REFLECTIVE JOURNAL

Week 1(07/10/2024 - 12/10/2024)

Completed tasks/trainings

- 11/10/2024
    - Research the structure of Windows Presentation Foundation (WPF) framework for graphical user interface (GUI) design. This include the file type XAML, its language and format, as well as the way C# interacts with it by controlling the operating logic behind the GUI.
    - Explore the design pattern Model-View-ViewModel (MVVM) and its application in designing graphical user interface.
- 12/10/2024
    - Research the MVVM design pattern more in-depth, inspecting other programs' usage of the pattern, as well as the logical structure behind each file.
        - The Model layer of the pattern control the database of the application, responsible for handling data acquisition, data processing, and data structure of each of the application objects.
        - The View layer creates the visual aspect of the GUI, handling layout design, elements (button, text block, etc.) design, custom styling, or window modifications. It is unaware of the Model layer but using the ViewModel layer to understand what needs to be updated instead.
        - The ViewModel layer connect the previous two layers through command defining, data conversion, or interfacing. Classes within this layer are responsible for notifying any changes in variables on the Model side and updating the View side accordingly.
- 13/10/2024
    - Implement the basic structure of the GUI for the EMS software, include:
        - Core classes to handle fundamental functions of the GUI elements.
            - ObservableObject – The base class for all ViewModel classes, able to notify the View files if any variables have been changed.
            - RelayCommand – Class used to delegate all the command created to the CommandManager class in the system.
        - Model classes contain all the data for the different views possible of the application, including:
            - Main – The main dashboard of the application, presents the main graphs (or any visual representation of the data read) for all of the parameters.
            - Current – The dashboard contains all of the different types of graphs for the current parameter.
            - Voltage – The dashboard contains all of the different types of graphs for the line voltage and phase voltage parameters.
            - Power - The dashboard contains all of the different types of graphs for the power factor, active power, reactive power, and apparent power parameters.

        - View files contain the design and layout of all the dashboards mentioned.
        - ViewModel classes use the Core and Model classes to relay information to the View files to update the GUI.
        - Styles files contain the custom styling of each of the GUI elements.
    - The finished GUI only present a frequently updated random number in a range for each of the parameters in their respective views at this stage.

Week 2 expected outcomes

- Research on the Modbus RTU communication protocols.
- Investigate how to use the WSMBT tool to control the protocol in C#.
- Implement the protocol in the existing GUI.

Week 2(14/10/2024 - 19/10/2024)

Completed tasks/trainings

- 18/10/2024
    - Research the mechanisms behind the Modbus RTU communication protocol:
        - The structure of a typical Modbus RTU message includes Slave ID, Function Code, Special Data, and CRC.
        - Slave ID can be any values in the range from 0 to 247.
        - Function Code dictates which function the messages serve. The functions can either be reading or reading/writing single or multiple discrete or analogue values. This can also be the error codes, where the original function codes have their binary values altered if there are any errors during data transmission.
        - These values are classified as one of four tables: Discrete Output Coil (DO), Discrete Input Contacts (DI), Analog Input Registers (AI), and Analog Output Holding Registers (DO). Each of them has distinctive register number pool and access level.
        - Special Data defines:
            - The address of the first register where the requested data are stored and the number of registers these data can be stored in for master side.
            - The number of bytes returned, and the data stored in each byte for slave's side.
        - CRC ensured the requested and received data matched.
- 19/10/2024
    - Analysed the project's network topology and determines the way data are transmitted from power plants to EMS server, which involves encapsulating the Modbus RTU messages with TCP/IP and send over internet.
    - This means that the EMS software needs to be able to pick up the TCP/IP package using Modbus protocol to perform further analysis.
    - The functions used was 0x03, which is Read Holding Register. This is particularly useful to transmit analogue data such as the parameters required for GUI display.

    - The structure of the Modbus RTU message was then determined, with each parameters' value assigned to an address number in the fourth data table, starting at 0. This means that the registered address of the first value is 40001 and the address of the first value is 1.
    - There will be 6 addresses allocated for transmission, corresponding to Current, Phase Voltage, Line Voltage, Active Power, Reactive Power, Power Factor in that order. This will change according to the meters' documentation and how they save the measured parameters to the Modbus RTU message.
- 20/10/2024
    - Implement the Modbus RTU over TCP/IP communication protocol into the existing GUI using C# NModbus library.
    - This involves developing a class in the Model folder, which responsible for setting up the TCP connection, Modbus over TCP/IP master, and reading the holding register from the slave of ID 1 connecting to the same IP address and port.
    - The read data are saved in a list, which is constantly getting replaced with new list every time the master read new data.
    - The rest of the Model's files will inherit from this class, which allow them to call the function and adding to the elapsed event using C#'s innate class Timer.
    - The program was then tested using a slave simulation software called Modbus Slave, which increment each register by 1 every second.

