
tariff bar chart to be displayed in the GUI
1606            try {
```

```
1607
exportRadio.setSelected(true);//initialises all radio
buttons to selected
1608                importRadio.setSelected(true);
1609                solarRadio.setSelected(true);
1610                demandPowerRadio.setSelected(true);
1611
chargeDischargeRadio.setSelected(true);
1612                SimpleDateFormat dateformatter = new
SimpleDateFormat("yyyy-MM-dd", Locale.ENGLISH);
1613                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd");
1614                String date =
LocalDate.now().format(formatter);//gets current date
in format that API uses
1615
givEnergyDatePicker.setDate(dateformatter.parse(date));
1616                ChartsGivEnergyAPI cge = new
ChartsGivEnergyAPI(null, null, date);//data from
GivEnergy collected with this class
1617                DateStrings datestrs =
cge.getDateStrs();
1618                Dictionary chartdata =
cge.inverterValuesDictionary(datestrs.getDate());
1619                Graph graph = new Graph();//new graph
for line graph with multiple sets which is created in
line below
1620                JPanel powerGraphDate =
graph.drawLineGraph("Power for: " + datestrs.getDate(),
"Time", "Power/W", new String[]{"Export(W)",
"Import(W)", "Solar Generation(W)", "Demanded
Power(W)", "Charging/Discharging(W)"},
cge.getTimevals(), chartdata.get("pacExport"),
chartdata.get("pacImport"), chartdata.get("ppv"),
chartdata.get("loadpower"),
chartdata.get("batpoweractual"));
1621                AgilePricesOctopusAPI octopusPrices =
new AgilePricesOctopusAPI();
1622                String dateplus =
LocalDate.now().plusDays(1).format(formatter);//date of
next day found for upper time limit in API request
1623                String[][] data =
octopusPrices.GETPrices(date + "T00:00Z", dateplus +
"T00:00Z");//prices fetched from API
```

```
1624                JPanel octopus =
graph.drawBarChart("Agile prices for: " + date, "Time",
"Unit Rate", data);//prices bar chart created
1625                powerGraphDate.setVisible(true);
1626                jPanel2.setLayout(new
BoxLayout(jPanel2, BoxLayout.PAGE_AXIS));
1627                jPanel2.add(powerGraphDate);//power
graph displayed
1628                jPanel4.setLayout(new
BoxLayout(jPanel4, BoxLayout.PAGE_AXIS));
1629                jPanel4.add(octopus);//electricity
prices displayed
1630            } catch (ParseException ex) {
1631                new Error("There was a problem
creating graphs.");
1632            }
1633        }
1634
1635        private void setChargingPeriods() { //Uses
optimisation classes to calculate the optimum
charging/discharging periods and then displays those in
the GUI. Also fetches previously calculated time
periods for the current day from the database.
1636            DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd/MM/yyyy");
1637            String date =
LocalDate.now().format(formatter);
1638            String dateplus =
LocalDate.now().plusDays(1).format(formatter);
1639            jLabel49.setText("Today - " + date);
//updates title of today section with correct date
1640            jLabel50.setText("Tomorrow - " +
dateplus); //updates title of tomorrow section with
correct date
1641            formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd"); //format
used by Octopus Energy API
1642            dateplus =
LocalDate.now().plusDays(1).format(formatter);
1643            AgilePricesOctopusAPI octopusAPI = new
AgilePricesOctopusAPI();
1644            Prices p;
1645            Energy e;
1646            ForecastAPI fa = new ForecastAPI();
1647            try {
```

```
1648                p = new
Prices(octopusAPI.GETPrices(dateplus + "T00:00Z",
dateplus + "T08:00Z")); //night charging period
displayed
1649                String[][] nightPrices =
p.findPrices(true, 5);//finds prices of electricity
over every 2.5 hour period between 00:00 and 08:00 and
orders them from cheapest to most expensive
1650                SimpleDateFormat sdf = new
SimpleDateFormat("HH:mm:ss");
1651                Date d1 =
sdf.parse(nightPrices[0][1].substring(11, 19));
//cheapest period at position [0] in nightPrices
1652                Calendar c = Calendar.getInstance();
1653                c.setTime(d1);
1654                c.add(Calendar.MINUTE, 30);
//adjustment as electricity prices are declared at the
start of each half hour but they last to the end of
each half hour
1655                d1 = c.getTime();
1656                jLabel54.setText("Night Charging
period: " + nightPrices[0][2].substring(11, 19) + " - "
+ d1.toString().substring(11, 19)); //night charging
period displayed
1657                p = new
Prices(octopusAPI.GETPrices(dateplus + "T07:00Z",
dateplus + "T12:00Z"));//loads electricity prices API
response into Prices class
1658                String[][] morningPrices =
p.findPrices(true, 7);//finds prices of electricity
over every 3.5 hour period between 07:00 and 12:00 and
orders them from cheapest to most expensive
1659                int length = morningPrices.length;
1660                d1 = sdf.parse(morningPrices[length -
1][1].substring(11, 19));//most expensive period is
needed as this is when the battery should discharge to
avoid paying for expensive energy
1661                c = Calendar.getInstance();
1662                c.setTime(d1);
1663                c.add(Calendar.MINUTE,
30);//adjustment as electricity prices are declared at
the start of each half hour but they last to the end of
each half hour
1664                d1 = c.getTime();
```

```
1665                jLabel58.setText("Discharging Period
1: " + morningPrices[length - 1][2].substring(11, 19) +
" - " + d1.toString().substring(11, 19));//discharging
period 1 displayed
1666                p = new
Prices(octopusAPI.GETPrices(dateplus + "T15:00Z",
dateplus + "T21:00Z"));//night charging period
displayed
1667                String[][] eveningPrices =
p.findPrices(true, 7);//finds prices of electricity
over every 2.5 hour period between 15:00 and 21:00 and
orders them from cheapest to most expensive
1668                length = eveningPrices.length;
1669                d1 = sdf.parse(eveningPrices[length -
1][1].substring(11, 19));//most expensive 3.5 hour
period needed as this is when the battery should
discharge instead of paying for expensive grid energy
1670                c = Calendar.getInstance();
1671                c.setTime(d1);
1672                c.add(Calendar.MINUTE,
30);//adjustment as electricity prices are declared at
the start of each half hour but they last to the end of
each half hour
1673                d1 = c.getTime();
1674                jLabel57.setText("Discharging Period
2: " + eveningPrices[length - 1][2].substring(11, 19) +
" - " + d1.toString().substring(11, 19));//discharging
period 2 displayed
1675                p = new
Prices(octopusAPI.GETPrices(dateplus + "T08:00Z",
dateplus + "T16:00Z"));//loads electricity prices API
response into Prices class
1676                String[][] dayPrices =
p.findPrices(false, 5);//finds prices of every 2.5 hour
period between 08:00 and 16:00 to import energy
1677                String forecast = fa.forecast(); //new
weather forecast
1678                e = new
Energy(fa.PPVForDayRecommendations(forecast)); //array
of predicted power values over next day added to Energy
class
1679                e.findEnergy();//power values
converted to energy generated over each period
```

```
1680                String[][] ratios =
e.findRatios(dayPrices);//ratios between price and
energy generated in each 2.5 period found
1681                QuickSort q = new QuickSort();
1682                q.quickSort(ratios, 0, ratios.length -
1);
1683                d1 =
sdf.parse(ratios[0][1].substring(11, 19)); //smallest
ratio used as that is most optimal
1684                c.setTime(d1);
1685                c.add(Calendar.MINUTE,
30);//adjustment as electricity prices are declared at
the start of each half hour but they last to the end of
each half hour
1686                d1 = c.getTime();
1687                jLabel53.setText("Day Charging period:
" + ratios[0][2].substring(11, 19) + " - " +
d1.toString().substring(11, 19));//day charging period
displayed
1688            } catch (ParseException |
NegativeArraySizeException ex) {//if energy prices are
not yet published at the time of viewing then a message
is added instead
1689                jLabel54.setText("Night Charging
period: " + "Octopus Energy Prices Not Availiable");
1690                jLabel53.setText("Day Charging period:
" + "Octopus Energy Prices Not Availiable");
1691                jLabel58.setText("Discharging Period
1: " + "Octopus Energy Prices Not Availiable");
1692                jLabel57.setText("Discharging Period
2: " + "Octopus Energy Prices Not Availiable");
1693            }
1694            DBpvModel dbpv = new DBpvModel();
1695            String[] periods =
dbpv.getChargingPeriods(); //charging/discharging
periods calculated yesterday for today then added to
today section
1696            jLabel51.setText("Night Charging period: "
+ periods[0]);
1697            jLabel52.setText("Day Charging period: " +
periods[1]);
1698            jLabel55.setText("Discharging period 1: "
+ periods[2]);
1699            jLabel56.setText("Discharging period 2: "
+ periods[3]);
```

```
1700      }
1701
1702      public void start() {
1703          if (thread == null) {
1704              thread = new Thread(this,
"loadComponents");
1705              thread.start();
1706          }
1707      }
1708
1709      private void
formWindowOpened(java.awt.event.WindowEvent evt) {
1710          //This method is called when the program
is started and the window is opened. It runs other
methods to initialise the GUI and it uses
multithreading to continually update the live system
values tab while the program is running.
1711          loading.setVisible(true); //shows loading
animation
1712          mainTabbedPane.setEnabled(false); //blocks
user from being able to click on components until it is
finished loading
1713          jTabbedPane1.setEnabled(false); //blocks
user from being able to click on components until it is
finished loading
1714          start(); //begins the thread that loads
all the components into the GUI which takes a few
seconds
1715          Timer timer = new Timer();//all code below
in this method is used to repeatedly update the live
system data that is displayed in the GUI
1716          TimerTask tt = new TimerTask() {
1717              @Override
1718              public void run() {
1719                  try {
1720                      CurrentStateGivEnergyAPI
currentState = new CurrentStateGivEnergyAPI();
1721
jLabel7.setText(currentState.GETCO2Today() + "
tonnes");
1722
jLabel8.setText(currentState.GETPVPower() + " W");
1723
jLabel9.setText(Integer.toString(currentState.GETBatter
ySOC()) + "%");
```

```
1724
jLabel10.setText(Integer.toString(currentState.GETLoadP
ower()) + " W");
1725
jLabel11.setText(Integer.toString(currentState.GETPower
Inv()) + " W");
1726
jLabel12.setText(currentState.GETGridPower() + " W");
1727                     } catch (IOException |
JSONException ex) {
1728                         new Error("There was a problem
updating live values.");
1729                     }
1730                 }
1731             };
1732             timer.schedule(tt, 120000, 120000);
//every 2 minutes = 120000ms
1733         }
1734
1735         private void
jLabel18MouseClicked(java.awt.event.MouseEvent evt) {
1736             try {
1737                 URI uri = new
URI("https://epsg.io/4326"); ////clickable link to
inform user about coordinate system being used
1738
java.awt.Desktop.getDesktop().browse(uri.normalize());
1739             } catch (URISyntaxException | IOException
ex) {
1740                 new Error("There was a problem please
try again.");
1741             }
1742         }
1743
1744         private void
jLabel19MouseClicked(java.awt.event.MouseEvent evt) {
1745             try {
1746                 URI uri = new
URI("https://epsg.io/4326"); //clickable link to inform
user about coordinate system being used
1747
java.awt.Desktop.getDesktop().browse(uri.normalize());
1748             } catch (URISyntaxException | IOException
ex) {
```

```
1749                new Error("There was a problem please
try again.");
1750            }
1751        }
1752
1753        private void
saveChangesButtonActionPerformed(java.awt.event.ActionE
vent evt) {
1754            if ((Integer.parseInt(longitude.getText())
> 180) | (Integer.parseInt(longitude.getText()) < -180)
| (Integer.parseInt(latitude.getText()) < -90) |
(Integer.parseInt(latitude.getText()) > 90)) { //code
to verify that inputted location values are within the
correct range
1755                new Error("Longitude must be in range
[-180 to 180] and latitude must be in range [-90 to
90]");
1756                return;
1757            }
1758            try {
1759                Files f = new Files();//data from
appSettings.txt in JSON format
1760                String settings = f.read();
1761                JSONObject jo = new
JSONObject(settings);
1762                jo.put("tilt",
panelTiltSlider.getValue());
1763                jo.put("azimuth",
panelAzimuthSlider.getValue());
1764                jo.put("longitude",
longitude.getText());
1765                jo.put("latitude",
latitude.getText());
1766                PVModel pvm = new PVModel();
1767                pvm.updateParams();
1768                f.clear();
1769                f.write(jo.toString());
1770            } catch (JSONException ex) {
1771                new Error("There was a problem reading
data please try again.");
1772            }
1773        }
1774
```

```
1775     private void octopusDatePickerPropertyChange(java.beans.PropertyChangeEvent evt) {
1776         //This method is run when the Octopus energy prices date picker component is changed to a new date. It updates the bar chart with data from the day selelected
1777         if (octopusDatePicker.getDate() == null) {
1778             return;
1779         }
1780         SimpleDateFormat format = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzz yyyy"); //date format that the date picker component uses
1781         try {
1782             Date date = format.parse(octopusDatePicker.getDate().toString());
1783             SimpleDateFormat newformat = new SimpleDateFormat("yyyy-MM-dd"); //date format that the Octopus Energy API uses
1784             String newdateStr = newformat.format(date);
1785             Calendar c = Calendar.getInstance();
1786             c.setTime(newformat.parse(newdateStr.toString()));
1787             c.add(Calendar.DATE, 1); //need to find the date of the next day so upper time limit can be inputted into API
1788             String dateplus = newformat.format(c.getTime());
1789             AgilePricesOctopusAPI octopusAPI = new AgilePricesOctopusAPI();
1790             Graph graph = new Graph();
1791             String[][] data = octopusAPI.GETPrices(newdateStr + "T00:00Z", dateplus + "T00:00Z"); //prices for the date selected got from Octopus API
1792             JPanel octopus = graph.drawBarChart("Agile prices for: " + newdateStr, "Time", "Unit Rate", data); //data represented as a bar chart
1793             jPanel4.remove(1);
1794             jPanel4.setLayout(new BoxLayout(jPanel4, BoxLayout.PAGE_AXIS));
1795             jPanel4.add(octopus);
1796         } catch (Exception ex) {
```

```
1797                 new Error("There was a problem finding
data for this date.");
1798             }
1799         }
1800
1801         private void
givEnergyDatePickerPropertyChange(java.beans.PropertyCh
angeEvent evt) {
1802             try { //when new date selected new power
graph is drawn for that new day
1803                 drawPowerGraph();
1804             } catch (Exception ex) {
1805                 new Error("There was a problem finding
data for this date.");
1806             }
1807         }
1808
1809         private void
exportRadioActionPerformed(java.awt.event.ActionEvent
evt) {
1810             activePlots[0] = !activePlots[0]; //power
graph updated when user selects/unselects export
dataset
1811             drawPowerGraph();
1812         }
1813
1814         private void
importRadioActionPerformed(java.awt.event.ActionEvent
evt) {
1815             activePlots[1] = !activePlots[1]; //power
graph updated when user selects/unselects import
dataset
1816             drawPowerGraph();
1817         }
1818
1819         private void
demandPowerRadioActionPerformed(java.awt.event.ActionEv
ent evt) {
1820             activePlots[3] = !activePlots[3]; //power
graph updated when user selects/unselects demand power
dataset
1821             drawPowerGraph();
1822         }
1823
```

```
1824      private void
solarRadioActionPerformed(java.awt.event.ActionEvent
evt) {
1825          activePlots[2] = !activePlots[2]; //power
graph updated when user selects/unselects solar
generation power dataset
1826          drawPowerGraph();
1827      }
1828
1829      private void
chargeDischargeRadioActionPerformed(java.awt.event.Acti
onEvent evt) {
1830          activePlots[4] = !activePlots[4]; //power
graph updated when user selects/unselects
charging/discharging dataset
1831          drawPowerGraph();
1832      }
1833
1834      private void
mode1RadioActionPerformed(java.awt.event.ActionEvent
evt) {
1835          if (mode1Radio.isSelected()) { //selects
the mode 1 radio button and unselects all the others
1836              mode2Radio.setSelected(false);
1837              mode3Radio.setSelected(false);
1838              mode4Radio.setSelected(false);
1839          }
1840      }
1841
1842      private void
mode2RadioActionPerformed(java.awt.event.ActionEvent
evt) {
1843          if (mode2Radio.isSelected()) { //selects
the mode 2 radio button and unselects all the others
1844              mode1Radio.setSelected(false);
1845              mode3Radio.setSelected(false);
1846              mode4Radio.setSelected(false);
1847          }
1848      }
1849
1850      private void
mode4RadioActionPerformed(java.awt.event.ActionEvent
evt) {
1851          if (mode4Radio.isSelected()) { //selects
the mode 4 radio button and unselects all the others
```

```
1852                mode1Radio.setSelected(false);
1853                mode2Radio.setSelected(false);
1854                mode3Radio.setSelected(false);
1855            }
1856        }
1857
1858        private void
mode3RadioActionPerformed(java.awt.event.ActionEvent
evt) {
1859            if (mode3Radio.isSelected()) { //selects
the mode 3 radio button and unselects all the others
1860                mode1Radio.setSelected(false);
1861                mode2Radio.setSelected(false);
1862                mode4Radio.setSelected(false);
1863            }
1864        }
1865
1866        private void
saveChangesActionPerformed(java.awt.event.ActionEvent
evt) {
1867            //When the save changes button is pressed
then this method is called and it uses the GivEnergy
API to update the system settings from what the user
inputted into the form
1868            BatteryGivEnergyAPI battery = new
BatteryGivEnergyAPI();
1869            if (smartCharge.isSelected() &
("".equals(chargeStart.getText()) |
"".equals(chargeStop.getText()) |
"".equals(chargeUpTo.getText()) |
"".equals(cutoff.getText()))) {
1870                new Error("Some Items in Independent
Features have no values");
1871                return;
1872            }
1873            if (mode1Radio.isSelected()) {
1874                battery.modeChange(1); //changes to
mode 1
1875            }
1876            if (mode2Radio.isSelected()) {
1877                battery.modeChange(2); //changes to
mode 2
1878            }
1879            if (mode3Radio.isSelected()) {
```

```
1880                battery.modeChange(3); //changes to
mode 3
1881                if
((validateDateFormat(mode3start1.getText()) &&
validateDateFormat(mode3stop1.getText())) == false) {
//checks if user has imputted times of the correct
format
1882                    new Error("Inputted times must be
of HHmm format. Try again.");
1883                    return;
1884                }
1885
battery.discharge(Integer.parseInt(mode3start1.getText(
)), Integer.parseInt(mode3stop1.getText()), 4, true);
//discharges battery in the periods entered by user
1886            }
1887            if (mode4Radio.isSelected()) {
1888                battery.modeChange(4); //changes to
mode 4
1889                if
((validateDateFormat(mode4start1.getText()) &&
validateDateFormat(mode4stop1.getText())) == false) {
//checks if user has imputted times of the correct
format
1890                    new Error("Inputted times must be
of HHmm format. Try again.");
1891                    return;
1892                }
1893
battery.discharge(Integer.parseInt(mode4start1.getText(
)), Integer.parseInt(mode4stop1.getText()), 4, true);
//discharges battery in the periods entered by user
1894            }
1895
battery.charge(Integer.parseInt(chargeStart.getText()),
Integer.parseInt(chargeStop.getText()),
Integer.parseInt(chargeUpTo.getText()));
1896
battery.shallowCharge(Integer.parseInt(cutoff.getText()
));
1897        }
1898
1899        private void
smartChargeActionPerformed(java.awt.event.ActionEvent
evt) {
```

```
1900            if (smartCharge.isSelected()) {
1901                smartCharge.setText("ON");
1902                chargeStart.setEnabled(true);
1903                chargeStop.setEnabled(true);
1904                chargeUpTo.setEnabled(true);
1905            }
1906            if (!smartCharge.isSelected()) {
1907                smartCharge.setText("OFF");
1908                chargeStart.setEnabled(false);
1909                chargeStop.setEnabled(false);
1910                chargeUpTo.setEnabled(false);
1911            }
1912        }
1913
1914        private void updateModelActionPerformed(java.awt.event.ActionEvent evt) {
1915            //When the update model button is pressed this method is called and uses methods in pvModel to update the mathematical model and then replaces old model graphs with new updated ones
1916            PVModel pv = new PVModel();
1917            pv.updateModel(); //updates the databases with new data points and calculated model coefficients
1918            ForecastAPI fa = new ForecastAPI();
1919            String forecast = fa.forecast(); //new forecast to update the prediced solar power output
1920            double[] ppv = fa.GETPPV(forecast); //uses new model to create new set of predicted power values
1921            Graph graph = new Graph();
1922            modelPanel.remove(1);
1923            modelPanel.repaint();
1924            JPanel ppvplot = graph.drawLineGraph("Power Output Forecast", "Time", "Power/kW", ppv); //new updated graph
1925            JPanel scatter = graph.drawScatter("GHI", "PV Power"); //new updated graph
1926            scatter.setVisible(true);
1927            ppvplot.setVisible(true);
1928            modelPanel.setLayout(new BoxLayout(modelPanel, BoxLayout.PAGE_AXIS));
1929            modelPanel.add(scatter);
1930        }
1931
```