Week 3 expected outcomes

- Implement handling multiple slaves' connection in the ModbusMaster class.
- Finding different simulation software that are free and can enable random number generation for better data representations.

Week 3(21/10/2024 - 26/10/2024)

Completed tasks/trainings

- 25/10/2024
    - Reviewed the topology of the project.
    - Identified the method to observe and monitor power consumption of multiple plants, which required the Modbus master to send read holding register request to multiple slaves.
        - The solution found is to ask users to input their desired slave's id to which the master send the requests to.
        - This allows users to monitor power consumption of one plant at a time based on the input id.
    - Investigated and installed the free, open-source Modbus slaves' simulator Open ModSim, which allows randomly generated numbers to be sent to an IP address.
        - To simulate multiple plants (slaves), the slave id will be changed during application operation.
- 26/10/2024
    - Implement basic GUI textbox to prompt users to input the desired slave id.
    - Styling the textbox to fit the theme of the existing GUI.

    - Refactor the ModbusMaster class to ModbusMasterModel class to use the dynamic field _slaveId that will be updated by users as the passed in slave id for the method responsible for reading holding registers instead of hard coding the slave id in.
    - Add the ModbusMasterVM class to handle passing information from the MosbusMasterModel class to the view files.
    - Binding the data context of the view model with the Text property of the text box in view.
- 27/10/2024
    - Troubleshooting the problem where the data in the view model do not get updated when the Text in the textbox updated.
        - Inspect the data flow between classes.
        - Specify the correct data context used.
        - Manage the notifications of property changes in view model and model.
        - Redesign the styling of the text box so that the modified text box created in the TextBox xaml file in the Style folder does not override and cover up that created in the view folder. This caused the text received when calling the property Text of the text box in the code behind the view xaml file not being the same as that shown on the GUI.
    - Refactor the ModbusMasterModel class to know when the changes in SlaveId happens as well as save such changes when a static event being invoked.
        - This allows the change in SlaveId is being reflected for every instance of ModbusMasterModel later on.

Week 4 expected outcomes

- Start the next step of the project (2.2)
- Look at different plotting and data analysing techniques for the read in data.
- Investigate how to put these graphs onto the existing GUI.

Week 4(28/10/2024 - 03/11/2024)

Completed tasks/trainings

- 01/11/2024
    - Fix bugs in the previous iteration of the software:
        - Change the field _slaveId in ModbusMasterModel class to static to let every instance of this class created containing the same slave Id.
        - Change the instance of the Model classes created in the View Model class to read-only to add data protection, restricting unnecessary changes to these instances.
    - Research the library LiveCharts.Wpf to plot the charts required for the second stage of the project. This graph provided an easier way to dynamically plot any graphical representation of the data read by define the data context in the xaml GUI design.
    - Revised the kind of graphical representation required for the read in parameters, and the way they will be represented on the current dashboard.

        - The main dashboard will contain 4 gauge charts to represent the read in Current, Line Voltage, Phase Voltage, and Power Factor parameters respectively.
        - Because of the additive relationship between Active Power, Reactive Power, and Apparent Power, these parameters will be represented as a donut chart: the sum of the 2 series Active Power and Reactive Power is the Apparent Power consumed by the plant.
        - This will also be placed in the main dashboard.
        - These charts will be shown in both the main dashboard, as well as their respective parameters' categories dashboards.
        - Further analysis of the type of graphs required for individual dashboards will be covered in the subsequent weeks.
- 12/10/2024
    - Implement the gauge charts for the Current, Line Voltage, Phase Voltage, and Power Factor parameters using LiveCharts.Wpf:
        - The chart required a single value to bind with its Value property. This will be the parameter that is being represented.
        - The maximum and minimum values are explicitly declared, along with the foreground and background colour, which are set to be the previously used foreground colour for the running text and transparent respectively.
        - The gauge fill colour was also declared explicitly.
        - This type of chart in does not have the property "title" or the equivalent to label the type of chart is being shown, hence a stacked panel was used with vertical orientation to show a text block on the top side of the chart.
        - The chart was created for its respective parameters, and put in an appropriate place in a grid layout:
            - A 2x3 grid for the main dashboard.
            - A 2x2 grid for the rest of the dashboards.
- 13/10/2024
    - Implement the donut chart for the Active Power, Reactive Power, and Apparent Power parameters using LiveCharts.Wpf:
        - The chart required a series of values to bind with the Series property. These needed to be declared in the view model defined in the xaml data context:
            - The PowerVM classes were refactored to include a PowerSeries property of type SeriesCollection, which is a class in the library installed.
            - This was constructed in the View Model's constructor to pass the values of the Active and Reactive Power to the field Value in the subclass PieSeries.
            - The fill colour, border thickness, border colour, and foreground colour were set within the constructor of this class as well.
            - The properties OnActivePowerChange and OnReactivePowerChange, previously used to update the new value for their respective parameters, were also used to update the PowerSeries property.
            - This is because the construction of this property within the class's constructor was only called upon once when initialized, hence a mechanism to update the value within the PowerSeries were required.
        - The size, foreground colour, and legends' location of the chart were declared in the xaml tag PieChart after binding the series with the its property.