```
1932      private void
resetFormActionPerformed(java.awt.event.ActionEvent
evt) {
1933          initControlPanel();
1934      }
1935      private void drawPowerGraph() { //This method
uses the graph class to produces a special type of
graph where datasets can be added and removed from the
axis and multiple dataset can be displayed over each
other.
1936          try {
1937              if ((givEnergyDatePicker.getDate() ==
null) | loadComplete == false) {
1938                  return;
1939              }
1940              SimpleDateFormat format = new
SimpleDateFormat("EEE MMM dd HH:mm:ss zzz yyyy");
//format used by date picker component
1941              Date date =
format.parse(givEnergyDatePicker.getDate().toString());
1942              SimpleDateFormat newformat = new
SimpleDateFormat("yyyy-MM-dd");
1943              String newdateStr =
newformat.format(date); //converts format into format
that givEnergy use
1944              int length = activePlots.length;
1945              ChartsGivEnergyAPI cge = new
ChartsGivEnergyAPI(null, null, newdateStr);
1946              DateStrings datestrs =
cge.getDateStrs();
1947              Dictionary chartdata =
cge.inverterValuesDictionary(datestrs.getDate());
//gets the dictionary object of titles of datasets and
datasets
1948              Graph graph = new Graph(); //new graph
1949              String[] options = new
String[]{"pacExport", "pacImport", "ppv", "loadpower",
"batpoweractual"}; //keys of values in API response
that are needed
1950              String[] dataTitles = new
String[]{"Export(W)", "Import(W)", "Solar
Generation(W)", "Demanded Power(W)",
"Charging/Discharging(W)"}; //titles of datasets on the
graph
```

```
1951                double[][] parameters = new
double[length][];
1952                for (int i = 0; i <
activePlots.length; i++) { //gets the datasets out the
dictionary and into an array called parameters in order
1953                    if (activePlots[i]) {
1954                        parameters[i] =
chartdata.get(options[i]);
1955                    } else {
1956                        dataTitles[i] = null;
1957                    }
1958                }
1959                JPanel powerGraphDate =
graph.drawLineGraph("Power for: " + datestrs.getDate(),
"Time", "Power/W", dataTitles, cge.getTimevals(),
parameters[0], parameters[1], parameters[2],
parameters[3], parameters[4]);
1960                jPanel2.remove(1);//removes old graph
1961                jPanel2.repaint();
1962                jPanel2.setLayout(new
BoxLayout(jPanel2, BoxLayout.PAGE_AXIS));
1963                jPanel2.add(powerGraphDate);//adds new
graph
1964                jPanel2.repaint();
1965                jPanel2.revalidate();
1966            } catch (ParseException ex) {
1967                new Error("There was a problem
displaying the graph.");
1968            }
1969        }
1970
1971        public boolean validateDateFormat(String
dateToValidate) { //Tests to see if a date String is in
a particular format by parsing that string into the
required format and if no error occurs it returns true
as the date is in the correct format
1972            SimpleDateFormat formatter = new
SimpleDateFormat("HHmm");
1973            formatter.setLenient(false);
1974            try {
1975                Date parsedDate =
formatter.parse(dateToValidate);
1976                return true;
```

```
1977          } catch (ParseException e) { //if error
then date input was not of required format so return
false
1978              return false;
1979          }
1980      }
1981
1982      public static void main(String args[]) {
1983          try {
1984              for
(javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
1985                  if
("Nimbus".equals(info.getName())) {
1986
javax.swing.UIManager.setLookAndFeel(info.getClassName(
));
1987                      break;
1988                  }
1989              }
1990          } catch (ClassNotFoundException |
InstantiationException | IllegalAccessException |
javax.swing.UnsupportedLookAndFeelException ex) {
1991              new Error("Error loading program,
please restart program");
1992          }
1993          java.awt.EventQueue.invokeLater(new
Runnable() {
1994
1995              @Override
1996              public void run() {
1997                  new MainFrame().setVisible(true);
1998              }
1999          });
2000      }
2001
2002      // Variables declaration - do not modify
2003      private javax.swing.JPanel SystemControlPanel;
2004      private javax.swing.JPanel TomorrowRec;
2005      private javax.swing.JPanel airtempPanel;
2006      private javax.swing.JToggleButton automate;
2007      private javax.swing.JPanel azimuthPanel;
2008      private javax.swing.JPanel batteryInfo;
2009      private javax.swing.JRadioButton
chargeDischargeRadio;
```

```
2010     private javax.swing.JTextField chargeStart;
2011     private javax.swing.JTextField chargeStop;
2012     private javax.swing.JTextField chargeUpTo;
2013     private javax.swing.JPanel cloudOpacityPanel;
2014     private javax.swing.JPanel co2info;
2015     private javax.swing.JPanel consumptionInfo;
2016     private javax.swing.JPanel controlPanel;
2017     private javax.swing.JTextField cutoff;
2018     private javax.swing.JRadioButton
demandPowerRadio;
2019     private javax.swing.JRadioButton exportRadio;
2020     private javax.swing.JPanel forecastingPanel;
2021     private javax.swing.JTabbedPane
forecastingTabbedPane;
2022     private javax.swing.JPanel
generalsettingsPanel;
2023     private javax.swing.JPanel ghiPanel;
2024     private com.toedter.calendar.JDateChooser
givEnergyDatePicker;
2025     private javax.swing.JPanel gridInfo;
2026     private javax.swing.JRadioButton importRadio;
2027     private javax.swing.JPanel invInfo;
2028     private javax.swing.JLabel jLabel1;
2029     private javax.swing.JLabel jLabel10;
2030     private javax.swing.JLabel jLabel11;
2031     private javax.swing.JLabel jLabel12;
2032     private javax.swing.JLabel jLabel13;
2033     private javax.swing.JLabel jLabel14;
2034     private javax.swing.JLabel jLabel15;
2035     private javax.swing.JLabel jLabel16;
2036     private javax.swing.JLabel jLabel17;
2037     private javax.swing.JLabel jLabel18;
2038     private javax.swing.JLabel jLabel19;
2039     private javax.swing.JLabel jLabel2;
2040     private javax.swing.JLabel jLabel21;
2041     private javax.swing.JLabel jLabel22;
2042     private javax.swing.JLabel jLabel23;
2043     private javax.swing.JLabel jLabel24;
2044     private javax.swing.JLabel jLabel25;
2045     private javax.swing.JLabel jLabel26;
2046     private javax.swing.JLabel jLabel27;
2047     private javax.swing.JLabel jLabel28;
2048     private javax.swing.JLabel jLabel29;
2049     private javax.swing.JLabel jLabel3;
2050     private javax.swing.JLabel jLabel30;
```

```
2051     private javax.swing.JLabel jLabel31;
2052     private javax.swing.JLabel jLabel32;
2053     private javax.swing.JLabel jLabel33;
2054     private javax.swing.JLabel jLabel34;
2055     private javax.swing.JLabel jLabel35;
2056     private javax.swing.JLabel jLabel36;
2057     private javax.swing.JLabel jLabel37;
2058     private javax.swing.JLabel jLabel38;
2059     private javax.swing.JLabel jLabel39;
2060     private javax.swing.JLabel jLabel4;
2061     private javax.swing.JLabel jLabel40;
2062     private javax.swing.JLabel jLabel41;
2063     private javax.swing.JLabel jLabel42;
2064     private javax.swing.JLabel jLabel43;
2065     private javax.swing.JLabel jLabel44;
2066     private javax.swing.JLabel jLabel45;
2067     private javax.swing.JLabel jLabel46;
2068     private javax.swing.JLabel jLabel47;
2069     private javax.swing.JLabel jLabel48;
2070     private javax.swing.JLabel jLabel49;
2071     private javax.swing.JLabel jLabel5;
2072     private javax.swing.JLabel jLabel50;
2073     private javax.swing.JLabel jLabel51;
2074     private javax.swing.JLabel jLabel52;
2075     private javax.swing.JLabel jLabel53;
2076     private javax.swing.JLabel jLabel54;
2077     private javax.swing.JLabel jLabel55;
2078     private javax.swing.JLabel jLabel56;
2079     private javax.swing.JLabel jLabel57;
2080     private javax.swing.JLabel jLabel58;
2081     private javax.swing.JLabel jLabel6;
2082     private javax.swing.JLabel jLabel7;
2083     private javax.swing.JLabel jLabel8;
2084     private javax.swing.JLabel jLabel9;
2085     private javax.swing.JPanel jPanel1;
2086     private javax.swing.JPanel jPanel2;
2087     private javax.swing.JPanel jPanel3;
2088     private javax.swing.JPanel jPanel4;
2089     private javax.swing.JPanel jPanel5;
2090     private javax.swing.JPanel jPanel6;
2091     private javax.swing.JRadioButton
jRadioButton5;
2092     private javax.swing.JRadioButton
jRadioButton6;
```

```
2093     private javax.swing.JRadioButton
jRadioButton7;
2094     private javax.swing.JRadioButton
jRadioButton8;
2095     private javax.swing.JSeparator jSeparator1;
2096     private javax.swing.JSeparator jSeparator10;
2097     private javax.swing.JSeparator jSeparator11;
2098     private javax.swing.JSeparator jSeparator12;
2099     private javax.swing.JSeparator jSeparator14;
2100     private javax.swing.JSeparator jSeparator15;
2101     private javax.swing.JSeparator jSeparator2;
2102     private javax.swing.JSeparator jSeparator3;
2103     private javax.swing.JSeparator jSeparator4;
2104     private javax.swing.JSeparator jSeparator7;
2105     private javax.swing.JSeparator jSeparator8;
2106     private javax.swing.JSeparator jSeparator9;
2107     private javax.swing.JTabbedPane jTabbedPane1;
2108     private javax.swing.JTabbedPane jTabbedPane2;
2109     private javax.swing.JTabbedPane jTabbedPane3;
2110     private javax.swing.JTabbedPane jTabbedPane4;
2111     private javax.swing.JTextField jTextField1;
2112     private javax.swing.JTextField jTextField11;
2113     private javax.swing.JTextField jTextField12;
2114     private javax.swing.JTextField jTextField13;
2115     private javax.swing.JTextField jTextField14;
2116     private javax.swing.JTextField latitude;
2117     private javax.swing.JLabel loading;
2118     private javax.swing.JTextField longitude;
2119     private javax.swing.JTabbedPane
mainTabbedPane;
2120     private javax.swing.JRadioButton mode1Radio;
2121     private javax.swing.JRadioButton mode2Radio;
2122     private javax.swing.JRadioButton mode3Radio;
2123     private javax.swing.JTextField mode3start1;
2124     private javax.swing.JTextField mode3start2;
2125     private javax.swing.JTextField mode3stop1;
2126     private javax.swing.JTextField mode3stop2;
2127     private javax.swing.JRadioButton mode4Radio;
2128     private javax.swing.JTextField mode4start1;
2129     private javax.swing.JTextField mode4start2;
2130     private javax.swing.JTextField mode4stop1;
2131     private javax.swing.JTextField mode4stop2;
2132     private javax.swing.JLabel modeDisplay;
2133     private javax.swing.JPanel modelPanel;
```

```
2134     private com.toedter.calendar.JDateChooser
octopusDatePicker;
2135     private javax.swing.JSlider
panelAzimuthSlider;
2136     private javax.swing.JPanel panelDetailsPanel;
2137     private javax.swing.JSlider panelTiltSlider;
2138     private javax.swing.JPanel predictedPVPanel;
2139     private javax.swing.JPanel pvGeneration;
2140     private javax.swing.JPanel pvforecast;
2141     private javax.swing.JPanel
recommendedsettingPanel;
2142     private javax.swing.JButton resetForm;
2143     private javax.swing.JButton saveChanges;
2144     private javax.swing.JButton saveChangesButton;
2145     private javax.swing.JPanel settingsPanel;
2146     private javax.swing.JToggleButton smartCharge;
2147     private javax.swing.JRadioButton solarRadio;
2148     private javax.swing.JPanel
systemOverviewPanel;
2149     private javax.swing.JPanel todayRec;
2150     private javax.swing.JButton updateModel;
2151     private javax.swing.JPanel zenithPanel;
2152     // End of variables declaration
2153 }
2154
```

Graph.java

```
  1 package solarproject;
  2
  3 import model.PVModel;
  4 import java.awt.Color;
  5 import java.text.DateFormat;
  6 import java.text.ParseException;
  7 import java.text.SimpleDateFormat;
  8 import java.time.LocalDateTime;
  9 import java.time.format.DateTimeFormatter;
 10 import java.time.temporal.ChronoUnit;
 11 import java.util.ArrayList;
 12 import java.util.Arrays;
 13 import java.util.Date;
 14 import javax.swing.JPanel;
 15 import org.knowm.xchart.CategoryChart;
 16 import org.knowm.xchart.CategoryChartBuilder;
 17 import org.knowm.xchart.XChartPanel;
 18 import org.knowm.xchart.XYChart;
```

```
 19 import org.knowm.xchart.XYChartBuilder;
 20 import org.knowm.xchart.XYSeries;
 21 import
org.knowm.xchart.XYSeries.XYSeriesRenderStyle;
 22 import
org.knowm.xchart.style.Styler.LegendPosition;
 23 import
org.knowm.xchart.style.markers.SeriesMarkers;
 24
 25 public class Graph {
 26
 27     public JPanel drawScatter(String xTitle, String
yTitle) { //Produces a scatter graph from set of data
and also adds a curve of best fit using the complex
mathematical model
 28         final XYChart chart = new
XYChartBuilder().width(600).height(400).title("Scatter
Plot with regresssion
curve").xAxisTitle(xTitle).yAxisTitle(yTitle).build();
 29
chart.getStyler().setLegendPosition(LegendPosition.Outs
ideS);
 30         chart.getStyler().setZoomEnabled(true);
 31
chart.getStyler().setDefaultSeriesRenderStyle(XYSeriesR
enderStyle.Scatter);
 32         PVModel pvModel = new PVModel();
 33         double[] xdata = new double[1100];
 34         double[] ydata;
 35         for (int i = 0; i < 1100; i++) {//generates
an array of x datapoints from 0 to 1099
 36             xdata[i] = i;
 37         }
 38         ydata = pvModel.useModel(xdata);//creates
set of data of values of the function up to x=1100
 39         chart.addSeries("Data set",
pvModel.getGHIData(), pvModel.getPPVData());//all
datapoints from model added to chart
 40         XYSeries line = chart.addSeries("Curve
Fit", xdata, ydata);//x and y data points transformed
in model function added to chart
 41
line.setXYSeriesRenderStyle(XYSeriesRenderStyle.Line);
 42         JPanel chartPanel = new
XChartPanel<XYChart>(chart);
```

```
 43          chartPanel.setVisible(true);
 44          chartPanel.setSize(600, 400);
 45          return chartPanel;
 46      }
 47
 48      public JPanel drawLineGraph(String title,
String xTitle, String yTitle, double[] yData) {
//Produces a line graph plot of a set of data. This
method is overloaded to take into account if multiple
datasets were represented on the same axis or if the
area under the graph was shaded in.
 49          final XYChart chart = new
XYChartBuilder().width(1429).height(722).title(title).x
AxisTitle(xTitle).yAxisTitle(yTitle).build();
 50
chart.getStyler().setLegendPosition(LegendPosition.Outs
ideS);
 51          chart.getStyler().setZoomEnabled(true);
 52          chart.getStyler().setCursorEnabled(true);
 53
chart.getStyler().setDefaultSeriesRenderStyle(XYSeriesR
enderStyle.Line);
 54          Number[] numsArr = toNums(yData);
 55          XYSeries series = chart.addSeries("Data
Set", Arrays.asList(createTimeArr(96)), new
ArrayList<Number>(Arrays.asList(numsArr)));//adds one
set of data with time on the x-axis as a Date object
 56          series.setMarker(SeriesMarkers.NONE);
 57          JPanel chartPanel = new
XChartPanel<XYChart>(chart);
 58          chartPanel.setVisible(true);
 59          chartPanel.setSize(1429, 722);
 60          return chartPanel;
 61      }
 62
 63      public JPanel drawLineGraphGHI(String title,
String Xtitle, String Ytitle, double[] Ydata, double[]
Ydata10, double[] Ydata90) { //Draws a line graph with
shaded high and low scenarios from 3 sets of data in
the parameters.
 64          final XYChart chart = new
XYChartBuilder().width(1429).height(722).title(title).x
AxisTitle(Xtitle).yAxisTitle(Ytitle).build();
```

```
 65
chart.getStyler().setLegendPosition(LegendPosition.OutsideS);
 66          chart.getStyler().setZoomEnabled(false);
 67
chart.getStyler().setDefaultSeriesRenderStyle(XYSeriesRenderStyle.Line);
 68
chart.getStyler().setPlotGridVerticalLinesVisible(false);
 69
chart.getStyler().setPlotGridHorizontalLinesVisible(false);
 70          chart.getStyler().setCursorEnabled(true);
 71          Number[] numsArr = toNums(Ydata);
 72          Number[] numsArr10 = toNums(Ydata10);
 73          Number[] numsArr90 = toNums(Ydata90);
 74          XYSeries series90 = chart.addSeries("High Scenario", Arrays.asList(createTimeArr(96)), new ArrayList<Number>(Arrays.asList(numsArr90)));//adds high scenario dataset to graph
 75          XYSeries series10 = chart.addSeries("Low Scenario", Arrays.asList(createTimeArr(96)), new ArrayList<Number>(Arrays.asList(numsArr10)));//adds low scenario dataset to graph
 76          XYSeries series = chart.addSeries("Data Set", Arrays.asList(createTimeArr(96)), new ArrayList<Number>(Arrays.asList(numsArr)));//adds mean most likely scenario dataset to graph
 77          series.setMarker(SeriesMarkers.NONE);//code below adds colours and shading to each plot to acheive desired effect
 78          series.setLineColor(Color.red);
 79          series90.setMarker(SeriesMarkers.NONE);
 80
series90.setXYSeriesRenderStyle(XYSeriesRenderStyle.Area);
 81          series90.setFillColor(new Color(211, 211, 211, 170));
 82          series10.setMarker(SeriesMarkers.NONE);
 83
series10.setXYSeriesRenderStyle(XYSeriesRenderStyle.Area);
 84          series10.setFillColor(new Color(255, 255, 255));
```

```
 85          series10.setLineColor(new Color(211, 211,
211, 170));
 86          series90.setLineColor(new Color(211, 211,
211, 170));
 87          JPanel chartPanel = new
XChartPanel<XYChart>(chart);
 88          chartPanel.setVisible(true);
 89          chartPanel.setSize(1429, 722);
 90          return chartPanel;
 91      }
 92
 93      public JPanel drawLineGraph(String title,
String Xtitle, String Ytitle, double[] Xdata,
double[]... Ydata) { //Produces a line graph plot of a
set of data. This method is overloaded to take into
account if multiple datasets were represented on the
same axis or if the area under the graph was shaded in.
 94          final XYChart chart = new
XYChartBuilder().width(1429).height(722).title(title).x
AxisTitle(Xtitle).yAxisTitle(Ytitle).build();
 95
chart.getStyler().setLegendPosition(LegendPosition.Outs
ideS);
 96          chart.getStyler().setZoomEnabled(true);
 97
chart.getStyler().setDefaultSeriesRenderStyle(XYSeriesR
enderStyle.Line);
 98          for (int i = 0; i < Ydata.length; i++) { //
method uses varargs to add any amount of datasets to a
graph this loop iterates through vararg parameter to
add all the plots
 99              XYSeries series =
chart.addSeries(Integer.toString(i), Xdata, Ydata[i]);
100              series.setMarker(SeriesMarkers.NONE);
101          }
102          JPanel chartPanel = new
XChartPanel<XYChart>(chart);
103          chartPanel.setVisible(true);
104          chartPanel.setSize(1429, 722);
105          return chartPanel;
106      }
107
108      public JPanel drawLineGraph(String title,
String Xtitle, String Ytitle, String[] dataTitles,
String[] Xdata, double[]... Ydata) { //Produces a line
```

```
graph plot of a set of data. This method is overloaded
to take into account if multiple datasets were
represented on the same axis or if the area under the
graph was shaded in.
109         final XYChart chart = new
XYChartBuilder().width(1429).height(722).title(title).x
AxisTitle(Xtitle).yAxisTitle(Ytitle).build();
110
chart.getStyler().setLegendPosition(LegendPosition.Outs
ideS);
111
chart.getStyler().setDefaultSeriesRenderStyle(XYSeriesR
enderStyle.Area);
112         chart.getStyler().setCursorEnabled(true);
113         for (int i = 0; i < Ydata.length; i++) {
114             if (Ydata[i] == null) {
115                 continue;
116             }
117             Number[] numsArr = toNums(Ydata[i]);
118             XYSeries series =
chart.addSeries(dataTitles[i],
Arrays.asList(toDateArr(Xdata)), new
ArrayList<Number>(Arrays.asList(numsArr)));
119             series.setMarker(SeriesMarkers.NONE);
120         }
121         JPanel chartPanel = new
XChartPanel<XYChart>(chart);
122         chartPanel.setVisible(true);
123         chartPanel.setSize(1429, 722);
124         return chartPanel;
125     }
126
127     public JPanel drawBarChart(String title, String
xTitle, String yTitle, String[][] data) { //Produces a
bar chart from inputted data
128         String[][] split = splitArray(data);
129         CategoryChart chart = new
CategoryChartBuilder().width(800).height(600).title(tit
le).xAxisTitle(xTitle).yAxisTitle(yTitle).build();
130
chart.getStyler().setLegendPosition(LegendPosition.Insi
deNW);
131         chart.getStyler().setHasAnnotations(true);
132
chart.getStyler().setAnnotationsRotation(90);
```

```
133
chart.getStyler().setPlotBackgroundColor(new Color(23,
0, 72)); //adds desired colours
134         chart.getStyler().setSeriesColors(new
Color[]{new Color(254, 71, 215)}); //adds desired
colours
135
chart.getStyler().setXAxisLabelRotation(85);
136         chart.getStyler().setLegendVisible(false);
137
chart.getStyler().setPlotTicksMarksVisible(false);
138         double[] doubleArr =
Arrays.stream(reverseArr(split[0])).mapToDouble(Double:
:parseDouble).toArray();
139         Number[] numsArr = toNums(doubleArr);
140         chart.addSeries("chart",
Arrays.asList(reverseArr(split[1])), new
ArrayList<Number>(Arrays.asList(numsArr)));
141         JPanel chartPanel = new
XChartPanel<CategoryChart>(chart);
142         chartPanel.setVisible(true);
143         chartPanel.setSize(1429, 722);
144         return chartPanel;
145     }
146
147     private String[][] splitArray(String[][] array)
{ //Splits a 2d array with many arrays that hold 2
values into a 2d array with 2 sub arrays where the
first contains all the values that were first in each
pair and the second contains all the values that were
second in the original pairs
148         int length = array.length;
149         String[] arr1 = new String[length];
150         String[] arr2 = new String[length];
151         for (int i = 0; i < length; i++) {
152             arr1[i] = array[i][0];
153             arr2[i] = array[i][1].substring(11,
19);
154         }
155         String[][] split = new String[2][length];
156         split[0] = arr1;
157         split[1] = arr2;
158         return split;
159     }
160
```

```
161     private Number[] toNums(double[] d) {
//converts to Number array from double
162         int length = d.length;
163         Number[] nums = new Number[length];
164         for (int i = 0; i < length; i++) {
165             nums[i] = d[i];
166         }
167         return nums;
168     }
169 
170     private String[] reverseArr(String[] arr) {
//Reverses the order of an array
171         int length = arr.length;
172         String[] newArr = new String[length];
173         for (int i = 0; i < length; i++) {
174             newArr[i] = arr[length - 1 - i];
175         }
176         return newArr;
177     }
178 
179     private Date[] toDateArr(String[] arr) {
//Converts a String array of dates into a Date array in
the HH:mm:ss format
180         int length = arr.length;
181         Date[] dateArr = new Date[length];
182         for (int i = 0; i < length; i++) {
183             try {
184                 String dateStr =
arr[i].substring(11, 19);
185                 DateFormat formatter = new
SimpleDateFormat("HH:mm:ss");
186                 Date date =
formatter.parse(dateStr);
187                 dateArr[i] = date;
188             } catch (ParseException ex) {
189                 new Error("There was a problem
creating graphs.");
190             }
191         }
192         return dateArr;
193     }
194 
195     private Date[] createTimeArr(int length) {
//Creates an array of Date objects which represent
```

```
every half an hour from the point in time when the
method is run up to a limit in the parameter
196         Date[] dateArr = new Date[length];
197         LocalDateTime ldt = LocalDateTime.now();
198         ldt = ldt.plusMinutes(30);
199         ldt =
ldt.truncatedTo(ChronoUnit.HOURS).plusMinutes(30 *
(ldt.getMinute() / 30));
200         DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd/MM HH:mm");
201         DateFormat formatterDate = new
SimpleDateFormat("dd/MM HH:mm");
202         for (int i = 0; i < length; i++) {
203             try {
204                 String formattedDate =
ldt.format(formatter);
205                 Date date =
formatterDate.parse(formattedDate);
206                 dateArr[i] = date;
207                 ldt = ldt.plusMinutes(30);
208             } catch (ParseException ex) {
209                 new Error("There was a problem
creating graphs.");
210                 return null;
211             }
212         }
213         return dateArr;
214     }
215 }
216
```

Files.java

```
 1 package solarproject;
 2
 3 import java.io.File;
 4 import java.io.FileNotFoundException;
 5 import java.io.FileWriter;
 6 import java.io.IOException;
 7 import java.io.PrintWriter;
 8 import java.net.URISyntaxException;
 9 import java.nio.file.Paths;
10 import java.security.CodeSource;
11
12 public class Files {
13
```

```
14      public void clear() { //Clears all data from a
text file
15          PrintWriter clear;
16          try {
17              clear = new PrintWriter(getPath() +
"\\appSettings.txt");
18              clear.print("");
19              clear.close();
20          } catch (FileNotFoundException ex) {
21              new Error("There was a problem reading
data.");
22          }
23      }
24
25      public void write(String data) { //Overwrites
all data in file with data in parameter of method
26          try {
27              FileWriter writer = new
FileWriter(getPath() + "\\appSettings.txt", true);
28              writer.write(data);
29              writer.close();
30          } catch (IOException e) {
31              new Error("There was a problem reading
data.");
32          }
33      }
34
35      public String read() { //Reads and returns all
data from text file
36          try {
37              String lineContent =
java.nio.file.Files.readAllLines(Paths.get(getPath() +
"\\appSettings.txt")).get(0);
38              return lineContent;
39          } catch (IOException ex) {
40              new Error("There was a problem reading
data.");
41          }
42          return null;
43      }
44
45      public String getPath() { //Finds the path that
the running jar file is stored in so it can find the
directory of the text file
46          try {
```

```
47              CodeSource codeSource =
MainFrame.class.getProtectionDomain().getCodeSource();
48              File jarFile = new
File(codeSource.getLocation().toURI().getPath());
49              String jarDir = jarFile.getParent();
50              return jarDir;
51          } catch (URISyntaxException ex) {
52              new Error("There was a problem reading
data.");
53              return null;
54          }
55      }
56  }
57
```

Error.java

```
 1 package solarproject;
 2
 3 import java.awt.Toolkit;
 4 import javax.swing.JFrame;
 5 import javax.swing.JOptionPane;
 6
 7 public class Error {
 8
 9     private final JFrame FRAME; //A constant that
holds a JFrame object that is used to display the error
message
10
11     private final Runnable ERROR_SOUND = (Runnable)
Toolkit.getDefaultToolkit().getDesktopProperty("win.sou
nd.exclamation"); //A Runnable object that will play a
windows error sound when the program has to display the
error window
12
13     Error(String error) { //Creates a JOptionPane
that can display to the user that there has been an
error. The parameter holds a string with a message to
display on the error window.
14         ERROR_SOUND.run();
15         FRAME = new JFrame();
16         JOptionPane.showMessageDialog(FRAME, error,
"Program Error", JOptionPane.WARNING_MESSAGE);
17     }
18 }
19
```

Dictionary.java

```
 1 package solarproject;
 2
 3 import java.util.HashMap;
 4
 5 public class Dictionary {
 6
 7     private final HashMap<String, double[]>
DICTIONARY; //This object of HashMap holds data and is
abstracted by the Dictionary class
 8
 9     public Dictionary() {
10         this.DICTIONARY = new HashMap<String,
double[]>();
11     }
12
13     public void add(String key, double value[]) {
//Adds a key and value to a hashmap
14         DICTIONARY.put(key, value);
15     }
16
17     public double[] get(String key) { //Returns data
from a key from the hashmap
18         return DICTIONARY.get(key);
19     }
20
21     public void remove(String key) { //Removes data
using a key
22         DICTIONARY.remove(key);
23     }
24
25     public void clear() { //Clears the hashmap
26         DICTIONARY.clear();
27     }
28 }
29
```

DateStrings

```
 1 package solarproject;
 2
 3 /**
 4  *
 5  * @author james
 6  */
 7 public class DateStrings {
```

```
 8
 9      private String year; //holds a string
representation of a year
10
11      private String month; //holds a string
representation of a month
12
13      private String date; //holds a string
representation of a date
14
15      public void setYear(String year) {
16          this.year = year;
17      }
18
19      public void setMonth(String month) {
20          this.month = month;
21      }
22
23      public void setDate(String date) {
24          this.date = date;
25      }
26
27      public String getYear() {
28          return this.year;
29      }
30
31      public String getMonth() {
32          return this.month;
33      }
34
35      public String getDate() {
36          return this.date;
37      }
38
39  }
40
```

ForecastAPI.java

```
  1 package other_apis;
  2
  3 import java.io.BufferedReader;
  4 import java.io.IOException;
  5 import java.io.InputStreamReader;
  6 import java.math.BigDecimal;
  7 import java.math.RoundingMode;
```

```
 8 import java.net.HttpURLConnection;
 9 import java.net.MalformedURLException;
10 import java.net.URL;
11 import java.time.LocalDate;
12 import java.time.LocalTime;
13 import java.time.format.DateTimeFormatter;
14 import java.util.ArrayList;
15 import org.json.JSONException;
16 import org.json.JSONObject;
17 import model.PVModel;
18 import model.PanelProperties;
19
20 public class ForecastAPI {
21
22     private HttpURLConnection connection; //Holds
the HttpURLConnection that is used when using the API
23
24     private String GETResponse(boolean forecast) {
//Uses the Solcast API to get a weather forecast for
the next 48 hours. Returns data as a string
representation of a JSON response
25         try {
26             String response;
27             PanelProperties pp = new
PanelProperties();
28             double longitude = pp.getLongitude();
29             double latitude = pp.getLatitude();
30             BufferedReader reader;
31             String line;
32             StringBuilder responseContent = new
StringBuilder();
33             URL url = null;
34             if (forecast) { //if forecast data is
wanted use this endpoint
35                 try {
36                     url = new
URL("https://api.solcast.com.au/world_radiation/forecas
ts?latitude=" + latitude + "&longitude=" + longitude +
"&format=json&%E2%80%8B&api_key=<insert API key>");
37                 } catch (MalformedURLException ex)
{
38                     new Error("There was a problem
collecting forecasts");
39                 }
```

```
 40              } else { // if estimated actuals are
wanted use this endpoint
 41                  try {
 42                      url = new
URL("https://api.solcast.com.au/world_radiation/estimat
ed_actuals?latitude=" + latitude + "&longitude=" +
longitude + "&format=json&%E2%80%8B&api_key=<insert API
key>");
 43                  } catch (MalformedURLException ex)
{
 44                      new Error("There was a problem
collecting forecasts");
 45                  }
 46              }
 47              connection = (HttpURLConnection)
url.openConnection();
 48              connection.setRequestMethod("GET");
 49              connection.setConnectTimeout(5000);
 50              connection.setReadTimeout(5000);
 51              int status =
connection.getResponseCode();
 52              if (status != 200) {
 53                  reader = new BufferedReader(new
InputStreamReader(connection.getErrorStream()));
 54                  while ((line = reader.readLine())
!= null) {
 55                      responseContent.append(line);
 56                  }
 57                  reader.close();
 58              } else {
 59                  reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
 60                  while ((line = reader.readLine())
!= null) {
 61                      responseContent.append(line);
 62                  }
 63                  reader.close();
 64              }
 65              response = responseContent.toString();
 66              connection.disconnect();
 67              return response;
 68          } catch (IOException ex) {
 69              new Error("There was a problem
collecting forecasts");
 70              return null;
```

```
 71          }
 72      }
 73
 74      public String forecast() { //if a forecast is
needed then this is called
 75          return GETResponse(true);
 76      }
 77
 78      public String estimatedActuals() { //if
estimated actual values are needed then this is run
 79          return GETResponse(false);
 80      }
 81
 82      public double[] GETGHI(String forecast, int
limit) {
 83          try {
 84              double[] ghiData = new double[limit];
 85              JSONObject response = new
JSONObject(forecast);
 86              for (int i = 0; i < limit; i++) {
 87                  ghiData[i] =
response.getJSONArray("forecasts").getJSONObject(i).get
Double("ghi"); //collects GHI data from json
 88              }
 89              return ghiData;
 90          } catch (JSONException ex) {
 91              new Error("There was a problem
collecting forecasts");
 92              return null;
 93          }
 94      }
 95
 96      public double[] GETGHI90(String forecast) {
 97          try {
 98              double[] ghi90Data = new double[96];
 99              JSONObject response = new
JSONObject(forecast);
100              for (int i = 0; i < 96; i++) {
101                  ghi90Data[i] =
response.getJSONArray("forecasts").getJSONObject(i).get
Double("ghi90"); //collects GHI90 data from json
102              }
103              return ghi90Data;
104          } catch (JSONException ex) {
```

```
105             new Error("There was a problem
collecting forecasts");
106             return null;
107         }
108     }
109
110     public double[] GETGHI10(String forecast) {
111         try {
112             double[] ghi10Data = new double[96];
113             JSONObject response = new
JSONObject(forecast);
114             for (int i = 0; i < 96; i++) {
115                 ghi10Data[i] =
response.getJSONArray("forecasts").getJSONObject(i).get
Double("ghi10"); //collects GHI10 data from json
116             }
117             return ghi10Data;
118         } catch (JSONException ex) {
119             new Error("There was a problem
collecting forecasts");
120             return null;
121         }
122     }
123
124     public String[][] calcTiltedPlaneRadiation(int
limit, double panelAzimuth, double panelTilt, String
radiationData, boolean forecast) {
125         /*
126         Performs calculations on the solar
radiation GHI data from the forecast to calculate the
actual intensity of light that is incident to the solar
panels (the API only returns the solar radiation
intensity for horizontal ground).
127         It used values of panel tilt angle and
azimuth and location coordinates to perform these
calculations.
128          */
129         try {
130             String[][] solarRadiation = new
String[limit][3];
131             JSONObject response = new
JSONObject(radiationData);//JSONObject that holds API
response
132             int dayOfYear;
133             double hour;
```

```
134             String arrayName, date;
135             if (forecast) {
136                 arrayName = "forecasts";
137             } else {
138                 arrayName = "estimated_actuals";
139             }
140             for (int i = 0; i < limit; i++) {
141                 try {
142                     date =
response.getJSONArray(arrayName).getJSONObject(i).getSt
ring("period_end");//gets time and date from json
response (needed for calculations)
143                     dayOfYear =
LocalDate.of(Integer.parseInt(date.substring(0, 4)),
Integer.parseInt(date.substring(5, 7)),
Integer.parseInt(date.substring(8,
10))).getDayOfYear(); // uses above date to work out
day of year
144                     hour =
Double.parseDouble(date.substring(11, 13)) +
(Double.parseDouble(date.substring(14, 16)) / 60); //
uses above date to work hour of day with a decimal to
represent minutes
145                     double elevation =
calcElevation(dayOfYear, hour); //uses clacElevation
method to work out the elevation of the sun at each
time
146                     double azimuth =
calcAzimuth(dayOfYear, hour); //uses clacAzimuth method
to work out the azimuth of the sun at each time
147                     double ghi =
response.getJSONArray(arrayName).getJSONObject(i).getDo
uble("ghi"); //gets GHI value fron API response
148                     double incidentRad = ghi /
Math.sin(toRadians(90 - elevation)); //calcuates the
radiation intensity in the direction of the direct path
from the sun to a point on the ground
149                     double moduleRad = incidentRad
* ((Math.cos(toRadians(90 - elevation)) *
Math.sin(toRadians(panelTilt)) *
Math.cos(toRadians(panelAzimuth - azimuth))) +
(Math.sin(toRadians(90 - elevation)) *
Math.cos(toRadians(panelTilt)))); //calculates the
radiation intensity that is incident to the plane of
the solar panels
```

```
150                     if ((moduleRad ==
Double.NEGATIVE_INFINITY) | (moduleRad ==
Double.POSITIVE_INFINITY) | (Double.isNaN(moduleRad)))
{ //if the calculated value tends to a very large value
due to the complex maths then the module radiation is
set to 0 as that is what it would actually be
151                         moduleRad = 0.0;
152                     } else if (moduleRad < 0) {
153                         moduleRad = moduleRad * -1;
154                     }
155                     BigDecimal bd = new
BigDecimal(moduleRad);
156                     moduleRad =
Double.parseDouble(bd.setScale(1,
RoundingMode.HALF_UP).toString());
157                     solarRadiation[i][0] =
Double.toString(moduleRad);
158                     String time =
response.getJSONArray(arrayName).getJSONObject(i).getSt
ring("period_end").substring(11, 19);
159                     DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("HH:mm:ss");//converts to
correct time format
160                     LocalTime lTime =
LocalTime.parse(time, dtf).plusSeconds(1);
161                     LocalTime lTime2 =
LocalTime.parse(lTime.toString(),
dtf).minusMinutes(30);
162                     solarRadiation[i][1] =
lTime.toString();
163                     solarRadiation[i][2] =
lTime2.toString();
164                 } catch (JSONException ex) {
165                     new Error("There was a problem
collecting forecasts");
166                     return null;
167                 }
168             }
169             return solarRadiation;
170         } catch (JSONException ex) {
171             new Error("There was a problem
collecting forecasts");
172             return null;
173         }
174     }
```

```
175
176     private double[] calcSolarParams(int days,
double hour) { //Calculates the current hour angle and
declination based upon the current date and time and
location
177         PanelProperties pp = new PanelProperties();
178         double longitude = pp.getLongitude();
179         double x = toRadians((360 / 365) * (days -
81));
180         double equationOfTime = 9.87 * (Math.sin(2
* x)) - 7.53 * (Math.cos(x)) - 1.5 * (Math.sin(x));
//calculates EoT
181         double timeCorrection = 4 * (longitude) +
equationOfTime; //calculates timeCorrection from EoT
182         double localSolarTime = hour +
(timeCorrection / 60); //calculates the local solar
time
183         double hourAngle = 15 * (localSolarTime -
12);  //calculates the hour angle
184         double declination = toRadians(3.45 *
Math.sin(x));//calculates the declination of the sun
185         return new double[]{declination,
hourAngle}; //returns the hour angle and declination
186         //maths sourced from:
https://www.pveducation.org/pvcdrom/properties-of-
sunlight/the-suns-position
187     }
188
189     private double calcElevation(int days, double
hour) { //Uses the calcSolarParams method and
trigonometry to calculate the elevation of the sun
above the horizon at a location and time
190         PanelProperties pp = new PanelProperties();
191         double[] solarParams =
calcSolarParams(days, hour);
192         double declination = solarParams[0];
193         double hourAngle = solarParams[1];
194         double latitude = pp.getLatitude();
195         double elevation =
toDegrees(Math.asin(Math.sin(declination) *
Math.sin(toRadians(latitude)) + Math.cos(declination) *
Math.cos(toRadians(latitude)) *
Math.cos(toRadians(hourAngle)))); //calculates angle of
sun in degrees above horizon
196         return elevation;
```

```
197     }
198
199     private double calcAzimuth(int days, double
hour) { //Uses the calcSolarParams method and
trigonometry to calculate the azimuth of the sun at a
location and time
200         PanelProperties pp = new PanelProperties();
201         double[] solarParams =
calcSolarParams(days, hour);
202         double declination = solarParams[0];
203         double hourAngle = solarParams[1];
204         double latitude = pp.getLatitude();
205         double azimuth =
toDegrees(Math.acos(((Math.sin(declination) *
Math.sin(toRadians(latitude))) - (Math.cos(declination)
* Math.sin(toRadians(latitude)) *
Math.cos(toRadians(hourAngle)))) /
(Math.cos(toRadians(calcElevation(days, hour))))));
//calculates azimuth of sun in degrees
206         return azimuth;
207     }
208
209     public double[] GETPPV(String forecast) {
//Uses the forecast response and mathematical model to
return an array of predicted power output from the
solar panels
210         PanelProperties pp = new PanelProperties();
211         PVModel pvm = new PVModel();
212         String[][] tiltedRadiation =
calcTiltedPlaneRadiation(96, pp.getPanelAzimuth(),
pp.getPanelTilt(), forecast, true);
213         double[] ppv = new double[96];
214         double[] radiation = new double[96];
215         for (int i = 0; i < ppv.length; i++) {
216             radiation[i] =
Double.parseDouble(tiltedRadiation[i][0]);
217         }
218         ppv = pvm.useModel(radiation);
219         return ppv;
220     }
221
222     public String[][]
PPVForDayRecommendations(String forecast) { //Uses the
mathematical model to return an array of predicted
power output and time
```

```
223         PVModel pvm = new PVModel();
224         PanelProperties pp = new PanelProperties();
225         String[][] tiltedRadiation =
calcTiltedPlaneRadiation(96, pp.getPanelAzimuth(),
pp.getPanelTilt(), forecast, true);//array of
forecasted radiation intensity on the plane of the
panels
226         double[] ppv = new double[96];
227         double[] radiation = new double[96];
228         for (int i = 0; i < ppv.length; i++) {
//converts string array to double array
229             radiation[i] =
Double.parseDouble(tiltedRadiation[i][0]);
230         }
231         ppv = pvm.useModel(radiation); //converts
solar radiation array to predicted power output array
using model
232         ArrayList<String[]> values = new
ArrayList();
233         boolean day = false;
234         for (int i = 0; i < ppv.length; i++)
{//loops that breaks full day of data into 3 parts
according to time boundaries at 8:00 and 15:30
235             if
((((!"08:00".equals(tiltedRadiation[i][2].substring(0,
5))) & day == false)) {
236             } else {
237                 day = true;
238                 if
("15:30".equals(tiltedRadiation[i][2].substring(0, 5)))
{
239                     values.add(new
String[]{Double.toString(ppv[i]),
tiltedRadiation[i][1], tiltedRadiation[i][2]});
240                     break;
241                 }
242                 values.add(new
String[]{Double.toString(ppv[i]),
tiltedRadiation[i][1], tiltedRadiation[i][2]});
243             }
244         }
245         String[][] powerPeriods = new
String[values.size()][3];
246         for (int i = 0; i < powerPeriods.length;
i++) {//arraylist items added to array to be returned
```