        - The inner radius of the PieChart tag was set to 80 to mimic the shape of the donut, finishing the main dashboard with the corresponding charts shown.

Week 5 expected outcomes

- Research on the types of charts required for the Current dashboard, as well as their position on the page.
- Look into the definition and methods to create these types of chart, as well as declaring any required class within the view model to label and bind the data to the charts.

Week 5(04/11/2024 - 09/11/2024)

Completed tasks/trainings

- 07/11/2024
    - Implement the line chart for the Current parameter in its corresponding view, utilizing LiveChart.Wpf's cartesian chart:
        - Define the CurrentValues property within the CurrentVM class to store the current read as ChartValues, which appended when new current values have been read.
        - Bind this CurrentValues property to the Values field within the Series tag of the Cartesian Chart created.
        - Define the TimeElapsed property to count the time at which a new current has been appended to the previous property.
        - Bind this to the Values field of the AxisX tag.
        - Label the axis correspondingly.
    - Implement the bar chart for the Current parameter in the corresponding view:
        - Bind the properties created previously to the corresponding fields.
    - Change the layout of the Current view and put the charts (gauge, line, and bar) in a 3x1 grid, while adjusting the size correspondingly.
- 08/11/2024
    - Fix the bug where the charts only start reading when switching to the view they are put in:
        - This happens due to a new instance of View Model classes is created in the NavigateVM class whenever the operator switch view, making the fields and method within the newly created View Model classes are constructed again in their constructor.
        - This is fine for the gauge charts because they do not rely on the time variables, and because the newly created classes have fields and methods reading from the same Modbus address, the gauge chart showing the same values every time it reset (when switching view).
        - However, since line and bar charts have a time axis, the charts resetting means the entirety of the chart is being reset, making the chart start from 0 second every time the view is being switched.
        - The object-oriented design pattern Singleton was being deployed to combat this problem.

        - Here, the constructor of all View Model classes is being kept private, and these classes can only be accessed through a static public property that create only an instance of that class.
        - This makes any time the classes are being used, the fields and methods of that class all comes from the same instance, rectifying the issue.
- 09/11/2024
    - Collaborating with project's supervisor to remotely connect to a Modbus device, placing in Vietnam, as well as read and write the registers of that device from Australia:
        - This is done through a Virtual Private Network (VPN) that is privately owned and required credential information of the company, who bought the license for software.
        - The remote computer, placing in Australia, need to run the VPN and send and received holding registers from the device through a Modbus pool simulator.
        - The operation was successful to an extent, as the device can send and receive information from the remote computer, which is reflected on the device as well as the computer's display.
        - However, the connection was unstable and major delays observed, which can be improved by changing the network use.

Week 6 expected outcomes

- Research the manual of the physical power monitoring device (PAC3120 and PAC3220) sent by the supervisor.
- Refactor the ModbusMasterModel class to manually implement the Modbus function code x014.
- Change the data structure of the parameters read and displayed in their corresponding classes.
- Develop an algorithm to read the file record that matches that of the actual physical device.

Week 6(11/11/2024 - 16/11/2024)

Completed tasks/trainings

- 14/11/2024
    - Research the manual of the physical power monitoring device (PAC3120 and PAC3220) sent by the supervisor:
        - The device can be requested to send the measured power values (individual line voltage, phase voltage, current, etc.) through Modbus x014 protocol (Read File Record) instead of the anticipated x03 (Read Holding Registers) implemented in the software. This function code is not available in the current Modbus library in C#, hence a manual implementation is required.
        - There are 309 ((617 + 1)/2) registers holding different parameters of different data type that can be requested to send from the device. This arises a need for analysing and determining which parameters can be read and displayed in the EMS software.