```
247             powerPeriods[i] = values.get(i);
248         }
249         powerPeriods = reverseArr(powerPeriods);
250         return powerPeriods;
251     }
252 
253     private String[][] reverseArr(String[][] arr) {
254         int length = arr.length;
255         int sublen = arr[0].length;
256         String[][] newarr = new
String[length][sublen];
257         for (int i = 0; i < length; i++) {
258             newarr[i] = arr[length - 1 - i];
259         }
260         return newarr;
261     }
262 
263     private double toRadians(double degrees) {
//radians to degrees calculation
264         double radians = degrees * (Math.PI / 180);
265         return radians;
266     }
267 
268     private double toDegrees(double radians)
{//degrees to radians calculation
269         double degrees = radians / (Math.PI / 180);
270         return degrees;
271     }
272 
273     public double[] GETCloudOpacity(String
forecast) {
274         try {
275             double[] cloudData = new double[96];
276             JSONObject response = new
JSONObject(forecast);
277             for (int i = 0; i < 96; i++) {
278                 cloudData[i] =
response.getJSONArray("forecasts").getJSONObject(i).get
Double("cloud_opacity");//collects cloud opacity data
from json
279             }
280             return cloudData;
281         } catch (JSONException ex) {
282             new Error("There was a problem
collecting forecasts");
```

```
283             return null;
284         }
285     }
286
287     public double[] GETZenith(String forecast) {
288         try {
289             double[] zenithData = new double[96];
290             JSONObject response = new
JSONObject(forecast);
291             for (int i = 0; i < 96; i++) {
292                 zenithData[i] =
response.getJSONArray("forecasts").getJSONObject(i).get
Double("zenith");//collects zenith data from json
293             }
294             return zenithData;
295         } catch (JSONException ex) {
296             new Error("There was a problem
collecting forecasts");
297             return null;
298         }
299     }
300
301     public double[] GETAzimuth(String forecast) {
302         try {
303             double[] azimuthData = new double[96];
304             JSONObject response = new
JSONObject(forecast);
305             for (int i = 0; i < 96; i++) {
306                 azimuthData[i] =
response.getJSONArray("forecasts").getJSONObject(i).get
Double("azimuth");//collects azimuth data from json
307             }
308             return azimuthData;
309         } catch (JSONException ex) {
310             new Error("There was a problem
collecting forecasts");
311             return null;
312         }
313     }
314
315     public double[] GETAirTemp(String forecast) {
316         try {
317             double[] tempData = new double[96];
318             JSONObject response = new
JSONObject(forecast);
```

```
319             for (int i = 0; i < 96; i++) {
320                 tempData[i] =
response.getJSONArray("forecasts").getJSONObject(i).get
Double("air_temp");//collects air temperature data from
json
321             }
322             return tempData;
323         } catch (JSONException ex) {
324             new Error("There was a problem
collecting forecasts");
325             return null;
326         }
327     }
328 }
329
```

OctopusAPI.java

```
 1 package other_apis;
 2
 3 import java.io.BufferedReader;
 4 import java.io.IOException;
 5 import java.io.InputStreamReader;
 6 import java.net.HttpURLConnection;
 7 import java.net.ProtocolException;
 8 import java.net.URL;
 9 import java.util.Base64;
10
11 public class OctopusAPI {
12
13     private String
returnDataString(HttpURLConnection connection) {
//Establishes a connection with the endpoint and forms
the response using an input stream. This data is then
returned and it will be of JSON format and returned as
a String.
14         try {
15             BufferedReader reader;
16             String line;
17             StringBuilder responseContent = new
StringBuilder();
18             int status =
connection.getResponseCode();
19             if (status != 200) {
20                 reader = new BufferedReader(new
InputStreamReader(connection.getErrorStream()));
```

```
21                  while ((line = reader.readLine()) !=
null) {
22                      responseContent.append(line);
23                  }
24                  reader.close();
25              } else {
26                  reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
27                  while ((line = reader.readLine()) !=
null) {
28                      responseContent.append(line);
29                  }
30                  reader.close();
31              }
32              String response =
responseContent.toString();
33              connection.disconnect();
34              return response;
35          } catch (IOException ex) {
36              new Error("There was a problem reading
data from Octopus API");
37              return null;
38          }
39      }
40
41      private HttpURLConnection
GETsetup(HttpURLConnection connection) { //Sets up a
GET API request and returns the HttpURLConnection
object
42          try {
43              connection.setRequestMethod("GET");
44              connection.setConnectTimeout(5000);
45              connection.setReadTimeout(10000);
46              return connection;
47          } catch (ProtocolException ex) {
48              new Error("There was a problem reading
data from Octopus API");
49              return null;
50          }
51      }
52
53      public String GETAgilePrices(String periodStart,
String periodEnd) { //Uses the Octopus API to get the
agile prices for a given time period and returns a
string representation of a JSON response
```

```
54          try {
55              HttpURLConnection connection;
56              URL url = new
URL("https://api.octopus.energy/v1/products/AGILE-18-
02-21/electricity-tariffs/E-1R-AGILE-18-02-21-
H/standard-unit-rates/?period_from=" + periodStart +
"&period_to=" + periodEnd);
57              connection = (HttpURLConnection)
url.openConnection();
58              connection = GETsetup(connection);
59              String response =
returnDataString(connection);
60              return response;
61          } catch (IOException ex) {
62              new Error("There was a problem reading
data from Octopus API");
63              return null;
64          }
65      }
66
67      public String GETAccounts() {//gets the user
account information
68          try {
69              HttpURLConnection connection;
70              String encoding =
Base64.getEncoder().encodeToString("<insert API
key>".getBytes("UTF-8"));
71              URL url = new
URL("https://api.octopus.energy/v1/accounts/<insert
account number>/");
72              connection = (HttpURLConnection)
url.openConnection();
73
connection.setRequestProperty("Authorization", "Basic "
+ encoding);
74              connection = GETsetup(connection);
75              String response =
returnDataString(connection);
76              return response;
77          } catch (IOException ex) {
78              new Error("There was a problem reading
data from Octopus API");
79          }
80          return null;
81      }
```

```
82 }
83
```

AgilePricesOctopusAPI.java

```
 1 package other_apis;
 2
 3 import optimisation.Prices;
 4 import org.json.JSONException;
 5 import org.json.JSONObject;
 6
 7 public class AgilePricesOctopusAPI extends
OctopusAPI {
 8
 9     private JSONObject GETJSON(String data) {
//turns a string in JSON format into a JSONObject
10         try {
11             JSONObject response = new
JSONObject(data);
12             return response;
13         } catch (JSONException ex) {
14             new Error("There was a problem
collecting Agile Prices");
15             return null;
16         }
17     }
18
19     public String[][] GETPrices(String start, String
end) { //Uses the Octopus API methods from the
OctopusAPI class it inherits from to get the agile
prices for an inputted day.
20         try {
21             String data = GETAgilePrices(start,
end);
22             JSONObject jo = GETJSON(data);//turns a
String of data into JSONObject
23             int count = jo.getInt("count");
24             String[][] prices = new
String[count][2];
25             for (int i = 0; i < count; i++) { //loop
formats each item and adds to an array that will be
returned from the method
26                 prices[i][0] =
Double.toString(jo.getJSONArray("results").getJSONObjec
t(i).getDouble("value_inc_vat"));
```

```
27                  prices[i][1] =
jo.getJSONArray("results").getJSONObject(i).getString("
valid_from");
28              }
29              Prices p = new Prices(prices);
30              p.findPrices(true, 5);
31              return prices;
32          } catch (JSONException ex) {
33              new Error("There was a problem
collecting forecasts");
34              return null;
35          }
36      }
37  }
38
```

Energy.java

```
 1 package optimisation;
 2
 3 public class Energy {
 4
 5     private final String[][] POWER;//holds power
values and a time for that value in a 2d array
 6
 7     private String[] energyPeriods;//string that
holds a value of energy produced for each time period
 8
 9     public Energy(String[][] power) {
10         this.POWER = power;
11     }
12
13     private double sumArr(String[][] arr) {
14         double sum = 0;
15         for (int i = 1; i < arr.length - 1; i++) {
16             sum += Double.parseDouble(arr[i][0]);
17         }
18         return sum;
19     }
20
21     public void findEnergy() { //Calculates a
forecast for the energy that is produced by the solar
panels based upon the expected power output of the
panels.
22         energyPeriods = new String[11];
23         String[][] subArr = new String[5][3];
```

```
24          double energy;
25          for (int i = 0; i < POWER.length - 5; i++) {
26              for (int j = 0; j < 5; j++) {
27                  subArr[j] = POWER[i + j];
28              }
29              energy = trapeziumCalculus(subArr);
//calcluates energy from power values
30              energyPeriods[i] =
Double.toString(energy);
31          }
32      }
33
34      private double trapeziumCalculus(String[][]
vals) { //Uses the Trapezium Rule to perform
integration on a set of data.
35          double energy =
Double.parseDouble(vals[0][0]) +
Double.parseDouble(vals[vals.length - 1][0]);
36          energy = 0.25 * (energy + (2 *
sumArr(vals)));
37          return energy;
38      }
39
40      public String[][] findRatios(String[][] prices)
{ //Finds the ratios between electricity price and
energy generated at time periods over a day.
41          String[][] ratios = new
String[prices.length][3];
42          for (int i = 0; i < prices.length; i++) {
43              try {
44                  ratios[i][0] =
Double.toString(Double.parseDouble(prices[i][0]) /
Double.parseDouble(this.energyPeriods[i]));//ratio of
price and energy
45              } catch (ArithmeticException e) {
46                  ratios[i][0] =
Double.toString(1000000000); //incase the energy
produced is equal to 0 then ratio set to very high
value
47              }
48              ratios[i][1] = prices[i][1];
49              ratios[i][2] = prices[i][2];
50          }
51          return ratios;
52      }
```

```
53 }
54
```

Prices.java

```
 1 package optimisation;
 2
 3 public class Prices {
 4
 5     private final String[][] PRICES; //Holds the
price and time for each half hour across a day
 6
 7     public Prices(String[][] prices) {
 8         this.PRICES = prices;
 9     }
10
11     private double sumArr(double[] arr) { //sums up
the values in a double array
12         double sum = 0;
13         for (Double arr1 : arr) {
14             sum += arr1;
15         }
16         return sum;
17     }
18
19     public String[][] findPrices(boolean sort, int
interval) { //Finds the total cost of importing energy
for every 2.5 hour period over a day
20         int length = PRICES.length - interval;
21         double[] periodVals = new double[interval];
22         String[] startAndEnd = new String[2];
23         String[][] periods = new String[length][3];
24         for (int i = 0; i < length; i++) {
25             boolean end = false; //has it reached
end of period
26             for (int j = 0; j < periodVals.length;
j++) {
27                 try {
28                     periodVals[j] =
Double.parseDouble(PRICES[i + j][0]);
29                 } catch (Exception e) {
30                     break;
31                 }
32             }
33             startAndEnd[0] = PRICES[i][1]; //set
start
```

```
34              try {
35                  startAndEnd[1] = PRICES[i + interval
- 1][1]; //set end
36              } catch (Exception e) {
37                  break;
38              }
39              double sum = sumArr(periodVals);
40              periods[i] = new
String[]{Double.toString(sum), startAndEnd[0],
startAndEnd[1]};
41              if (end) {
42                  break;
43              }
44          }
45          if (sort) { //if the array is needed to be
sorted from lowest to highest (needed in optimisation
process) then quicksort can be used
46              QuickSort qs = new QuickSort();
47              qs.quickSort(periods, 0, periods.length
- 1);
48          }
49          return periods;
50      }
51  }
52
```

QuickSort.java

```
 1 package optimisation;
 2
 3 public class QuickSort {
 4
 5     //algorithm from
https://www.geeksforgeeks.org/quick-sort/
 6     private int partition(String[][] list, int
start, int end) { //splits array around a value then
puts that value in sorted array and put all smaller
elements before that value and all greater elements
after that value
 7         double pivot =
Double.parseDouble(list[end][0]);
 8         int i = start - 1;
 9         for (int j = start; j <= end - 1; j++) {
10             if (Double.parseDouble(list[j][0]) <
pivot) {
11                 i++;
```

```
12                  String[] temp = list[i];
13                  list[i] = list[j];
14                  list[j] = temp;
15              }
16          }
17          String[] temp = list[i + 1];
18          list[i + 1] = list[end];
19          list[end] = temp;
20          return (i + 1);
21      }
22
23      public void quickSort(String[][] list, int
start, int end) { //A recursive method that can order
an array of items very fast
24          if (start < end) {
25              int part = partition(list, start, end);
26              quickSort(list, start, part - 1);
27              quickSort(list, part + 1, end);
28          }
29      }
30  }
31
```

PolynomialRegression.java

```
 1 package model;
 2
 3 public class PolynomialRegression {
 4
 5     private final double[] X_DATA; //Holds double
values for the x data in a data set
 6
 7     private final double[] Y_DATA; //Holds double
values for the y data in a data set
 8
 9     private double[] coeff; //Holds a set of values
that are the coefficients to the polynomial function
10
11     PolynomialRegression(double[] xData, double[]
yData, int degree, boolean init) {
12         this.X_DATA = xData;
13         this.Y_DATA = yData;
14         if (init) {
15             coeffCalc(degree);//object can be
initialised with coefficients so model doesnt need to
be recalculated every time to improve efficiency
```

```
16          }
17      }
18
19      public double[] getCoeff() {
20          return coeff;
21      }
22
23      public void setCoeff(double[] coeff) {
24          this.coeff = coeff;
25      }
26
27      public double curveFunction(double x) {
//Returns a value based on an input value using the
polynomial function stored in that object
28          double value = 0;
29          for (int i = 0; i < coeff.length; i++) {
30              value += coeff[i] * (Math.pow(x, i));
//adds a power to each domain in incrementing values
for polynomial funciton
31          }
32          return value;
33      }
34
35      private void coeffCalc(int degree) { //Uses
matrices and arrays of x and y data to perform least
squares regression to produces a polynomial function
that represents a best fit for a set of data.
36          int m = degree + 1;
37          coeff = new double[m];
38          Matrix xMatrix = new Matrix(m, m);
39          for (int i = 0; i < m; i++) {
40              for (int j = 0; j < m; j++) {
41                  xMatrix.add(sum(X_DATA, i + j), i,
j);//sum of Xs matrix
42              }
43          }
44          Matrix xyMatrix = new Matrix(m, 1);
45          double[] XYdata;
46          for (int i = 0; i < m; i++) {
47              XYdata = arrayProduct(i);
48              xyMatrix.add(sum(XYdata, 1), i, 0);//sum
of XYs matrix
49          }
50          Matrix inverse =
xMatrix.inverse(xMatrix);//inverses matrix
```

```
51          Matrix coeffMat =
inverse.leftMultiply(xyMatrix); //multiplys xMatrix and
xyMatrix to get vector matrix with coefficient values
in
52          for (int i = 0; i < m; i++) {
53              coeff[i] = coeffMat.getMatrix()[i][0];
54          }
55      }
56
57      private double sum(double[] data, int power) {
//Sums data in an array where each data item is raised
to the power in the parameter
58          double sum = 0;
59          for (int i = 0; i < data.length; i++) {
60              sum += Math.pow(data[i], power);
61          }
62          return sum;
63      }
64
65      private double[] arrayProduct(int power) {
//Combines the x data and y data by multiplying each y
term by the corresponding x term raised to the power of
their index
66          double[] XYdata = new double[X_DATA.length];
67          for (int i = 0; i < XYdata.length; i++) {
68              XYdata[i] = Math.pow(X_DATA[i], power) *
Y_DATA[i];
69          }
70          return XYdata;
71      }
72 }
73
```

PVModel.java

```
  1 package model;
  2
  3 import databases.DBpvModel;
  4 import givenergy_apis.ChartsGivEnergyAPI;
  5 import other_apis.ForecastAPI;
  6 import java.math.BigDecimal;
  7 import java.math.RoundingMode;
  8 import java.text.Format;
  9 import java.text.SimpleDateFormat;
 10 import java.time.LocalDate;
 11 import java.time.LocalTime;
```

```
 12 import java.time.format.DateTimeFormatter;
 13 import java.util.Date;
 14 import org.json.JSONException;
 15 import org.json.JSONObject;
 16 import solarproject.DateStrings;
 17 import solarproject.Files;
 18
 19 public final class PVModel {
 20
 21     private final PanelProperties PANEL_PROPERITES
= new PanelProperties(); //An object that holds
variables about the orientation and location of the
solar panels
 22
 23     private final DateStrings DATE_STRS = new
DateStrings(); //An object that holds variables that
represent a year, month and date
 24
 25     public PVModel() {
 26         updateParams();
 27         LocalDate ld = LocalDate.now();
 28
DATE_STRS.setYear(Integer.toString(ld.getYear()));
 29
DATE_STRS.setMonth(Integer.toString(ld.getMonthValue())
);
 30         Format formatter = new
SimpleDateFormat("yyyy-MM-dd");
 31         String date = formatter.format(new Date());
 32         DATE_STRS.setDate(date);
 33     }
 34
 35     public PanelProperties getProperties() {
 36         return this.PANEL_PROPERITES;
 37     }
 38
 39     public void updateParams() { //After the user
changes settings in the GUI such as the location
coordinates of the solar panels or tilt of the panels,
this method can update the attributes in the class that
are holding information
 40         try {
 41             Files files = new Files();//reads from
appSettings.txt file to get panel info
 42             String settings = files.read();
```

```
 43              JSONObject jo = new
JSONObject(settings);
 44
PANEL_PROPERITES.setPanelAzimuth(jo.getInt("azimuth"));
 45
PANEL_PROPERITES.setPanelTilt(jo.getInt("tilt"));
 46
PANEL_PROPERITES.setLongitude(jo.getDouble("longitude")
);
 47
PANEL_PROPERITES.setLatitude(jo.getDouble("latitude"));
 48          } catch (JSONException ex) {
 49              new Error("There was a problem reading
data");
 50          }
 51      }
 52
 53      public double[] getGHIData() { //Returns all
the solar radiation data that is stored in the database
that is being used by the complex mathematical model
 54          DBpvModel db = new DBpvModel();
 55          db.getModelData();
 56          return db.getGHIdata();//gets all GHI
datapoints from database
 57      }
 58
 59      public double[] getPPVData() { //Returns all
the Solar Panel power output data that is stored in the
database that is being used by the complex mathematical
model
 60          DBpvModel db = new DBpvModel();
 61          db.getModelData();
 62          return db.getPVoutputdata();//gets all PPV
datapoints from database
 63      }
 64
 65      public void updateModel() { //A key function
that will update and refine the complex mathematical
model with new data that has been collected over the
last day.
 66          DBpvModel db = new DBpvModel();
 67          ChartsGivEnergyAPI cgeAPI = new
ChartsGivEnergyAPI(DATE_STRS.getYear(),
DATE_STRS.getMonth(), DATE_STRS.getDate());
 68          ForecastAPI fa = new ForecastAPI();
```

```
 69          String actuals =
fa.estimatedActuals();//array of estimated actual solar
radiation values over the last day
 70          String[][] radData =
fa.calcTiltedPlaneRadiation(48,
PANEL_PROPERITES.getPanelAzimuth(),
PANEL_PROPERITES.getPanelTilt(), actuals, false);
//recalculated array of solar raditaion values taking
panel orientation into account
 71          String[][] ppv =
cgeAPI.GETPVpowerByDay(DATE_STRS.getDate()); //array of
the actual power output of the solar panels
 72          double[] newppv = format(radData, ppv);
 73          double[] newRadData =
removeSecondItem(radData);
 74          db.addToPVModelData(newRadData,
newppv);//adds the new collected data to the data table
in the database
 75          db.getModelData();
 76          PolynomialRegression pr = new
PolynomialRegression(db.getGHIdata(),
db.getPVoutputdata(), 5, true); //re-evaluates the
complex mathematical model with all the newly collected
data
 77          double[] coeff = pr.getCoeff();
 78          db.addToPVModel(coeff[5], coeff[4],
coeff[3], coeff[2], coeff[1], coeff[0]); //adds the
newly calculated model coefficients to the database
 79      }
 80
 81      public double useModel(double ghi) { //From an
inputted value of GHI (solar radiation) in the
parameter it will return a value for the expected power
output of the solar panels based on the mathematical
model
 82          DBpvModel db = new DBpvModel();
 83          double[] model = db.getModel(); // fetches
mathematical model coefficients from DB
 84          model = reverseArr(model);
 85          PolynomialRegression nlr = new
PolynomialRegression(db.getGHIdata(),
db.getPVoutputdata(), 5, false);
 86          nlr.setCoeff(model); //sets coefficients as
model has already been calculated and fetched from
storage in DB
```

```
 87          double predictedpv =
nlr.curveFunction(ghi);
 88          return predictedpv;
 89      }
 90
 91      public double[] useModel(double[] ghi) { //From
an inputted array of values of GHI (solar radiation) in
the parameter it will return an array of values for the
expected power output of the solar panels based on the
mathematical model
 92          DBpvModel db = new DBpvModel();
 93          db.getModelData();
 94          double[] model = db.getModel(); //fetches
mathematical model coefficients from DB
 95          model = reverseArr(model);
 96          double[] ppv = new double[ghi.length];
 97          PolynomialRegression nlr = new
PolynomialRegression(db.getGHIdata(),
db.getPVoutputdata(), 5, false);
 98          nlr.setCoeff(model); //sets coefficients as
model has already been calculated and fetched from
storage in DB
 99          double predictedPV;
100          for (int i = 0; i < ppv.length; i++) {
//iterates through array of inputs to produce array of
outputs
101              predictedPV =
nlr.curveFunction(ghi[i]);
102              ppv[i] = predictedPV;
103          }
104          return ppv;
105      }
106
107      private double[] format(String[][]
ghiEstimatedValues, String[][] ppv) {
108          /*
109          Formats data that is collected from
temporary storage in database and data that is
collected from weather forecast API.
110          It calculated average power outputs over
half hour periods of the forecasted time frame
according to the times returned from the API.
111           */
112          double[] newppv = new
double[ghiEstimatedValues.length];
```