        - Theses registers are stored in only the odd-numbered addresses, meaning that there should be an algorithm to read only the odd-numbered addresses registers in the software.
- 16/11/2024
    - Collaborating with project supervisor to perform connection testing between the developed software and the remote mustimeters through a VPN and utilizing Modbus RTU over TCP/IP communications protocol:
        - Set up a new VPN software that allows access to the VPN without needing to share activation key.
        - Briefing the supervisor about the software's functionality and data handling.
        - Modify the ModbusMasterModel class to read from the corresponding addresses and write on the console, ensuring that the values read match with that of the devices.
        - Discuss the way in which the data sent by the devices can be read by the software: the devices are sending a record file (x014 function code), while the current implementation read single holding registers at a time (x03 function code). This appears to not be an issue, as the software can still read the holding registers within the record file sent by the devices.
- 17/11/2024
    - Implement a status text block next to the "Submit" button for slave id. This text block will change colour and read in status to indicate whether the connection attempted to the provided IP addresses is successful.
        - This element will be handled by the ModbusMasterModel and ModbusMasterVM classes, where a function within the former class attempt to connect to the provided IP address as a TCP client, before returning whether that attempt is successful or not.
        - The latter class will call that function and change the content of the text block accordingly.

Week 7 expected outcomes

- Modify the class to read from the required registers in the PAC3022 manual.
- Test the software again the physical hardware that are measuring the power consumption status to verify the feasibility of the design.
- Implement the charts within the voltage page.

Week 7(18/11/2024 - 23/11/2024)

Completed tasks/trainings

- 21/11/2024
    - Change the ModbusMasterModel class to read from the required registers in the PAC3022 manual, which starts at the address 30001 up to address 30085.
        - The properties used to get the reading values from the mustimeters are changed to read values from all three phases and save as a dictionary, where the keys are the symbols denoted in the manual.

        - The corresponding Model classes will parse these dictionaries and save each value into a variable, which can be used by the ViewModel classes to notify the Views and display on the application.
    - Change the layout of the application from grid to scroll viewer with a stack panel embedded to provide more flexibility in placing the charts.
    - There will be three gauge and donut charts corresponding to each line/phase parameters, with labelling and synchronized readings.
    - The line and bar charts, situated in the current view, will contains three series plotted on the same axis, representing the change of current on each phase over time.
- 22/11/2024
    - Analysed the relationship between Apparent Power, Active Power and Reactive Power:
        - They follow the formular: $S = \sqrt{P^2 + Q^2}$, where S, P, and Q denoted Apparent Power, Active Power and Reactive Power respectively.
        - This means that graphically representing these parameters in either a donut chart, stack bar chart, or any charts that represent the relationship a sum and its parts are not suitable, as they do not accurately represent the parameters.
    - Review the proposed graphical representation from the Project Outline:
        - The scatter plot, line charts, and bar charts are too similar to one another, as they all represent the changes of the parameter with respect to time.
        - This means that creating these types of charts separately are redundant as they do not provide any new information to users.
    - These analyses arouse a need for re-implementation of graphical representation, aiming to only shows useful information in a concise and compact manner.
- 17/11/2024
    - Research on the way in which the polar chart can be implemented in the current solution:
        - The proposed solution is to demonstrate the relationship between the Power parameters (such as active, reactive, and apparent power) in each phase with respect to one another.
        - This, however, can also be represented by plotting bar charts of three series in a single pair of axes, rendering the polar chart unnecessary.
        - Furthermore, the library LiveChart.Wpf doesn't contain innate function to create polar chart at version 0.
        - This means that either the entire project is being refactored to use version 2, which is still in beta and might cause some issues with external library (such as SkiaSharp); or use a different library for generating polar chart, which create issues with visual consistencies.
    - Overall, it is decided that the polar chart will not be added, and the graphical representation of the parameters will be limited to line charts (showing changes of values over time, can be implemented in conjunction with bar charts or scatter plot for clarity), and gauge chart (showing the instantaneous values of the parameters in each line/phase).

Week 8 expected outcomes

- Implement the changes for the current, and voltage pages to only include gauge and normal cartesian charts.
- Test the software again the physical hardware that are measuring the power consumption status to verify the feasibility of the design.