```
113         DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("HH:mm:ss");
114         for (int i = 0; i <
ghiEstimatedValues.length; i++) {
115             String periodEnd =
ghiEstimatedValues[i][1];
116             LocalTime periodEndlt =
LocalTime.parse(periodEnd, dtf); //end of the half hour
period
117             LocalTime periodStartlt =
LocalTime.parse(periodEnd, dtf).minusMinutes(30);
//start of the half hour period
118             double avrPower, totalPower = 0;
119             int count = 0;
120             for (String[] ppv1 : ppv) {
121                 String ppvTime = ppv1[1];
122                 LocalTime ppvTimelt =
LocalTime.parse(ppvTime.substring(11, 19), dtf); //the
time the power value was recorded
123                 boolean inRange =
ppvTimelt.isAfter(periodStartlt) &
ppvTimelt.isBefore(periodEndlt); //calculates whether
the LocalTime object is within two other times and sets
boolean values accordingly
124                 if (inRange) {
125                     totalPower +=
Double.parseDouble(ppv1[0]); //if in range then start
adding up values in order to be able to calculate
average later
126                     count++;
127                 }
128             }
129             avrPower = totalPower / count;
//average value in time period calcualted by dividing
sum by the count
130             if (Double.isNaN(avrPower)) {
131                 avrPower = 0;
132             }
133             BigDecimal bd = new
BigDecimal(avrPower);
134             avrPower =
Double.parseDouble(bd.setScale(1,
RoundingMode.HALF_UP).toString()); //rounds the value
to remove unnessicary decimal places
```

```
135             newppv[i] = avrPower; //adds average
power value for that period to array
136         }
137         return newppv;
138     }
139
140     private double[] removeSecondItem(String[][]
twoDarray) {
141         double[] oneDArray = new
double[twoDarray.length];
142         for (int i = 0; i < oneDArray.length; i++)
{
143             oneDArray[i] =
Double.parseDouble(twoDarray[i][0]);
144         }
145         return oneDArray;
146     }
147
148     private double[] reverseArr(double[] arr) {
149         int length = arr.length;
150         double[] newArr = new double[length];
151         for (int i = 0; i < length; i++) {
152             newArr[i] = arr[length - 1 - i];
153         }
154         return newArr;
155     }
156 }
157
```

PanelProperties

```
 1 package model;
 2
 3 public class PanelProperties {
 4
 5     private int panelTilt; //Holds an integer value
for panel tilt
 6
 7     private int panelAzimuth; //Holds an integer
value for panel azimuth
 8
 9     private double longitude; //Holds a double value
for a longitude
10
11     private double latitude; //Holds a double value
for a latitude
12
```

```
13      public void setPanelTilt(int tilt) {
14          this.panelTilt = tilt;
15      }
16
17      public void setPanelAzimuth(int azimuth) {
18          this.panelAzimuth = azimuth;
19      }
20
21      public void setLongitude(double longitude) {
22          this.longitude = longitude;
23      }
24
25      public void setLatitude(double latitude) {
26          this.latitude = latitude;
27      }
28
29      public int getPanelTilt() {
30          return this.panelTilt;
31      }
32
33      public int getPanelAzimuth() {
34          return this.panelAzimuth;
35      }
36
37      public double getLongitude() {
38          return this.longitude;
39      }
40
41      public double getLatitude() {
42          return this.latitude;
43      }
44  }
45
```

Matrix.java

```
 1 package model;
 2
 3 public class Matrix {
 4
 5     private final int ROWS; //Holds the number of
rows in the matrix
 6     private final int COLUMNS; //Holds the number of
columns in the matrix
 7
 8     private double[][] matrix; //Holds values stored
in the matrix object
```

```
 9
10      public Matrix(int rows, int columns) {
11          this.ROWS = rows;
12          this.COLUMNS = columns;
13          build();
14      }
15
16      public double[][] getMatrix() {
17          return this.matrix;
18      }
19
20      private void build() { //Creates an n*m
dimensional array that is used to fundamentally hold
the data stored in a matrix object
21          matrix = new double[ROWS][COLUMNS]; //matrix
array given dimentions of matrix
22          for (int i = 0; i < ROWS; i++) {
23              double[] row = new double[COLUMNS];
24              matrix[i] = row;
25          }
26      }
27
28      public void add(double val, int row, int column)
{ //Adds a value to a specific position in the matrix
29          matrix[row][column] = val;
30      }
31
32      public Matrix inverse(Matrix matrix) {
//Performs an inversing operation on the matrix that is
passed into the method. The method then returns the
inversed version of that matrix.
33          Matrix inverse = new Matrix(ROWS, COLUMNS);
//new matrix object created to hold inverse matrix
34          for (int i = 0; i < matrix.ROWS; i++)
{//loops through every element
35              for (int j = 0; j <
matrix.matrix[i].length; j++) {
36                  inverse.matrix[i][j] = Math.pow(-1,
i + j) * determinant(submatrix(matrix, i, j)); //uses
determinant and submatrix method to calculate value of
element in inverse array and then depending on position
makes it negative
37              }
38          }
39          double multiplier = 1 / determinant(matrix);
```

```
40          for (int i = 0; i < inverse.ROWS; i++) {
41              for (int j = 0; j <= i; j++) {
42                  double temp = inverse.matrix[i][j];
43                  inverse.matrix[i][j] =
inverse.matrix[j][i] * multiplier;//mulitplies every
element by multiplier
44                  inverse.matrix[j][i] = temp *
multiplier;
45              }
46          }
47          return inverse;
48      }
49
50      public Matrix leftMultiply(Matrix rMatrix) {
//Performs matrix multiplication and returns the result
of the operation as a matrix.
51          Matrix product = new Matrix(rMatrix.ROWS,
rMatrix.COLUMNS);
52          for (int i = 0; i < rMatrix.ROWS; i++) {
53              double sum = 0;
54              for (int j = 0; j < COLUMNS; j++) {
55                  sum += rMatrix.matrix[j][0] *
matrix[i][j];
56              }
57              product.add(sum, i, 0);
58          }
59          return product;
60      }
61
62      private double determinant(Matrix matrix) {
//Calculates the determinant of a given inputted matrix
63          if (matrix.ROWS == 2) {
64              return matrix.matrix[0][0] *
matrix.matrix[1][1] - matrix.matrix[0][1] *
matrix.matrix[1][0]; //special case if matrix is only
2x2
65          }
66          double det = 0;
67          for (int i = 0; i < matrix.ROWS; i++) {
68              det += Math.pow(-1, i) *
matrix.matrix[0][i] * determinant(submatrix(matrix, 0,
i)); //recursion as determinant of each submatrix needs
to be found, every other value multiplied by -1
69          }
70          return det;
```

```
71      }
72
73      private Matrix submatrix(Matrix matrix, int row,
int column) { //Finds the particular submatrix for a
given row and column of a matrix.
74          Matrix submatrix = new Matrix(matrix.ROWS -
1, matrix.COLUMNS - 1); //new matrix object to hold
submatrix
75          for (int i = 0; i < matrix.ROWS; i++) {
76              for (int j = 0; (i != row) && (j <
matrix.matrix[i].length); j++) {
77                  if (j != column) {
78                      submatrix.matrix[i < row ? i : i
- 1][j < column ? j : j - 1] = matrix.matrix[i][j];
79                  }
80              }
81          }
82          return submatrix;
83      }
84  }
85
```

GivEnergyAPI.java

```
  1 package givenergy_apis;
  2
  3 import java.io.BufferedReader;
  4 import java.io.IOException;
  5 import java.io.InputStreamReader;
  6 import java.io.OutputStream;
  7 import java.io.UnsupportedEncodingException;
  8 import java.net.HttpCookie;
  9 import java.net.HttpURLConnection;
 10 import java.net.ProtocolException;
 11 import java.net.URL;
 12 import java.util.List;
 13 import java.util.Map;
 14
 15 public class GivEnergyAPI {
 16
 17     private String jSessionID; //String
representation of the session cookie used by the
GivEnergyAPI used to retain the connection to the API
 18
 19     public String getSessionID() {
 20         return this.jSessionID;
```

```
 21      }
 22
 23      private String
returnDataString(HttpURLConnection connection) { //Uses
an HttpURLConnection object to get the API response
from any GivEnergy API endpoint and returns it as a
string.
 24          try {
 25              BufferedReader reader;
 26              String line;
 27              StringBuilder responseContent = new
StringBuilder();
 28              int status =
connection.getResponseCode();
 29              if (status != 200) {
 30                  reader = new BufferedReader(new
InputStreamReader(connection.getErrorStream()));
 31                  while ((line = reader.readLine())
!= null) {
 32                      responseContent.append(line);
 33                  }
 34                  reader.close();
 35              } else {
 36                  reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
 37                  while ((line = reader.readLine())
!= null) {
 38                      responseContent.append(line);
 39                  }
 40                  reader.close();
 41              }
 42              String response =
responseContent.toString();
 43              connection.disconnect();
 44              return response;
 45          } catch (IOException ex) {
 46              new Error("There was a problem with the
connection to GivEnergy.");
 47              return null;
 48          }
 49      }
 50
 51      public void login() { //Method that is called
when the program needs to connect to the GivEnergy API.
```

```
It also sets a value to the jsessionid attribute so the
same connection can be retained
 52          try {
 53              HttpURLConnection connection;
 54              URL url = new
URL("https://www.givenergy.cloud/GivManage/api/login?ac
count=<insert GivEnergy username>&password=<insert
GivEnergy password>"); //uses login endpoint to login
to the API
 55              connection = (HttpURLConnection)
url.openConnection();
 56              connection.setRequestMethod("POST");
 57              jSessionID = findCookies(connection);
//sets the cookie attribute so that the session can be
retained
 58              connection.setConnectTimeout(5000);
 59              connection.setReadTimeout(5000);
 60          } catch (IOException ex) {
 61              new Error("There was a problem with the
connection to GivEnergy.");
 62          }
 63      }
 64
 65      private String findCookies(HttpURLConnection
connection) { //Gets the cookie string from the API
connection so that the program can remain logged into
the API with cookie retention
 66          java.net.CookieManager msCookieManager =
new java.net.CookieManager();
 67          Map<String, List<String>> headerFields =
connection.getHeaderFields();
 68          List<String> cookiesHeader =
headerFields.get("Set-Cookie");
 69          if (cookiesHeader != null) {
 70              for (String cookie : cookiesHeader) {
 71
msCookieManager.getCookieStore().add(null,
HttpCookie.parse(cookie).get(0));
 72              }
 73          }
 74          String cookie =
(cookiesHeader.toString().subSequence(12,
44)).toString();
 75          return cookie;
 76      }
```

```
 77
 78      public String GETInvData(String date) {
 79          try {
 80              HttpURLConnection connection;
 81              URL url = new
URL("https://www.givenergy.cloud/GivManage/invData/<ins
ert inverter serial number>/" + "" + date + "" +
"+?page=1&rows=1000"); //uses inverter data endpoint to
get data about inverter
 82              connection = (HttpURLConnection)
url.openConnection();
 83              connection = GETsetup(connection);
 84              String response =
returnDataString(connection);
 85              return response;
 86          } catch (IOException ex) {
 87              new Error("There was a problem with the
connection to GivEnergy.");
 88              return null;
 89          }
 90      }
 91
 92      public String GETPlantSummary() {
 93          try {
 94              HttpURLConnection connection;
 95              URL url = new
URL("https://www.givenergy.cloud/GivManage/api/plant/ge
tPlantSummary?plantId=<insert plant id>"); //uses plant
summary endpoint to get data about the plant (whole
solar panel system)
 96              connection = (HttpURLConnection)
url.openConnection();
 97              connection = GETsetup(connection);
 98              String response =
returnDataString(connection);
 99              return response;
100          } catch (IOException ex) {
101              new Error("There was a problem with the
connection to GivEnergy.");
102              return null;
103          }
104      }
105
106      public String GETPlantChartByMonth(String year,
String month) {
```

```
107         try {
108             HttpURLConnection connection;
109             URL url = new
URL("https://www.givenergy.cloud/GivManage/api/plantCha
rt/monthColumn?plantId=<insert plant id>&year=" + year
+ "&month=" + month); //uses plantChart endpoint to get
data about the plant by month
110             connection = (HttpURLConnection)
url.openConnection();
111             connection = GETsetup(connection);
112             String response =
returnDataString(connection);
113             return response;
114         } catch (IOException ex) {
115             new Error("There was a problem with the
connection to GivEnergy.");
116             return null;
117         }
118     }
119
120     public String GETPlantChartByYear(String year)
{
121         try {
122             HttpURLConnection connection;
123             URL url = new
URL("https://www.givenergy.cloud/GivManage/api//plantCh
art/yearColumn?plantId=<insert plant id>&year=" +
year); //uses plantChart endpoint to get data about the
plant by year
124             connection = (HttpURLConnection)
url.openConnection();
125             connection = GETsetup(connection);
126             String response =
returnDataString(connection);
127             return response;
128         } catch (IOException ex) {
129             new Error("There was a problem with the
connection to GivEnergy.");
130             return null;
131         }
132     }
133
134     public String GETInverterChartByDay(String
date) {
135         try {
```

```
136             HttpURLConnection connection;
137             URL url = new
URL("https://www.givenergy.cloud/GivManage/api/invChart
/dayMultiLine?serialNum=<insert serial
number>&dateText=" + date); //uses inverter data
endpoint to get data about inverter by day
138             connection = (HttpURLConnection)
url.openConnection();
139             connection = GETsetup(connection);
140             String response =
returnDataString(connection);
141             return response;
142         } catch (IOException ex) {
143             new Error("There was a problem with the
connection to GivEnergy.");
144             return null;
145         }
146     }
147
148     public String GETAllBatteryData() {
149         try {
150             HttpURLConnection connection;
151             URL url = new
URL("https://api.givenergy.cloud/batteryData/all");
152             connection = (HttpURLConnection)
url.openConnection();
153             connection.setRequestMethod("GET");
154
connection.addRequestProperty("Authorization",
"lGVWeCFcRrJG7NmZwOcYCjXbO3hIWq"); //uses the battery
data endpoint to get data about battery
155             connection.setConnectTimeout(5000);
156             connection.setReadTimeout(5000);
157             String response =
returnDataString(connection);
158             return response;
159         } catch (IOException ex) {
160             new Error("There was a problem with the
connection to GivEnergy.");
161             return null;
162         }
163     }
164
165     public String POSTchargeBattey(int start, int
end, int chargeto) {
```

```
166         try {
167             HttpURLConnection connection;
168             String json = "{\r\n    \"enable\":
true,\r\n    \"start\": \"" + start + "\",\r\n
\"finish\": \"" + end + "\",\r\n
\"chargeToPercent\": \"" + chargeto + "\"\r\n}"; //json
formed to pass into POST API request
169             URL url = new
URL("https://api.givenergy.cloud/chargeBattery");
170             connection = (HttpURLConnection)
url.openConnection();
171             connection = POSTsetup(connection,
json);
172             String response =
returnDataString(connection);
173             return response;
174         } catch (IOException ex) {
175             new Error("There was a problem with the
connection to GivEnergy.");
176             return null;
177         }
178     }
179
180     public String POSTdischargeBattey(boolean
enable, int start, int end, int dischargeto) {
181         try {
182             HttpURLConnection connection;
183             String json = "{\r\n    \"" + enable +
"\": true,\r\n    \"start\": \"" + start + "\",\r\n
\"finish\": \"" + end + "\",\r\n
\"dischargeToPercent\": \"" + dischargeto + "\"\r\n}";
//json formed to pass into POST API request
184             URL url = new
URL("https://api.givenergy.cloud/dischargeBattery");
185             connection = (HttpURLConnection)
url.openConnection();
186             connection = POSTsetup(connection,
json);
187             String response =
returnDataString(connection);
188             return response;
189         } catch (IOException ex) {
190             new Error("There was a problem with the
connection to GivEnergy.");
191             return null;
```

```
192         }
193     }
194
195     public String POSTmode(int mode) {
196         try {
197             HttpURLConnection connection;
198             String json = "{\r\n    \"mode\": \"" +
mode + "\"\r\n}"; //json formed to pass into POST API
request
199             URL url = new
URL("https://api.givenergy.cloud/mode");
200             connection = (HttpURLConnection)
url.openConnection();
201             connection = POSTsetup(connection,
json);
202             String response =
returnDataString(connection);
203             return response;
204         } catch (IOException ex) {
205             new Error("There was a problem with the
connection to GivEnergy.");
206             return null;
207         }
208     }
209
210     public String POSTchargeFlag(int flag) {
211         try {
212             HttpURLConnection connection;
213             String json = "{\r\n    \"value\": \""
+ flag + "\"\r\n}"; //json formed to pass into POST API
request
214             URL url = new
URL("https://api.givenergy.cloud/registers/chargeFlag")
;
215             connection = (HttpURLConnection)
url.openConnection();
216             connection = POSTsetup(connection,
json);
217             String response =
returnDataString(connection);
218             return response;
219         } catch (IOException ex) {
220             new Error("There was a problem with the
connection to GivEnergy.");
221             return null;
```

```
222             }
223         }
224 
225         public String POSTdischargeFlag(int flag) {
226             try {
227                 HttpURLConnection connection;
228                 String json = "{\r\n    \"value\": \""
+ flag + "\"\r\n}"; //json formed to pass into POST API
request
229                 URL url = new
URL("https://api.givenergy.cloud/registers/dischargeFla
g");
230                 connection = (HttpURLConnection)
url.openConnection();
231                 connection = POSTsetup(connection,
json);
232                 String response =
returnDataString(connection);
233                 return response;
234             } catch (IOException ex) {
235                 new Error("There was a problem with the
connection to GivEnergy.");
236                 return null;
237             }
238         }
239 
240         public String POSTshallowCharge(int charge) {
241             try {
242                 HttpURLConnection connection;
243                 String json = "{\r\n    \"value\": \""
+ charge + "\"\r\n}"; //json formed to pass into POST
API request
244                 URL url = new
URL("https://api.givenergy.cloud/registers/shallowValue
");
245                 connection = (HttpURLConnection)
url.openConnection();
246                 connection = POSTsetup(connection,
json);
247                 String response =
returnDataString(connection);
248                 return response;
249             } catch (IOException ex) {
250                 new Error("There was a problem with the
connection to GivEnergy.");
```

```
251             return null;
252         }
253     }
254
255     private HttpURLConnection
POSTsetup(HttpURLConnection connection, String json) {
//Method that sets up a connection that can send data
in JSON format to a POST endpoint so that the program
is able to send data to the GivEnergy system.
256         try {
257             connection.setRequestMethod("POST");
258             connection.setDoOutput(true);
259             connection.setDoInput(true);
260             connection.setConnectTimeout(10000);
261             connection.setReadTimeout(10000);
262
connection.addRequestProperty("Authorization",
"lGVWeCFcRrJG7NmZwOcYCjXbO3hIWq"); //sets authorization
so POST request is accepted
263             connection.addRequestProperty("Accept",
"application/json"); //sets data format
264             connection.addRequestProperty("Content-
Type", "application/json; charset=UTF-8");
265             OutputStream os =
connection.getOutputStream();
266             os.write(json.getBytes("UTF-8"));
267             os.close();
268             return connection;
269         } catch (UnsupportedEncodingException ex) {
270             new Error("There was a problem with the
connection to GivEnergy.");
271             return null;
272         } catch (IOException ex) {
273             new Error("There was a problem with the
connection to GivEnergy.");
274             return null;
275         }
276     }
277
278     private HttpURLConnection
GETsetup(HttpURLConnection connection) { //Method that
sets up the connection to a GET request
279         try {
280             connection.setRequestMethod("POST");
```

```
281             connection.addRequestProperty("Cookie",
"JSESSIONID=" + jSessionID);
282             connection.setConnectTimeout(10000);
283             connection.setReadTimeout(10000);
284             return connection;
285         } catch (ProtocolException ex) {
286             new Error("There was a problem with the
connection to GivEnergy.");
287             return null;
288         }
289     }
290 }
291
```

CurrentStateGivEnergyAPI.java

```
  1 package givenergy_apis;
  2
  3 import java.io.IOException;
  4 import org.json.JSONException;
  5 import org.json.JSONObject;
  6
  7 public class CurrentStateGivEnergyAPI extends
GivEnergyAPI {
  8
  9     private final JSONObject PLANT_SUMMARY_JO;
//Holds a JSONObject object of an API response that is
used in the rest of the class
 10
 11     public CurrentStateGivEnergyAPI() throws
IOException, JSONException {//method is called from
within a try-catch so throws is acceptable
 12         /*
 13         Uses its super class to get an API response
for the current live values of the system.
 14         Also ensures that the program is logged
into the API and if not uses the login() method to do
so.
 15          */
 16         if (super.getSessionID() == null) {
 17             super.login();
 18         }
 19         String plantSummary =
super.GETPlantSummary();
 20         this.PLANT_SUMMARY_JO =
GETJSON(plantSummary);
```