- Implement the charts within the power page, as well as change the Power charts in the main page to match the newly proposed solution.
- Consult with the supervisor to figure out which chart types are suitable for the application.

Week 8(25/11/2024 - 30/11/2024)

Completed tasks/trainings

- 28/11/2024
    - Implement the changes for the current, and voltage pages to only include gauge and normal cartesian charts.
        - A custom divisor was added to separate between the lien and phase voltage charts, increasing the visual clarity of the interface.
    - Implement the “running window” property for each chart:
        - The charts will be locked in a time window, where the time in the x-axis will continuously increase but the minimum and maximum will also continuously change so that the window is kept the same.
        - This was implemented by checking whether the current time is larger or smaller than the window size (which is chosen to be 10 seconds):
            - If it is larger, the maximum value is changed to the current time while the minimum value will be the maximum value minus the window size.
            - Otherwise, the maximum value is changed to the window size and the minimum value is kept at 0.
        - This reduces visual cluttering on the charts, while also reducing the computational power required for continuously running multiple charts together.
- 29/11/2024
    - Implement the proposed charts for the Power view:
        - Change the PowerVM class to update values for all power parameters (Active Power, Reactive Power, Power Factor, and Apparent Power)
    - The solution now containing 7 sets of line, scatter, and bar charts running simultaneously, resulting in 21 cartesian charts constantly updating in the background.
    - This creates major performance issues, where switching between views and letting the charts to run for a short while can easily makes the application crashed due to the amount of computation load it has to endure.
    - A way to remedy this is to only update the charts when the corresponding view is being switched into.
        - This means only the charts at that view are updating simultaneously, reducing the computation workload.
    - This solution, however, still creates a variety of issues that needed to be investigate further.
- 30/11/2024
    - Introduce a “refresh timer” field as a potential implementation for the solution, which control the frequency at which the chart update itself:

        - When the view is switched into, this timer starts ticking, updating the chart. It stops when users no longer on the charts' view.
        - The logic controlling when the user switch to the corresponding view is handled by the OnLoaded and OnUnloaded methods in the code-behind.
        - This approach is deeply flawed, as the chart only update the values being read when the view is being switched into, making all the values read during other times being lost.
        - Furthermore, the time axis is constantly updating in the background, making this parameter runs faster than the chart's values, resulting in the chart not updating fast enough when the view is being switched many times.

Week 9 expected outcomes

- Explore another implementation.

Week 9(02/12/2024 - 07/11/2024)

Completed tasks/trainings

- 6/12/2024
    - Implement another approach to the proposed solution is to check whether the view has been switched into or not.
        - If it has, clear the chart values and re-populate it by iterating through a list containing all the values read from the mustimeter since the application start.
        - This approach required the Model and ViewModel classes to be refactored to save the read in parameters into a list, including handling OnPropertyChanges method when adding new elements into the list.
        - It is still not working, however, since the charts does not reflect the changes when adding new elements into the list. This result in the chart only showing a snapshot of the values when switching in, but not dynamically updating.
    - Another solution is to show an empty chart initially, with a button to toggle the chart updating. This means that the chart will only continuously update when user click on the button, saving resources in the process.
- 7/12/2024
    - Continue trying to reduce computational load of the application by unbinding all chart values within the OnUnloaded method:
        - Create SeriesCollection properties for each chart types (line, bar, and scatter) and bind the Series field of the LiveChart's CartesianChart tag with the newly created properties. This allow the binding of chart values to be done in the code-behind.
        - Bind the chart values initially as well as in the OnLoaded method,
        - Set the chart values to null in the OnUnloaded method.
    - Refactor all ViewModel classes to implement the changes.
    - This approach was successful in unbinding the chart values, evidenced by the fact that the charts not getting lagged behind significantly in the Current view.
    - However, the Voltage and Power views still have prominent lags when letting the application run for a while. This implies that LiveChart.Wpf library doesn't have

extensive support for multiple simultaneously updated charts, making views that contains 3 or more charts whose values are dynamically changing every seconds have significant performance issues.
    - This causes the changes to be reverted.
- 8/12/2024
    - Implement a different visual representation of the charts, introducing 3 buttons on the sides of the main chart area to start plotting the parameter values over time, to switch between different types of charts, and to stop plotting.
        - These buttons were styled using similar styling template as the SubmitId button, albeit with different colour scheme and margin.
            - This template was separated to allow for further modification if required, as more buttons with different function might be added.
        - The layout for these buttons was defined a stacked panel embedded within another horizontal stacked panel to place them on the left side of the chart, which is defined as a ContentControl element in the xaml code file.