```
 21      }
 22
 23      private JSONObject GETJSON(String data) {
//Converts a String of data into a JSONObject of data
in JSON format that can be more easily accessed
 24          try {
 25              JSONObject response = new
JSONObject(data);
 26              return response;
 27          } catch (JSONException ex) {
 28              new Error("There was a problem reading
data.");
 29              return null;
 30          }
 31      }
 32
 33      public int GETBatterySOC() {
 34          try {
 35              int soc =
PLANT_SUMMARY_JO.getInt("soc");
 36              return soc;
 37          } catch (JSONException ex) {
 38              new Error("There was a problem reading
data");
 39              return -1;
 40          }
 41      }
 42
 43      public String GETBatteryPower() {
 44          try {
 45              String batPower =
PLANT_SUMMARY_JO.getString("batPower");
 46              return batPower;
 47          } catch (JSONException ex) {
 48              new Error("There was a problem reading
data");
 49              return null;
 50          }
 51      }
 52
 53      public String GETGridPower() {
 54          try {
 55              String gridpower =
PLANT_SUMMARY_JO.getString("gridPower");
 56              return gridpower;
```

```
 57          } catch (JSONException ex) {
 58              new Error("There was a problem reading
data");
 59              return null;
 60          }
 61      }
 62
 63      public String GETPVPower() {
 64          try {
 65              String pvpower =
PLANT_SUMMARY_JO.getString("pvPower");
 66              return pvpower;
 67          } catch (JSONException ex) {
 68              new Error("There was a problem reading
data");
 69              return null;
 70          }
 71      }
 72
 73      public int GETLoadPower() {
 74          try {
 75              int load =
PLANT_SUMMARY_JO.getInt("loadPower");
 76              return load;
 77          } catch (JSONException ex) {
 78              new Error("There was a problem reading
data");
 79              return -1;
 80          }
 81      }
 82
 83      public String GETCO2Today() {
 84          try {
 85              String co2 =
PLANT_SUMMARY_JO.getString("todayCo2Saved");
 86              return co2;
 87          } catch (JSONException ex) {
 88              new Error("There was a problem reading
data");
 89              return null;
 90          }
 91      }
 92
 93      public String GETCO2Month() {
 94          try {
```

```
 95              String co2 =
PLANT_SUMMARY_JO.getString("monthCo2Saved");
 96              return co2;
 97          } catch (JSONException ex) {
 98              new Error("There was a problem reading
data");
 99              return null;
100          }
101      }
102
103      public String GETCO2total() {
104          try {
105              String co2 =
PLANT_SUMMARY_JO.getString("totalCo2Saved");
106              return co2;
107          } catch (JSONException ex) {
108              new Error("There was a problem reading
data");
109              return null;
110          }
111      }
112
113      public int GETPowerInv() {
114          try {
115              int powerInv =
PLANT_SUMMARY_JO.getInt("pInv");
116              return powerInv;
117          } catch (JSONException ex) {
118              new Error("There was a problem reading
data");
119              return -1;
120          }
121      }
122 }
123
```

ChartsGivEnergyAPI.java

```
  1 package givenergy_apis;
  2
  3 import java.util.ArrayList;
  4 import java.util.Arrays;
  5 import org.json.JSONException;
  6 import org.json.JSONObject;
  7 import solarproject.DateStrings;
  8 import solarproject.Dictionary;
```

```
 9
10 public class ChartsGivEnergyAPI extends
GivEnergyAPI {
11
12     private final DateStrings DATE_STRS = new
DateStrings(); //An object that holds variables of
dates such as year, month and a date
13
14     private String[] timeVals; //An array that
holds String representations of times that are used
when generating graphs of power or energy against time.
15
16     public ChartsGivEnergyAPI(String year, String
month, String date) {
17         if (super.getSessionID() == null) {
18             super.login();
19         }
20         DATE_STRS.setYear(year);
21         DATE_STRS.setMonth(month);
22         DATE_STRS.setDate(date);
23     }
24
25     public DateStrings getDateStrs() {
26         return this.DATE_STRS;
27     }
28
29     public String[] getTimevals() {
30         return this.timeVals;
31     }
32
33     private JSONObject GETJSON(String data) {
34         JSONObject response;
35         try {
36             response = new JSONObject(data);
//turns a string representation of JSON into a
JSONObject
37             return response;
38         } catch (JSONException ex) {
39             new Error("There was a problem reading
data.");
40             return null;
41         }
42     }
43
```

```
 44      public double[][] GETEnergyByMonth(String year,
String month) { //Uses a GivEnergyAPI response to
return a set of data representing different energy uses
such as generation and demand over the course of a
month during that year.
 45          String response =
GETPlantChartByMonth(year, month);
 46          JSONObject chartJO = GETJSON(response);
//turns response into JSONObject
 47          String[] names = new String[]{"day",
"exportEnergy", "DischargeEnergyToday",
"ChargeEnergyToday", "importEnergy",
"consumptionEnergy", "InvImportEnergy",
"InvExportEnergy", "pvEnergy"};
 48          double[][] chartData = new double[9][31];
 49          for (int j = 0; j < 9; j++) {
 50              for (int i = 0; i < 31; i++) {
 51                  try {
 52                      chartData[j][i] =
chartJO.getJSONArray("data").getJSONObject(i).getDouble
(names[j]); //fetches the correct value of data from
the API response from chartJO using the data keys
stored in names
 53                  } catch (JSONException ex) {
 54                      new Error("There was a problem
reading data.");
 55                  }
 56              }
 57          }
 58          return chartData;
 59      }
 60
 61      public String[][] GETPVpowerByDay(String date)
{ //Uses a GivEnergy API response to return a set of
data about the power generation by the solar panels for
a specific date.
 62          try {
 63              String response =
GETInverterChartByDay(date);
 64              JSONObject chartJO = GETJSON(response);
 65              int length =
chartJO.getJSONArray("data").length();
 66              String[][] chartData = new
String[length][2];
 67              for (int i = 0; i < length; i++) {
```

```
 68                  chartData[i][0] =
Double.toString(chartJO.getJSONArray("data").getJSONObj
ect(i).getDouble("ppv"));
 69                  chartData[i][1] =
chartJO.getJSONArray("data").getJSONObject(i).getString
("time");
 70              }
 71              return chartData;
 72          } catch (JSONException ex) {
 73              new Error("There was a problem reading
data.");
 74              return null;
 75          }
 76      }
 77
 78      public double[][] GETEnergyByYear(String year)
{ //Uses a GivEnergyAPI response to return sets of data
representing various energy uses such as generation and
demand over the course of each month over the given
year.
 79          String response =
GETPlantChartByYear(year);
 80          JSONObject chartJO = GETJSON(response);
 81          String[] names = new String[]{"month",
"exportEnergy", "DischargeEnergyToday",
"ChargeEnergyToday", "importEnergy",
"consumptionEnergy", "InvImportEnergy",
"InvExportEnergy", "pvEnergy"};
 82          double[][] chartData = new double[9][12];
 83          for (int j = 0; j < 9; j++) {
 84              for (int i = 0; i < 12; i++) {
 85                  try {
 86                      chartData[j][i] =
chartJO.getJSONArray("data").getJSONObject(i).getDouble
(names[j]); //fetches the correct value of data from
the API response from chartJO using the data keys
stored in names
 87                  } catch (JSONException ex) {
 88                      new Error("There was a problem
reading data.");
 89                  }
 90              }
 91          }
 92          return chartData;
 93      }
```

```
 94
 95      public Dictionary
inverterValuesDictionary(String date) { //Uses the
Dictionary class to create a new Dictionary and store
array objects against a key. This is an effective way
to return a single object that holds data sets used in
graphs that show multiple sets of data on the same
axis.
 96          try {
 97              Dictionary dict = new Dictionary();
 98              String response =
GETInverterChartByDay(date);
 99              JSONObject chartJO = GETJSON(response);
100              timeVals = new
String[chartJO.getJSONArray("data").length()];
101              ArrayList pacExportList = new
ArrayList(), pacImportList = new ArrayList(), ppvList =
new ArrayList(), loadPowerList = new ArrayList(),
batPowerActualList = new ArrayList();
102              for (int j = 0; j <
chartJO.getJSONArray("data").length(); j++) {
103                  timeVals[j] =
chartJO.getJSONArray("data").getJSONObject(j).getString
("time");
104
pacExportList.add(chartJO.getJSONArray("data").getJSONO
bject(j).getDouble("pacExport"));
105
pacImportList.add(chartJO.getJSONArray("data").getJSONO
bject(j).getDouble("pacImport"));
106
ppvList.add(chartJO.getJSONArray("data").getJSONObject(
j).getDouble("ppv"));
107
loadPowerList.add(chartJO.getJSONArray("data").getJSONO
bject(j).getDouble("loadPower"));
108
batPowerActualList.add(chartJO.getJSONArray("data").get
JSONObject(j).getDouble("batPowerActual"));
109              }
110              dict.add("pacExport",
Arrays.stream(pacExportList.toArray()).mapToDouble(num
= Double.parseDouble(num.toString())).toArray());
```

```
111             dict.add("pacImport",
Arrays.stream(pacImportList.toArray()).mapToDouble(num
= Double.parseDouble(num.toString())).toArray());
112             dict.add("ppv",
Arrays.stream(ppvList.toArray()).mapToDouble(num =
Double.parseDouble(num.toString())).toArray());
113             dict.add("loadpower",
Arrays.stream(loadPowerList.toArray()).mapToDouble(num
= Double.parseDouble(num.toString())).toArray());
114             dict.add("batpoweractual",
Arrays.stream(batPowerActualList.toArray()).mapToDouble
(num = Double.parseDouble(num.toString())).toArray());
115             return dict;
116         } catch (JSONException ex) {
117             new Error("There was a problem reading
data.");
118             return null;
119         }
120     }
121 }
122
```

BatteryGivEnergyAPI.java

```
  1 package givenergy_apis;
  2
  3 import org.json.JSONException;
  4 import org.json.JSONObject;
  5
  6 public class BatteryGivEnergyAPI extends
GivEnergyAPI {
  7
  8     private final JSONObject BATTERY_DATA_JO;
//Holds a JSON data response that contains all the
battery data from the system after an API response has
been receive
  9
 10     public BatteryGivEnergyAPI() {
 11         /*
 12         Uses its super class to get a API response
JSON. This is stored in the batteryDataJO attribute of
this class.
 13          */
 14         String batteryData =
super.GETAllBatteryData();
 15         BATTERY_DATA_JO = GETJSON(batteryData);
```

```
 16      }
 17
 18      private JSONObject GETJSON(String data) {
//Converts a String of data into a JSONObject of data
in JSON format that can be more easily accessed
 19          try {
 20              JSONObject response = new
JSONObject(data);
 21              return response;
 22          } catch (JSONException ex) {
 23              new Error("There was a problem reading
data");
 24              return null;
 25          }
 26      }
 27
 28      public int GETMode() { //returns the current
system mode that is in the API response JSON
 29          try {
 30              int mode =
BATTERY_DATA_JO.getInt("mode");
 31              return mode;
 32          } catch (JSONException ex) {
 33              new Error("There was a problem reading
data");
 34              return -1;
 35          }
 36      }
 37
 38      public int GETSelfConsumptionMode() { //returns
the current self consumption mode from the API response
JSON
 39          try {
 40              int selfConsumptionMode =
BATTERY_DATA_JO.getInt("selfConsumptionMode");
 41              return selfConsumptionMode;
 42          } catch (JSONException ex) {
 43              new Error("There was a problem reading
data");
 44              return -1;
 45          }
 46      }
 47
```

```
 48      public int GETshallowCharge() { //returns the
minimum percentage value that the battery SOC will go
to
 49          try {
 50              int shallowCharge =
BATTERY_DATA_JO.getInt("shallowCharge");
 51              return shallowCharge;
 52          } catch (JSONException ex) {
 53              new Error("There was a problem reading
data");
 54              return -1;
 55          }
 56      }
 57
 58      public int GETdischargeScheduleStart() {
//returns the discharge start time from the API
response
 59          try {
 60              int dischargeStart =
BATTERY_DATA_JO.getInt("dischargeScheduleStart");
 61              return dischargeStart;
 62          } catch (JSONException ex) {
 63              new Error("There was a problem reading
data");
 64              return -1;
 65          }
 66      }
 67
 68      public int GETdischargeScheduleEnd() {
//returns the discharge end time from the API response
JSON
 69          try {
 70              int dischargeEnd =
BATTERY_DATA_JO.getInt("dischargeScheduleEnd");
 71              return dischargeEnd;
 72          } catch (JSONException ex) {
 73              new Error("There was a problem reading
data");
 74              return -1;
 75          }
 76      }
 77
 78      public int GETchargeScheduleStart() { //returns
the charge start time from the API response
 79          try {
```

```
 80              int chargeStart =
BATTERY_DATA_JO.getInt("chargeScheduleStart");
 81              return chargeStart;
 82          } catch (JSONException ex) {
 83              new Error("There was a problem reading
data");
 84              return -1;
 85          }
 86      }
 87
 88      public int GETchargeScheduleEnd() { //returns
the charge end time from the API response
 89          try {
 90              int chargeEnd =
BATTERY_DATA_JO.getInt("chargeScheduleEnd");
 91              return chargeEnd;
 92          } catch (JSONException ex) {
 93              new Error("There was a problem reading
data");
 94              return -1;
 95          }
 96      }
 97
 98      public int GETdischargeDownTo() { //returns the
value that the battery will discharge to in the
discharge period
 99          try {
100              int dischargeTo =
BATTERY_DATA_JO.getInt("dischargeDownTo");
101              return dischargeTo;
102          } catch (JSONException ex) {
103              new Error("There was a problem reading
data");
104              return -1;
105          }
106      }
107
108      public int GETchargeUpTo() { //returns the
value that the battery will charge up to in the
charging period
109          try {
110              int chargeTo =
BATTERY_DATA_JO.getInt("chargeUpTo");
111              return chargeTo;
112          } catch (JSONException ex) {
```

```
113                 new Error("There was a problem reading
data");
114                 return -1;
115             }
116         }
117
118         public void modeChange(int mode) { //uses the
inherited methods from GivEnergy to change the most of
the system with a POST request to the API
119             POSTmode(mode);
120         }
121
122         public void charge(int start, int end, int
chargeto) { //uses the API to send a POST request to
charge the battery between a start and end time and
what value to charge to
123             POSTchargeBattey(start, end, chargeto);
124         }
125
126         public void discharge(int start, int end, int
dischargeto, boolean enable) { //uses the API to send a
POST request to discharge the battery between a start
and end time and what value to discharge down to
127             POSTdischargeBattey(enable, start, end,
dischargeto);
128         }
129
130         public void dischargeflag(int flag) { //sets
the discharge flag value using a POST API request
131             POSTdischargeFlag(flag);
132         }
133
134         public void chargeflag(int flag) { //sets the
charge flag value using a POST API request
135             POSTchargeFlag(flag);
136         }
137
138         public void shallowCharge(int charge) { //sets
the shallow charge value using a POST API request
139             POSTshallowCharge(charge);
140         }
141 }
142
```

DBpvModel.java

```
  1 package databases;
  2
  3 import java.sql.ResultSet;
  4 import java.sql.SQLException;
  5 import java.time.LocalDate;
  6
  7 public class DBpvModel extends DBConnection {
  8
  9     private double[] ghiData; //Array that contains
all the forecasted GHI data stored in the database
 10
 11     private double[] pvOutputData; //Array that
contains all the power output data from the solar
panels that is stored in the database
 12
 13     public DBpvModel() {
 14         DBCredentials dbCreds = new
DBCredentials("<insert DB Name>", "<insert DB
username>", "insert DB password"); //Uses the
DBCredentials class to add the passwords and usernames
and other infomation that needs to be uesd in the
DBConnection class
 15         super.setDBCredentials(dbCreds);
 16         super.connect();
 17     }
 18
 19     public double[] getGHIdata() {
 20         return this.ghiData;
 21     }
 22
 23     public double[] getPVoutputdata() {
 24         return this.pvOutputData;
 25     }
 26
 27     public void addToPVModelData(double[]
forecastGHIdata, double[] pvOutputdata) { //Inserts new
data that can be used in the model into a database
table
 28         LocalDate date = LocalDate.now();
 29         String query;
 30         for (int i = 0; i < forecastGHIdata.length;
i++) {
 31             if ((pvOutputdata[i] == 0) |
(forecastGHIdata[i] == 0)) {
```

```
 32                 continue;
 33             }
 34             query = "INSERT INTO
pvModelData(forecastGHI,pvOutput,FK_pvModel)" + "VALUES
(" + forecastGHIdata[i] + "," + pvOutputdata[i] + ",'"
+ date + "')"; //SQL query to insert a record into the
pvModelData table
 35             perform(query);
 36         }
 37     }
 38
 39     private void deleteByID(String id) { //Can
delete a set of data points that have a specific
primary key date
 40         String query = "DELETE FROM pvModelData
WHERE pvModel.PK_pvModel='" + id + "'"; //SQL query to
delete every item with a specific foriegn key
 41         perform(query);
 42     }
 43
 44     public void getModelData() { //Gets all the
model data points from the database and stores them
into arrays that can be used elsewhere in the program
 45         try {
 46             int count = getCountpvModelData();
 47             ghiData = new double[count];
 48             pvOutputData = new double[count];
 49             ResultSet results;
 50             String query = "SELECT * FROM
pvModelData"; //SQL query to select everything from
pvModelData table
 51             results = getResponse(query);
 52             int i = 0;
 53             while (results.next()) {
 54                 ghiData[i] =
results.getDouble("forecastGHI");
 55                 pvOutputData[i] =
results.getDouble("pvOutput");
 56                 i++;
 57             }
 58         } catch (SQLException ex) {
 59             new Error("Database Error");
 60         }
 61     }
 62
```

```
 63      private int getCountpvModelData() { //Performs
SQL statement to find how many records are in the model
data table
 64          try {
 65              String query = "SELECT COUNT(*) FROM
pvModelData"; //SQL query to count the number of
records in the pvModelData table
 66              ResultSet results;
 67              results = getResponse(query);
 68              results.next();
 69              int count = results.getInt(1);
 70              return count;
 71          } catch (SQLException ex) {
 72              new Error("Database Error");
 73              return -1;
 74          }
 75      }
 76
 77      public void addToPVModel(double order5, double
order4, double order3, double order2, double order1,
double order0) { //Adds calculated model parameters to
a table in the database
 78          LocalDate date = LocalDate.now();
 79          String query = "INSERT INTO
pvModel(Order5,Order4,Order3,Order2,Order1,Order0,PK_pv
Model)" + "VALUES (" + order5 + "," + order4 + "," +
order3 + "," + order2 + "," + order1 + "," + order0 +
",'" + date + "')";
 80          perform(query);
 81      }
 82
 83      public double[] getModel() { //Looks in the
model table to get the model parameters that can be
used to make forecast predictions
 84          try {
 85              String query = "SELECT * FROM pvModel
ORDER BY date(PK_pvModel) DESC"; //SQL query to select
every value in pvModel table and order them by the date
in the primary key of that table descending
 86              ResultSet rs = getResponse(query);
 87              double[] model = new double[6];
 88              rs.next();
 89              model[0] = rs.getDouble("Order5");
 90              model[1] = rs.getDouble("Order4");
 91              model[2] = rs.getDouble("Order3");
```

```
 92                 model[3] = rs.getDouble("Order2");
 93                 model[4] = rs.getDouble("Order1");
 94                 model[5] = rs.getDouble("Order0");
 95                 return model;
 96             } catch (SQLException ex) {
 97                 new Error("Database Error");
 98                 return null;
 99             }
100         }
101 
102         public String[] getChargingPeriods() {
//Returns a list of previously calculated charging and
discharging periods to display in the GUI that are
stored overnight in a table.
103             try {
104                 String[] charging = new String[4];
105                 String query = "SELECT * FROM
SystemSettings"; //SQL query to select everything from
SystemSettings table
106                 ResultSet results;
107                 results = getResponse(query);
108                 results.next();
109                 charging[0] =
results.getString("NightChargingTime");
110                 charging[1] =
results.getString("DayChargingTime");
111                 charging[2] =
results.getString("DischargePeriod1");
112                 charging[3] =
results.getString("DischargePeriod2");
113                 return charging;
114             } catch (SQLException ex) {
115                 new Error("Database Error");
116                 return null;
117             }
118         }
119 }
120 
```

DBCredentials.java

```
 1 package databases;
 2 
 3 import java.sql.Connection;
 4 
 5 public class DBCredentials {
```

```
 6
 7      private final String dbName; //String object
that holds the DB name
 8
 9      private final String username; //String object
that holds the DB username
10
11      private final String password; //String object
that holds the DB password
12
13      private Connection connection; //Connection
object that holds the connection to the remote database
14
15      DBCredentials(String dbName, String username,
String password) { //initialises final values when
object is created
16          this.dbName = dbName;
17          this.username = username;
18          this.password = password;
19      }
20
21      public Connection getConnnection() { //A getter
for the Connection obeject
22          return this.connection;
23      }
24
25      public String getDBName() { //A getter for the
database name
26          return this.dbName;
27      }
28
29      public String getUsername() { //A getter for the
database username
30          return this.username;
31      }
32
33      public String getPassword() { //A getter for the
database password
34          return this.password;
35      }
36
37      public void setConnnection(Connection
connection) { //A setter for the Connection object
38          this.connection = connection;
39      }
```