Week 10 expected outcomes

- Continue to implement the back end of the new visual representation.
- Start research on database saving (Stage 3).

Week 10(09/12/2024 - 14/12/2024)

Completed tasks/trainings

- 12/12/2024
    - Continue implementing the new visual representation:
        - The corresponding commands for these buttons were defined for each parameters chart in the ViewModel classes.
        - These commands were bind with the corresponding methods that defined which charts should be created:
            - The Plot button plot the parameter value over time as a line chart.
            - The Switch button switch between line chart, bar chart, or scatter chart, cycling through these types in that order.
            - The Stop button set the chart to null, stop the rendering of the plot completely.
- 13/12/2024
    - Research how to integrate a database structure to the current application:
        - Energy readings will be saved directly to a database every time a new reading has been detected.
        - A button on each view's section will be added to generate weekly, monthly, or yearly report.
        - The report format will be in the form of a table, outlining the average readings daily in a certain period of time as per user request.
        - There might be an option for exporting the report to a more readable format (excel), however, the details on how to do it and the necessity of doing this is to be determined.

    - Research on the type of database available, including SQLite, SQL Server, and PostgreSQL:
        - SQLite is the easiest to use and to the lightest database structure. However, its capacity and capabilities are generally not scalable to large-scale projects such as Energy Monitoring Software for a large company.
        - SQL Server is a paid enterprise level database structure that contains many useful functionalities such as enhanced security, integration with Microsoft ecosystem, and have built-in data analysis.
        - PostgreSQL is a free database structure for developers, with a lot of basic functionalities and support, as well as being open source, which allows developers to modify any functionalities base on their project.
        - The chosen database for this project is PostgreSQL.
- 14/12/2024
    - The database management software was installed and configured.
    - A table, storing all necessary data read from the mustimeter was created, using a SQL query script.
    - The query scripts for inserting, deleting, and reading from the table was also tested and saved.
    - Add the class EnergyReading to the Core folder of the solution.
    - This class is responsible for getting the instantaneous value of all readings at a specific time stamp, which can then be used in other classes to access the instantaneous value without needing to know about the ModbusMasterModel class.

Week 11 expected outcomes

- Integrate the database to the application.

Week 11(16/12/2024 - 21/12/2024)

Completed tasks/trainings

- 19/12/2024
    - Add the interface IEnergyReadingRepository and the class EnergyReadingRepository to the Core folder of the solution.
    - The class inherit from the interface, which set the expected methods that the class need to have.
    - This approach increases the solution's principle of separation, where the interface can be used in place of the class as a generic repository for unit testing or any other generic usage. The interface can also be used to create other classes that have different implementations of its methods.
    - The class contains two methods: SaveReadingAsync and GetReadingAsync, both utilises C#'s async, which define methods that continuously run asynchronously with the main program in a separate thread:
        - SaveReadingAsync query the database structure, which is passed in as the connection string by the private field, to save an instance of data read from the mustimeters (utilises the class EnergyReading defined previously). This is done by passing each property of the EnergyReading class to the command string.

    - On the other hand, GetReadingAsync query the database to selectively access an entry in the database's table and save that instance in a new instance of EnergyReading.
- 20/12/2024
    - Develop the EnergyMonitoringService class:
        - Inherit from the interface IDisposable, which allow the class to dispose of the unused memories after the continuous async task.
        - The class contains an object of the collection class BlockingConnection<EnergyReading>, where an object of class EenrgyReading can be saved in a separate thread for non-blocking computation.
        - A method adding an EnergyReading parameter to the aforementioned collection after checking whether the computation have been disposed was developed.
        - An async method that run indefinitely, or until the computation is intentionally cancelled, trying to take an EnergyReading from the collection to save into the database (using the IEnergyReadingRepository interface defined in previous week) was developed.
        - This was set to run in the constructor, meaning that the service will be initiated at the moment of constructing the class.
        - This logic was wrapped in a try catch block to catch any error that may arise to ensure smooth operation.
        - Another method (with 2 overloaded implementations) was also added, cancelling the computation queue and disposing of any unused memories.
- 21/12/2024
    - Integrate the previously developed classes with the current application architecture:
        - Add two private fields into the class CurrentVM: a field of interface IEnergyReadingRepository and of class EnergyMonitoringService.
        - These fields were initiated in the constructor of the View Model class.
        - The connection string of the repository field was set to contain the database information such as address, username, password, etc.
        - The data were saved into the repository by creating an instance of class EnergyReading, passing the data into the respective properties within that class, and calling the method EnqueReading with the readings passed in as parameter.
        - These algorithms were initiated any time a current value changed, with each phase handling the logic separately.
    - This implementation result in an exception being thrown, which does not specify what was wrong with the approach, meaning that an investigation was required:
        - The first hypothesis was that the View Model class handling the display logic of the Current view was only responsible for the current readings in each phase.
        - This means that the amount of input parameters to the database table does not match the amount of column the table, resulting in the exception.
        - This can be solved by migrating the implementation to the ModbusMasterModel class, where all parameters were read at the same times (saved in the register field).
        - The algorithm for saving into the repository was handled by the GetReadingRegister method, which is the moment when the data was being received by the application via the RTU over TCP/IP communication protocol.

    - This, unfortunately, also resulted in an exception, meaning that further investigation was required.

Week 12 expected outcomes

- Investigate a solution to the issue.

Week 12(23/12/2024 - 28/12/2024)

Completed tasks/trainings

- 26/12/2024
    - Continue the investigation in the exception being thrown:
        - The implementation of the saving procedures was copied into another Console App program to investigate the component individually.
        - The same exception was still being thrown, requiring further investigation.
        - A SQL database handling for the C# language tutorial was being researched, where a sample code for establishing a simple query to a remote database is detailed.
        - Running that code result in an exception of the database was either does not exist, or the connection is being interrupted.
        - This leads to the second hypothesis: the postgresSQL database structure required a different implementation for database connection; the current implementation was for connection to the SQL database structure provided by the Microsoft ecosystem, meaning that the created table within the postgresSQL database was not the same as the table the application trying to connect to.
        - Potential solutions include change the database structure to that compatible with the current implementation or change the implementation to be compatible with current database structure.
- 27/12/2024
    - Research the way in which postgresSQL database structure can be used with the current implementation:
        - Download the C# library Npgsql via NuGet package manager, which allows the application to connect to a PostgresSQL database.
        - Change the current implementation to use Nqsql connection instead of normal SQLClient connection while maintaining the structure of the code.
        - Create a private property that return the Npgsql connection, containing the connection string that is defined to match the created database table. This allows the connection to be reused without needing to be defined again.
        - The approach allows data, sent through the Modbus RTU over TCP/IP protocol, to be instantly saved into the EnergyReadings table within the custom EMS postgresSQL database.
- 28/12/2024
    - Add the Reactive Power and Apparent Power gauge charts in the main page. This was missing from the 2nd stage of the project.
    - Develop the layout within each parameter view to include the table report as well as the functional buttons/ menus to aid in the creation of the report extracted from the database:

        - In addition to the Plot, Switch, and Stop buttons in the Current View, a Report button was added right bellow where a table report will appear upon clicking.
        - The report needed to have a start date, end date, and the frequency at which each row is taken by; all defined by users via some form of UI on the application.
        - This was implemented as:
            - A textbox for users to input start date of the report.
            - A textbox for users to input end date of the report.
            - A drop-down menu for users to choose between Weekly, Monthly, or Yearly report.
        - These were implemented with the TextBox and ComboBox elements within the xaml file correspondingly.

Week 13 expected outcomes

- Implement the frontend and backend for each newly added element and integrated them to the current MVVM design pattern.
- Test the implementation with the Current View.
- If successful, further implement the same logic with other Views.

Week 13(30/12/2024 - 04/01/2025)

Completed tasks/trainings

- 30/12/2024
    - Continue the implementation for the report creation feature in the Current View:
        - In-between each element (TextBox and ComboBox), it was decided that a TextBox element will be added for clarity.
        - This is because it is unnatural and complicated to have these elements labelled on the TextBox or ComboBox themselves, unlike the Button element as previously implemented.
        - A ToolTip field was added to all the TextBox elements to further instruct users of what to put in. This include the SubmitId text box.
        - The ComboBox and ComboBoxItems are stylized by defining a UserControl file in the Styles folder, where all aspects of the elements are defined and can be reused for all elements of the same types.
        - The elements' colour scheme was kept with the overall application theme.
        - The "chart" area, previously placed in a horizontal stack panel next to the functional buttons area (navigation area), are not reorganized to include another stack panel, with the DataGrid element placed right bellow where the chart was placed.
        - The DataGrid element can be used to display the table report, which will be implemented in the future.
- 31/12/2024
    - Implement the method OnCurrentReportClick() to bind with the click field of the Report button in the CurrentView.