```
40 }
41
```

DBConnection.java

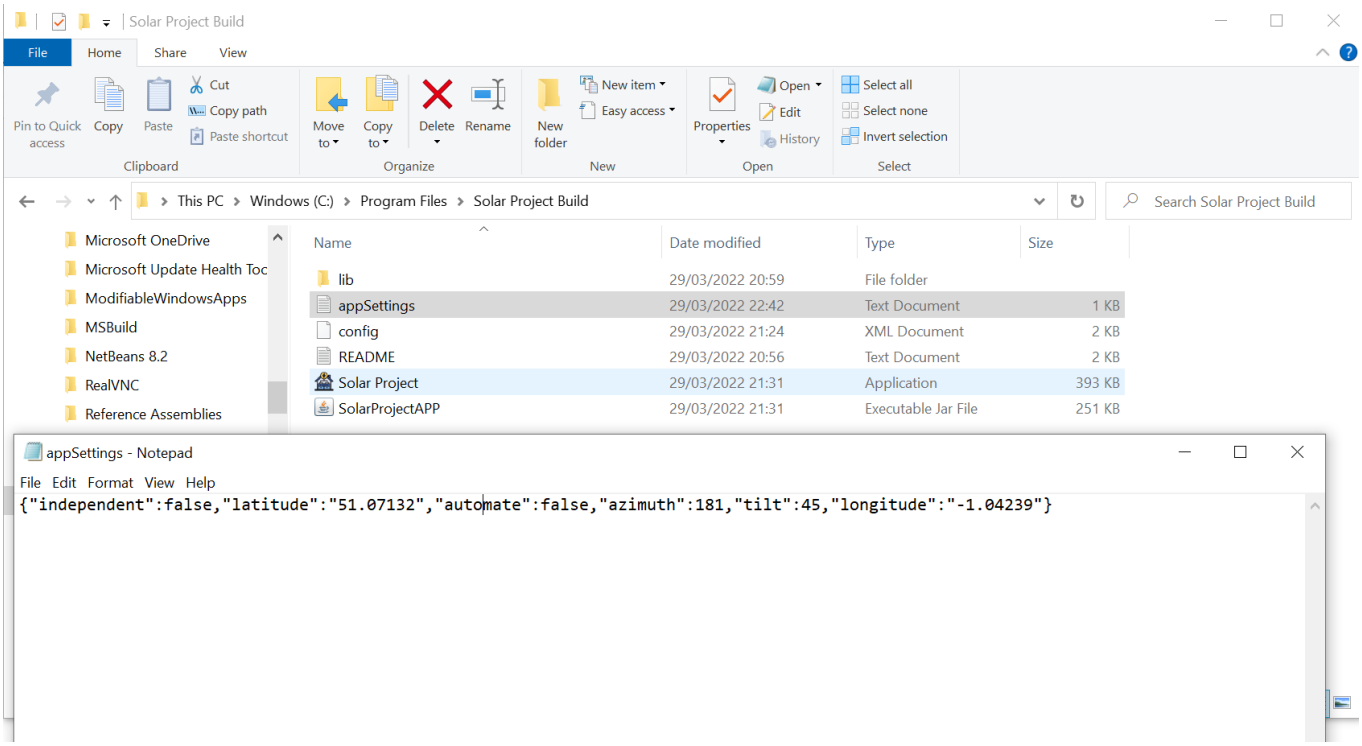
```
 1 package databases;
 2
 3 import java.sql.DriverManager;
 4 import java.sql.PreparedStatement;
 5 import java.sql.ResultSet;
 6 import java.sql.SQLException;
 7 import java.sql.Statement;
 8
 9 public class DBConnection {
10
11     private DBCredentials dbCreds; //DBCredentials
object that holds all the values needed for the
connection to the remote MySQL database
12
13     public void setDBCredentials(DBCredentials
dbCreds) {
14         this.dbCreds = dbCreds;
15     }
16
17     public void connect() { //Connects to the remote
DB using the credentials a
18         try {
19             Class.forName("com.mysql.jdbc.Driver");
20
dbCreds.setConnnection(DriverManager.getConnection("jdb
c:mysql://sql4.freesqldatabase.com:3306/" +
dbCreds.getDBName(), dbCreds.getUsername(),
dbCreds.getPassword()));
21         } catch (Exception e) {
22             new Error("Could not connect to
database: " + e);
23         }
24     }
25
26     public void disconnect() { //Closes the
connection to the database
27         try {
28             dbCreds.getConnnection().close();
29         } catch (SQLException ex) {
30             new Error("Database Error");
31         }
```

```
32      }
33
34      public void perform(String sqlquery) {
//Executes an SQL query on the database
35          try {
36              String query = sqlquery;
37              if (dbCreds.getConnnection().isClosed())
{ //Removes chance that error could occur if the
connection happened to be closed
38                  connect();
39              }
40              PreparedStatement ps =
dbCreds.getConnnection().prepareStatement(query);
41              ps.executeUpdate();
42              disconnect(); //closes the connection to
ensure that there is never more than one connection to
the database
43          } catch (SQLException ex) {
44              new Error("Database Error");
45          }
46      }
47
48      public ResultSet getResponse(String query) {
//If the query is something that will return data then
this method is used to return a ResultSet object where
the data can be found
49          try {
50              if (dbCreds.getConnnection().isClosed())
{ //Removes chance that error could occur if the
connection happened to be closed
51                  connect();
52              }
53              Statement statement;
54              ResultSet results;
55              String queryStr = query;
56              statement =
dbCreds.getConnnection().createStatement();
57              results =
statement.executeQuery(queryStr);
58              disconnect(); //Removes chance that
error could occur if the connection happened to be
closed
59              return results;
60          } catch (SQLException ex) {
61              new Error("Database Error");
```

```
62                  return null;
63              }
64          }
65  }
66
```

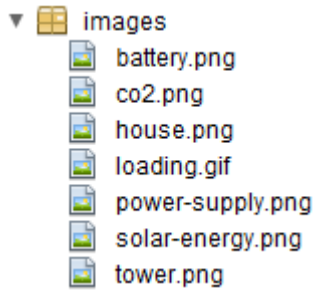
## Files

### appSettings.txt




appSettings.txt is used to store data that the program needs to remember every time it is loaded up. The data stored in this file will be changed if the user changes the settings when the program is running so it needs to be stored. This file is saved in the same directory as the compiled running executable. So, the program needs to be installed with that in mind. The data inside the file is stored in JSON format which makes changing values very easy.

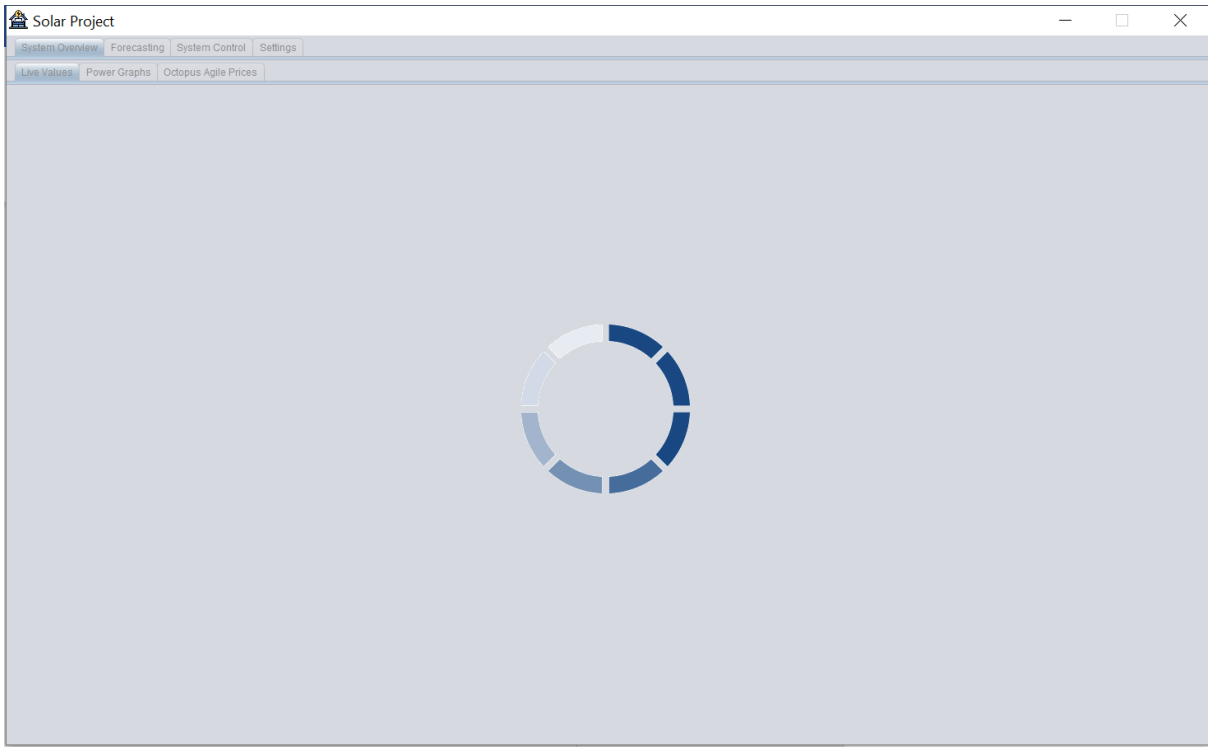
### Images



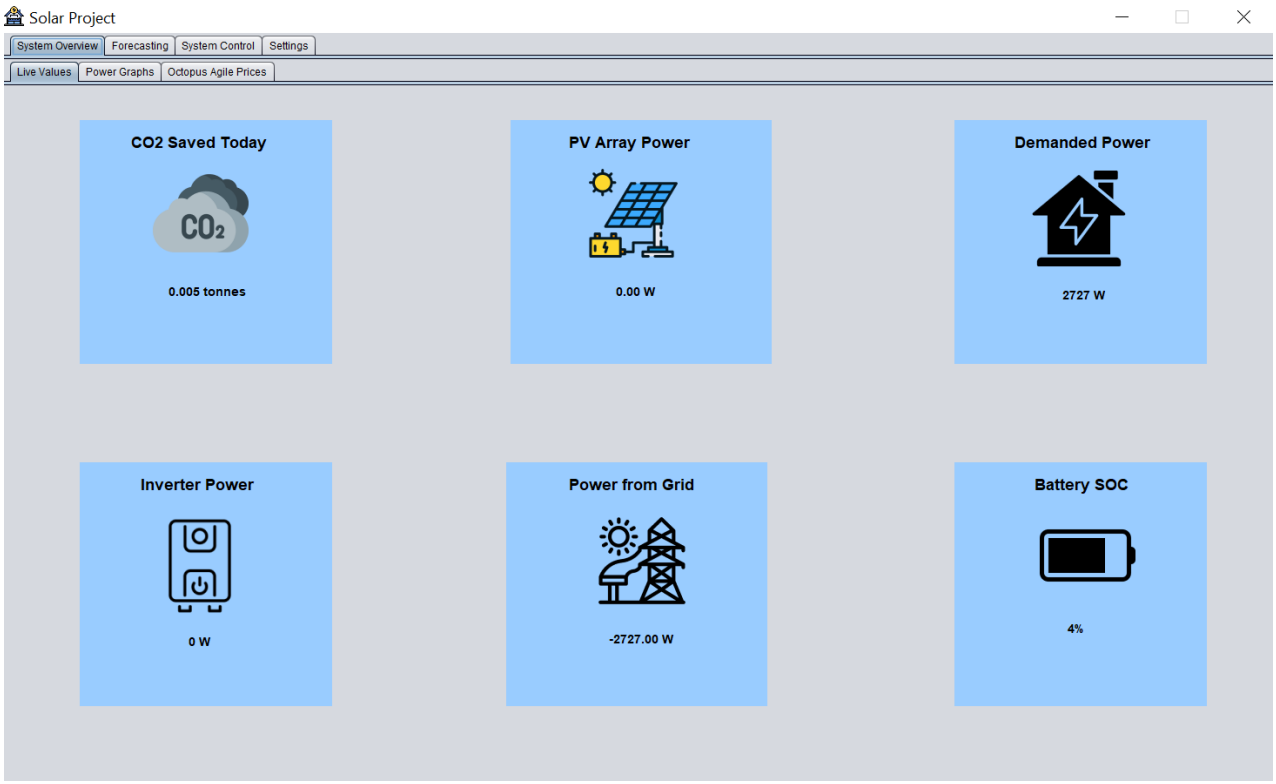

The source URL for these images are listed below:

- battery.png: https://cdn-icons.flaticon.com/png/512/664/premium/664883.png?token=exp=1650219620~hmac=df99232c23501eb86572d1e980e18659
- co2.png: https://cdn-icons.flaticon.com/png/512/573/premium/573027.png?token=exp=1650219193~hmac=060eecc8f309a2237167ee67f123f640
- house.png: https://cdn-icons.flaticon.com/png/512/3444/premium/3444114.png?token=exp=1650219302~hmac=3306031bf738f3f72ff5ca0f8dcaa3ed
- loading.png: This GIF was sourced from https://icons8.com/preloaders/ and was built using the customizer on the website.
- power-supply.png https://cdn-icons.flaticon.com/png/512/5006/premium/5006285.png?token=exp=1650219415~hmac=64bf0bfe11ff5e85c395fff996d61e60
- solar-energy.png: https://cdn-icons-png.flaticon.com/512/2933/2933980.png
- tower.png: https://cdn-icons-png.flaticon.com/512/6875/6875839.png

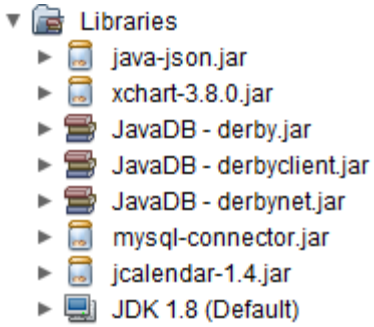
These image files are compiled into the final application and stored in an image package. The loading.gif is used in the GUI when the program is loading:




The other images are used on the live values tab to give an illustration of what data is being displayed:




## Libraries




This is a screenshot of the libraries that are being used in the program

### java-json.jar

This library has methods and classes that allow data of JSON format to be accessed very easily. Creating of JSON objects and arrays is done through the creation of a java object and methods can be run from these objects to add/remove data for any key value pair.

### xchart-3.8.8.jar

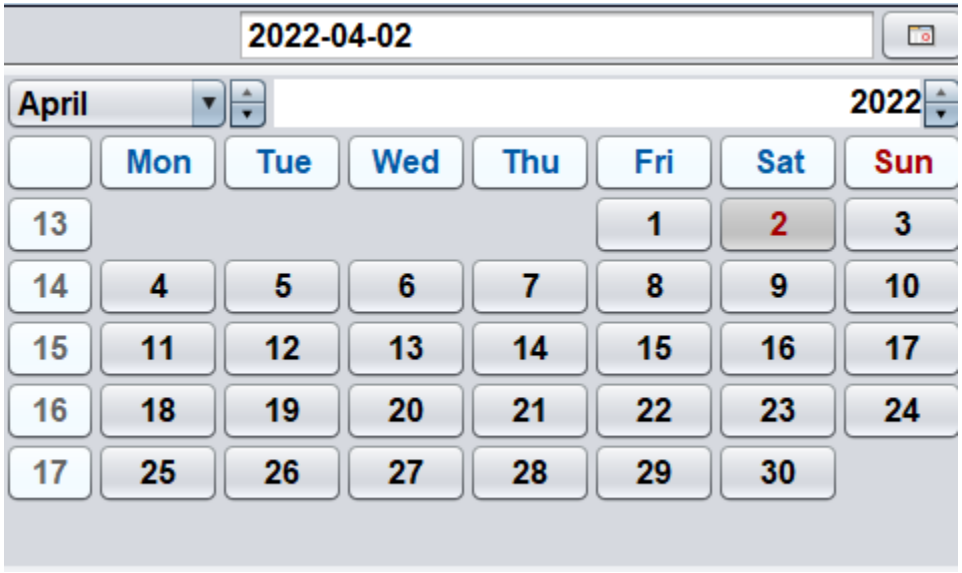
This library is used to created graphs and charts to display data. Graphs and charts can be created in JPanel form and so it can be implemented in Java Swing easily. It has methods that allow you to add datasets and customise the design of the chart, so the graph displays data in the optimum way.

### derby.jar, derbyclient.jar, derbynet.jar, mysql-connector.jar

These libraries are used for the MySQL database connection. They provide the database drivers for JDBC.

### jcalendar-1.4.jar

This library is allows a new Swing component to be used in the GUI. This component allows the user to graphically select a date from a small calendar with buttons on it. This is used in my program when the user selects a date of data they want to view on a graph. A screenshot is included below:




## Database

### pvModel table

Export of SQL code to create pvModel table:

```
--
-- Table structure for table `pvModel`
--

DROP TABLE IF EXISTS `pvModel`;
CREATE TABLE `pvModel` (
  `Order5` double NOT NULL,
  `Order4` double NOT NULL,
  `Order3` double NOT NULL,
  `Order2` double NOT NULL,
  `Order1` double NOT NULL,
  `Order0` double NOT NULL,
  `PK_pvModel` varchar(10) NOT NULL DEFAULT ''
)
```

```
--
-- Indexes for table `pvModel`
--
ALTER TABLE `pvModel`
  ADD PRIMARY KEY (`PK_pvModel`);
```

pvModel design:

| | # | Name | Type | Collation | Attributes | Null | Default |
|---|---|---|---|---|---|---|---|
| ☐ | 1 | Order5 | double | | | No | None |
| ☐ | 2 | Order4 | double | | | No | None |
| ☐ | 3 | Order3 | double | | | No | None |
| ☐ | 4 | Order2 | double | | | No | None |
| ☐ | 5 | Order1 | double | | | No | None |
| ☐ | 6 | Order0 | double | | | No | None |
| ☐ | 7 | PK_pvModel 🔑 | varchar(10) | latin1_swedish_ci | | No | |

pvModel data:

| # | Order5 | Order4 | Order3 | Order2 | Order1 | Order0 | PK_pvModel |
|---|---|---|---|---|---|---|---|
| 1 | 2.9731273949789215E-11 | -8.926983165505852E-8 | 8.693438697582323E-5 | -0.026570488335437403 | 4.156709955394035 | -6.2096968900877... | 2022-03-23 |
| 2 | 1.8241797445621483E-11 | -5.989863010799623E-8 | 6.205758978583553E-5 | -0.01874206596755812 | 3.30492205786868 | 9.428616903489456 | 2022-03-24 |

## pvModelData table

Export of SQL code to create pvModelData table:

```
--
-- Table structure for table `pvModelData`
--

DROP TABLE IF EXISTS `pvModelData`;
CREATE TABLE `pvModelData` (
  `forecastGHI` double NOT NULL,
  `pvOutput` double NOT NULL,
  `FK_pvModel` varchar(10) NOT NULL
)
```

pvModelData design:

| | # | Name | Type | Collation | Attributes | Null | Default |
|---|---|---|---|---|---|---|---|
| ☐ | 1 | forecastGHI | double | | | No | None |
| ☐ | 2 | pvOutput | double | | | No | None |
| ☐ | 3 | FK_pvModel | varchar(10) | latin1_swedish_ci | | No | None |

pvModelData data:

| # | forecastGHI | pvOutput | FK_pvModel |
|---|---|---|---|
| 1 | 4.0 | 16.8 | 2022-03-23 |
| 2 | 43.0 | 91.5 | 2022-03-23 |
| 3 | 132.0 | 222.0 | 2022-03-23 |
| 4 | 248.0 | 354.4 | 2022-03-23 |
| 5 | 372.0 | 667.0 | 2022-03-23 |
| 6 | 496.0 | 1572.3 | 2022-03-23 |
| 7 | 613.0 | 1492.7 | 2022-03-23 |
| 8 | 719.0 | 3463.2 | 2022-03-23 |
| 9 | 778.0 | 4229.3 | 2022-03-23 |
| 10 | 884.0 | 4557.4 | 2022-03-23 |
| 11 | 940.0 | 4769.7 | 2022-03-23 |
| 12 | 976.0 | 4911.3 | 2022-03-23 |
| 13 | 992.0 | 5017.6 | 2022-03-23 |
| 14 | 986.0 | 4915.5 | 2022-03-23 |

### SystemSettings table

Export of SQL code to create pvModelData table:

```
--
-- Table structure for table `SystemSettings`
--

DROP TABLE IF EXISTS `SystemSettings`;
CREATE TABLE `SystemSettings` (
  `NightChargingTime` varchar(30) NOT NULL,
  `DayChargingTime` varchar(30) NOT NULL,
  `DischargePeriod1` varchar(30) NOT NULL,
  `DischargePeriod2` varchar(30) NOT NULL
)
```

SystemSettings design:

| | # | Name | Type | Collation | Attributes | Null | Default |
|---|---|---|---|---|---|---|---|
| ☐ | 1 | NightChargingTime | varchar(30) | latin1_swedish_ci | | No | None |
| ☐ | 2 | DayChargingTime | varchar(30) | latin1_swedish_ci | | No | None |
| ☐ | 3 | DischargePeriod1 | varchar(30) | latin1_swedish_ci | | No | None |
| ☐ | 4 | DischargePeriod2 | varchar(30) | latin1_swedish_ci | | No | None |

SystemSettings data:

| # | NightChargingTime | DayChargingTime | DischargePeriod1 | DischargePeriod2 |
|---|---|---|---|---|
| 1 | 00:30:00 - 03:00:00 | 12:30:00 - 14:00:00 | 08:00:00 - 11:30:00 | 16:00:00 - 19:30:00 |

## Evaluation

Overall, I am very satisfied with the program. I am pleased that I was able to access so much data from the solar PV system and present it in aesthetic ways in the program. One of the main

challenges when developing the program was the complex mathematical model. This model aimed to use past solar PV power output data and solar radiation data to predict future performance of the solar panels. Once this model was working this allowed me to design an algorithm that could recommend when the user charges and discharges the battery using this prediction. This allowed my project to be very useful to the user as they can get a calculated battery charging and discharging strategy and they no longer rely on guessing when the best time for charging discharging would be. I am also pleased that the solar PV system can be controlled through the program using buttons and input boxes via the GivEnergy API.

Unfortunately, I did not quite have time to finish my extension objectives which were automated changing of the system modes and daily email updates to the user. However, I do not think these missing features particularly reduce the quality as all the data and functionality is still available to the user, just only via the GUI application.

## Objectives

| Objective | Test Number | Evaluation | Passed |
|---|---|---|---|
| 1.1 | 1 | This passed test 1 as the program is able to use forecast data to calculate and predict solar panel performance and the program can display forecast graphs. | YES |
| 1.2 | 2 | The objective was met well as there is a bar chart which shows the agile electricity prices as proved by test 2. | YES |
| 1.3 | 3 | This objective was met well as the live values tab and the power graph both show accurate and relevant data from GivEnergy which was proved by test 3. | YES |
| 1.4 | 15 | This objective was met well as the program does return an error if the API response is erroneous which was proved by test 15. | YES |
| 1.5 | 16 | This objective was met well as the program does return an error if the internet connection is not available which was proved by test 16. | YES |
| 2.1 | 4 | This objective was met well as the complex mathematical model can be used and the model can be viewed in a graph with all the datapoints which was proved in test 4. | YES |
| 2.2 | 4 | This objective was met well, and the model function can also be viewed on the model scatter graph as a curve which was proved in test 4. | YES |
| 2.3 | 4 | This objective was met well as the program can use a database table to collect data about the current day's | YES |

| | | | |
|---|---|---|---|
| | | charging and discharging times which was proved in test 4. | |
| 3.1 | 5 | This objective was met as there is a tab in the GUI where solar panel information can be inputted which was proved in test 5. | YES |
| 3.2 | 5, 29 | Objective was met as a slider was used which was proved in test 5. (The buttons/components used in this objective were tested and passed by test 29) | YES |
| 3.3 | 5, 30 | Objective was met as a slider was used which was proved in test 5. (The buttons/components used in this objective were tested and passed by test 30) | YES |
| 3.4 | 5, 31, 32 | Objective was met as text fields were used which was proved in test 5. (The buttons/components used in this objective were tested and passed by test 31 and 32) | YES |
| 3.5 | 17 | This objective was met well as the program does return an error if the inputted longitude or latitude is outside an acceptable range which was proved in test 17. | YES |
| 4.1 | 5 | This objective was met well as the program is able to use text files to read data about the solar panels which was proved in test 5. | YES |
| 5.1 | 6.1, 6.2, 6.3, 6.4, 35, 37, 40 | This objective was met well as the radio buttons can be used to select any of the modes that GivEnergy offers which was proved in tests 6.1, 6.2, 6.3 and 6.4. (The buttons/components used in this objective were tested and passed by test 35, 37 and 40) | YES |
| 5.2 | 6.1, 6.2, 6.3, 6.4 | This objective was met well as the radio buttons can be used to select only one mode at a time to avoid errors which was proved in tests 6.1, 6.2, 6.3 and 6.4. | YES |
| 5.3 | 18, 38, 39, 41, 42 | This objective was met as if the inputted time is of the wrong format then an error window pops up to tell the user that the input was wrong which was proved in test 18. (The buttons/components used in this objective were tested and passed by test 39, 41, 42) | YES |

| | | | |
|---|---|---|---|
| 5.4 | 6.5, 43, 44, 45, 46 | This objective was met well as these features are available to the user which was proved in test 6.5. (The buttons/components used in this objective were tested and passed by test 44, 45, 46) | YES |
| 5.5 | 6.6, 47 | This objective was met well as the minimum battery charge is able to be set by the user with no errors which was proved in test 6.6. (The buttons/components used in this objective were tested and passed by test 47) | YES |
| 5.6 | 6.6, 49 | This objective was met well as the user is able to use the save changes button to finalise the settings with no errors which was proved in test 6.6. (The buttons/components used in this objective were tested and passed by test 49) | YES |
| 5.7 | 6.7, 48 | This objective was met well as the user is able to press a reset form button and revert to the previous values which was proved in test 6.7. (The buttons/components used in this objective were tested and passed by test 48) | YES |
| 6.1 | 7, 22, 23, 24, 25, 26 | This objective was met well as the buttons can be used to select all 5 for the data sets and any combination for any date which was proved in test 7. (The buttons/components used in this objective were tested and passed by test 23, 24, 25, 26) | YES |
| 6.2 | 7, 21 | This objective was met well because a date selector component can be used to graphically select a date which also helps reduce error in user input which was proved in test 7. (The buttons/components used in this objective were tested and passed by test 21) | YES |
| 7.1 | 8 | This objective was met well as all the information displayed in the program and all controls were laid out in the tabbed layout in a logical order which was proved in test 8. | YES |
| 7.2 | 8 | This was met and each main tab had several sub-tabs which was proved in test 8. | YES |

| | | | |
|---|---|---|---|
| 7.3 | 8 | This was met as the user is able to intuitively select a different tab that is labelled accordingly to what is contained within it which was proved in test 8. | YES |
| 8.1 | 13, 28 | This objective was met successfully, however it relies on the user having to press a button to update it at the end of the day so I think it could be improved by some automation. This was proved by test 13. (The buttons/components used in this objective were tested and passed by test 28) | YES |
| 8.2 | 13 | This objective was met successfully as the model is able to be updated using power data from GivEnergy which was proved in test 13. | YES |
| 8.3 | 13 | This objective was met successfully as the model is able to be updated using solar radiation data from Solcast which was proved in test 13. | YES |
| 8.4 | 13 | This objective was met successfully as the model takes into account the incident light on the panels at each point in the day when the solar radiation data was from which was proved in test 13. | YES |
| 8.5 | 13 | The objective was met as the program is able to add new model data values into the database which was proved in test 13. | YES |
| 8.6 | 13 | The objective was met as a regression calculation can happen on the datapoints to create the mathematical model which was proved in test 13. | YES |
| 9.1 | 14 | This objective was met as the recommended charging and discharging periods are based upon the electricity prices and predicted power output. This was proved in test 14. | YES |
| 9.2 | 14 | This objective was met as the program can calculate energy output from power output at periods in the day. This was proved in test 14. | YES |
| 9.3 | 14 | This was met well as the program is able to display suggested time periods in the day which are the most optimum as determined by the algorithm. This was proved in test 14. | YES |

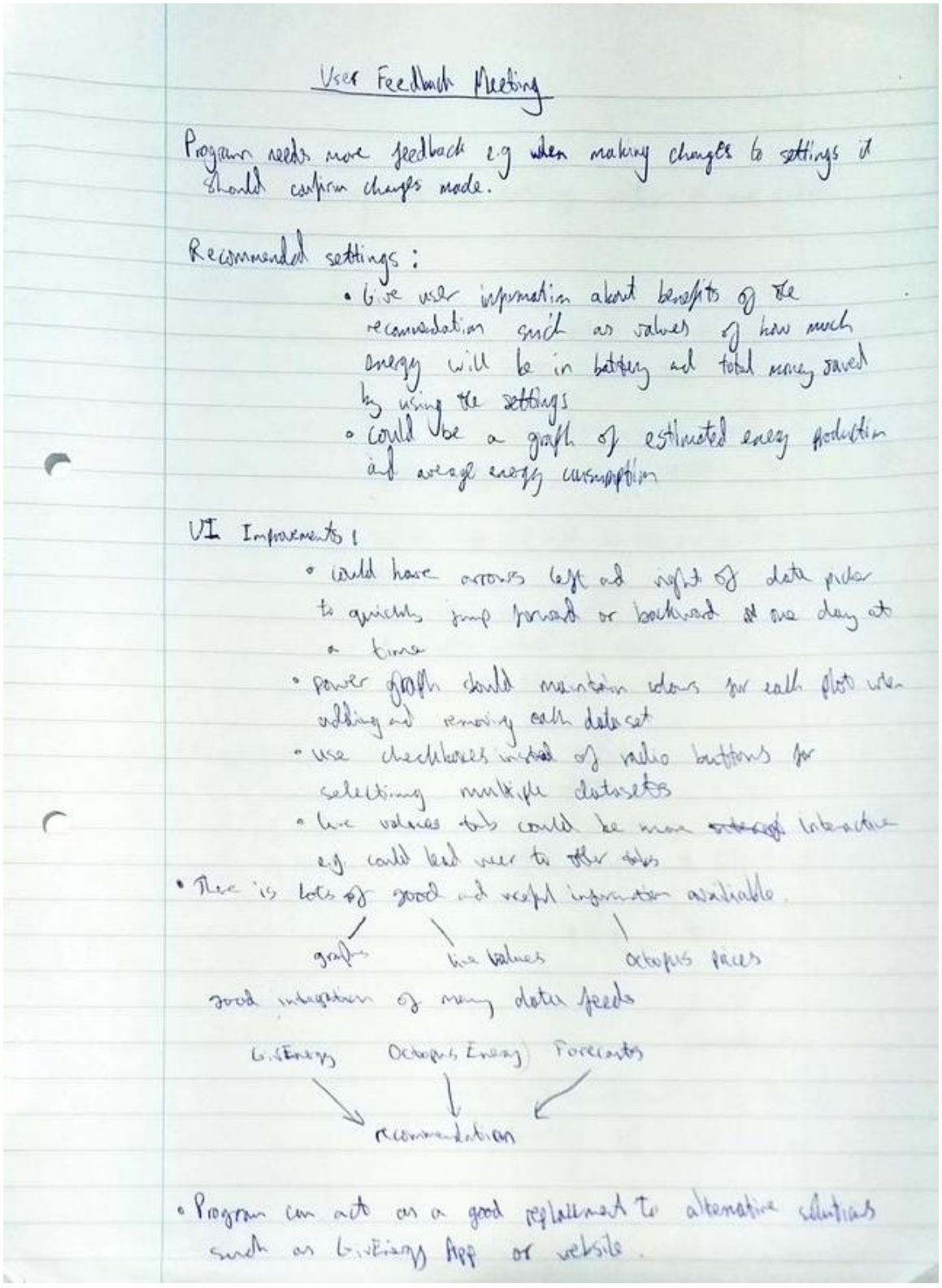
| | | | |
|---|---|---|---|
| 9.4 | 14 | This was met well as the program is able to display suggested time periods in the day which are the most optimum as determined by the algorithm. This was proved in test 14. | YES |
| 9.5 | | This was an extension objective and unfortunately it was not met. | NO |
| 9.6 | 14 | This passed well as the user is able to look at the charging/discharging periods recommendations and then go to the adjacent tab to make changes to the solar PV system settings. which was proved in test 14. | YES |
| 10.1 | 8 | This objective was met as there are never any more than 6 subtabs per main tab. This was proved in test 8 | YES |
| 10.2 | 8 | This objective was met as all of these tabs were included in this arrangement and they all display useful information or provide useful functionality. This was proved in test 8. | YES |
| 10.3 | 9.1 | This was met well because on any tabs that had user input the components used to input into the program were grouped but functionality and lined up, so they were not skew so that they made most logical sense. This was proved in test 9.1. | YES |
| 10.4 | | This was an extension objective and unfortunately it was not met | NO |
| 10.5 | 9.2 | This objective was met as a loading spinner animation does show when the program is started. This was proved in test 9.2. | YES |
| 11.1 | 10 | This objective was met very well as power data is displayed on a graph. I am pleased how this graph turned out as it has the ability to display different permutations of datasets as decided by the user. Also, the data displayed can come from any date which the user can also control. This was proved in test 10. | YES |
| 11.2 | 10, 27 | This objective was met well as the bar chart clearly shows how the price varies over the course of the day and the user is able to select a date from when the data is from. This was proved in test 10. (The buttons/components used in this | YES |

| | | | |
|---|---|---|---|
| | | objective were tested and passed by test 27) | |
| 11.3 | 10 | This objective was met as there is a graph that displays predicted power output from the solar panels over the next 48 hours. I am pleased that this graph is turning out to be fairly accurate as well. This was proved in test 10. | YES |
| 11.4 | 10 | This objective was met as there is a graph which displays the model function and all the datapoints. This was met well as it gives the user a good understanding of the accuracy of the model and how it was derived. This was proved in test 10. | YES |
| 11.5 | | This objective was not met as I did not have time to implement this. It Is not a massive loss to the project that this was not completed as the rest of the program can function fully without it but it would be good for the user to be able to see the accuracy and precision of the predictions the program is making. | NO |
| 11.6 | 10 | This objective was met as all these forecast graphs are displayed. This was proved in test 10. | YES |
| 12 | 11 | This objective was met well as all the key live values are displayed and are useful and interesting to the user. This was proved in test 11. | YES |
| 13 | | This objective was an extension and it was not met unfortunately as the program is unable to send the user daily emails about the system. | NO |
| 14.1 | 12, 33 | This objective was met as the program has a panel details tab where this information can be entered. This was proved in test 12. (The buttons/components used in this objective were tested and passed by test 33) | YES |
| 14.2 | 12, 33 | This objective was met as the program has a panel details tab where this information can be entered. This was proved in test 12. (The buttons/components used in this objective were tested and passed by test 33) | YES |

| | | | |
|---|---|---|---|
| 15.1 | 19 | This objective was met as the program is able to store all model datapoints in one table. This was proved in test 19. | YES |
| 15.2 | 19 | This objective was met as the program is able to store complex mathematical model coefficients in a database table. This was proved in test 19. | NO |
| 15.3 | 19 | This objective passed as the program is able to access the database tables and use the data within the program. This was proved in test 19. | YES |
| 15.4 | 16 | This objective was met well as an error message does show if the device cannot connect to the database due to internet connection issues. This was proved in test 16. | YES |
| 15.5 | 19 | This objective was met as during the calculations of the model if a value is NaN then the program will remove that entry so that the model data is valid and only valid data is added to the database. This was proved in test 19. | YES |
| 15.6 | 19 | This objective was not met but I think the only time it would really make a difference is when the model is updating as lots of data is being transferred and lots of calculations are happening. This would help with the user experience as the user would know that the processing is still happening if there is a loading animation. This was proved in test 19. | NO |
| 16.1 | 20 | This objective was met as the program is able to store data in JSON format in text files. This was proved in test 20. | YES |

## User Feedback

I met with my client to discuss good features and areas to improve on with a working version of my program. I have added an image of our written meeting notes which cover all points we discussed. Below the image I have analysed the good features and suggestions that the client discussed with me.




## Good Features

The client expressed that overall, the program had good integration of many data feeds and collated them and presented the data in useful ways through graphs and charts. He felt that all the data he would need when using the program was available to him within the program including data from the inverter, Octopus Energy Agile tariff electricity prices and weather forecasts. I was pleased to

hear this as it has assured me that my program has been successful in accessing and presenting all the data user needs and so the program can be genuinely helpful. The live values tab in the program was also a good and helpful feature. This was something I was keen to implement as it is very interesting to be able to see exactly what the system is doing at the current time and so I am pleased that the client also found this to be a good feature. The client liked how the program could predict the power output of the solar panels and that it could use this information to give suggested charging and discharging periods for the battery. This links back to the second to last bullet point on the meeting notes where the client was saying how there was good integration of many data feeds. It was very important that the client was pleased with this feature as this is the most complex and significant part of the program. The client said how this program could act as a viable replacement to the alternate systems such as the GivEnergy website or app and the fact it also combined this with weather forecasts and electricity prices was even better. This feedback was really positive me as it confirmed that my program had provided a decent and useful solution to the problem the client was facing.

### Suggested Changes and Improvements

The client made some suggestions about the recommended settings tab in the program. The first was to have some more transparency to the user about how the charging and discharging periods were being calculated. This could be achieved by showing a few graphs of estimated energy production and average energy consumption. This would not be very hard for me to implement as all this information about energy consumption is available to me through the GivEnergy API and I already have code to calculate forecasted energy production in the Energy class. For creating the graphs, I would use the XChart graphing libraries that I have already used in my program.

They also suggested it would be good for the program to show the benefits of charging and discharging at the recommended times. This could include projections of amount of energy that will be stored in battery at certain times of the day if the recommended settings were to be followed. Then the user could be able to see the benefit if future battery SOC was plotted on a graph for the rest of the day. This graph would be created using the XChart library. It could also combine this with the electricity prices and perform a calculation to find the total cost of electricity saved by using these recommendations. I would be able to implement this as the electricity prices are available through Octopus Energy API so to work out electricity cost saved, the program would have methods to calculate how much energy we consumed in the house and at what times of the day and where the energy came from. There would need to be a method that works out how much energy in each half hour period of the day did not come from the grid then it can use the electricity prices for that time of day and that amount of energy to calculate how much money could be saved. To accommodate these methods in the program I would make a new class called Costs that is designed to perform calculations on the cost benefits of the system.

The client also suggested some UI improvements. The first point was to have left and right arrows around the date selector to easily choose a date one more or one less than the currently selected date. I could use Swing button components for this as it would be easy to integrate and would provide an effective solution. This would speed up the process of choosing a date as only one button would need to be clicked and it is more intuitive than clicking on a day in the interactive calendar. This would be implemented with arrow buttons that have a listener using the java swing EventListener class so that when clicked changes the value of the date in the date selector and then run the required methods to change the data that is being displayed on the graphs.

Also, the client suggested using checkboxes instead of radio buttons to choose which datasets should be added to the power graph as it is more conventional. There are checkbox components in Swing and so I can use them instead of the radio button component. This is a minor change that would need to be made to the swing GUI, but no logical code would need to change.

The client also said when navigating the GUI there should be more user feedback. For example, on the tab which allows the user to set the solar panel tilt, azimuth, and location the save changes button should give an indication that the save was successful as it currently gives no feedback that anything happened after pressing the button. I could implement this by having a text output on the screen saying "changes saved" which shows after the user has pressed the button and then disappears after a few seconds. This could be achieved using the JDialog class which would allow the message to pop up in a small window which would be a neat solution.

## Improvements

Overall, my program provides an adequate and effective solution to the client's problem. It significantly reduces the amount of time and effort the client needs to put into managing the solar PV system as they only need to use one app to view all the relevant data and have full control over the system. I think the best point about the program is its ability to display and use data from all the sources such as the weather forecast data, electricity price data and solar PV system data. With access to all this data the program can give the client good recommendations and useful foresight into the systems performance and the system will definitely be able to perform at a much more optimal level than if not using the program.

Objective 10.5 is an is an objective I would like to have implemented and it is a graph that displays actual solar panel power output and predicted solar panel power output for the last 48 hours on the same graph. This would be a particularly useful feature as the user would be able to see how accurate and precise the program is with predicting power output and therefore, they would be able to gauge how much trust they put into it. However, it is not essential that this is completed as it is only another display of data. It would not affect how the rest of the program operates or the decisions it makes. Its purpose would be to give the user a greater insight of the quality of the decisions the program is making and the mathematical model those decisions are based upon.

Another negative point on the program which was highlighted in both testing and client feedback is how the program gives user feedback about what it is currently doing. For example, when the 'save changes', button is pressed in the settings tab the button is just pressed and there is no confirmation that the changes were actually saved and this could lead to confusion for the client. Objective 14.2 was also not met under a similar issue. This objective is when there is a lot of complex processing and transfer of data between the program and databases or APIs then a loading spinner animation should show to indicate that the program is currently busy. This is aimed towards when the user presses the update model button as there tends to be a few seconds when the program is unresponsive as updating the model is a large and complex process. If there was an indication that the user should wait until the program is finished processing such as the loading spinner, then it would prevent the user being confused by the unresponsiveness of the program. However, I have implemented the loading animation when the program is first started as this is also another time when there is intensive processing.

Unfortunately, I did not manage to complete my extension objectives. I have listed them below:

4.8. [Extension] There will be a toggle button that allows the program to automate setting changes so that the user never has to do the changes manually

8.5. [Extension] If the user has selected that the program automates the changing of modes then these calculated timings and modes will be dealt by with the server, and everything will be automated

9.4. [Extension]A toggle button will be included that allows an automatic system control to take over which means the user does not have to open the software as often

10.5. [Extension] Graph comparing predicted power output and actual power output over the previous 48 hours

12.0. [Extension] The program will send a daily email to the client with graphs for predicted power output and electricity pricing over the next day, it will also include which settings it plans to change

To improve my program, I could add in these extension objectives as fairly major upgrades which would significantly improve the user experience.

The extension objectives all rely on another program I could write and that could run on a server, so it is running all the time. For example, I could run that program on a raspberry Pi. That program would provide the user with information about the performance of the system over the last day as well as the predicted power output for the next day and electricity prices for the next day. It would do this by sending an email to the client with graphs attached to display the data and a summary written in the email. This would improve the project as it would mean the client would not have to keep opening the program to view the data which could become inconvenient if it was done frequently. The client could instead easily view key data on any device with access to their emails. The other purpose of this server program would be to automate system mode changes so that the user does not need any input at all. The program would look at the recommended charging and discharging times and set timers in the program to communicate with the GivEnergy API at the right times and make the suggested changes.