    - The method checks the input strings within the Star Date and End Date to see whether they are in valid format or if they are null, then parse them into a DateTime class using TryParse method.
    - Rewrite the GetReadingAsync() method within the EnergyReadingRepository class to get the readings between two specific dates and with a specific frequency of aggregated data:
        - The PostgresSQL query for each case (weekly, monthly, yearly) listed in the drop-down menu will differ by the frequency at which the data are requested.
        - The function date_trunc() was used to truncate a timestamp to a specific level of precision, which was saved as the variable “period”, which was used to group and order the resulting reader.
        - The reader was then having the value for its columns to be save in the corresponding property within an instant of the EnergyReading class called “readings”, which was ultimately return.
- 1/1/2025
    - The project was reviewed by the supervisor in the end-of-year meeting, where all features of the application’s latest version were explained and demonstrated.
    - Some feedback for the application includes:
        - Make the application able to be maximised as full screen to allow for more space to monitor.
        - Make the report feature able to export the report based on user’s requirements, ideally as an excel file.
        - Adjust the range at which each parameter’s gauge chart is capped at. The current implementation is not realistic in this aspect.
        - Aim to maximise the available space for ease of use and avoid cluttering.
        - Aim to only employ usable charts that fives meaningful information to customers.
        - Mimicking some aspects of the current industrial used EMS applications, especially they layout and functionality.
    - Discuss the application testing methodology with the supervisor:
        - The company currently don’t have a functional energy system that include power source and load to test all parameters that can be transmitted.
        - This means that the final testing can only be done with some parameters, while the rest may have to used simulated data.

Week 14 expected outcomes

- Implement the features suggested by the supervisor.
- Continue to develop the report feature to show the aggregated data from the database in a dynamic grid.
- Change the minimum and maximum limit of the gauge charts according to the realistic values.

Week 14(06/01/2025 - 11/01/2025)

Completed tasks/trainings

- 9/1/2025

    - Add the Maximised App Button on the application's top right corner (where the Exit App and Minimised App Buttons are located).
    - Developed a custom style for this button, using the template provided by the previous twos.
    - Developed a custom method in the main window's code behind to handle the maximising functionality of the button:
        - The method whether the application is in Fullscreen or not by using a flag that get changed based on the state of the application.
        - If the application is not in Fullscreen, the method save the current size of the application in a Rect class, then set the application's left and top sides to 0 and 0 respectively. The application's right and bottom side are then set to the screen's width and right respectively to emulate the Fullscreen effect.
        - If the application is already in Fullscreen, the method set the application sides to the previously saved Rect class (since the default state is not Fullscreen) to return to the original size.
    - Re-ordering the application's elements (such as gauge charts, other charts, and the data grid elements) to fit and make more uses from the space created by being Fullscreen:
        - The elements in the main view are re-oriented so that all the gauge charts are visible when entering Fullscreen mode, with additional spaces underneath for report generation or alarm areas that will be implemented in the future.
        - Other views adopt similar orientation from the main view with regard to the gauge charts (which are transposed to stack each chart for each line on top of one another vertically), making spaces to place the other-charts-plotting area as well as the data grid area on the same row.
        - The divisor border implemented previously to separate the parameter within the voltage and power views are merge with the title to save more space.
        - The gauge charts were having the title for each line/phase removed to save space. Now users can know which line/phase the chart is showing by hovering their mouse on top of the chart.
        - The gauge charts for each parameter are contained within a filled border to enhance visually clarity.
- 10/1/2025
    - Continue to implement the method bind to the Report button:
        - Check whether the input dates have the right sequence (if the star date is after the end date).
        - Create a variable of class EnergyReadingRepository to gain access to the method GetReadingAsync, which return a list of aggregated EnergyReading based on the input star date, end date, and aggregated frequency.
        - Bind the list to the DataGrid element after checking for null variable.
    - Change the GetReadingAsync method within the EnergyReadingRepository to return the average of each parameter instead of their sum.
- 11/1/2025
    - Check the implementation:
        - Randomly generated dummy data for each parameter were inserted into the table by running a looping query on the pgAdmin4's query tool.
        - The query will constantly be generated and inserted into the table as long as the pre-set start time get incremented up to the pre-set end time, which are 01/01/2025 and 12/01/2025 respectively.
        - The increment value was set to 1 minute.
        - The variables were carefully casted to the appropriate data type for this query.

        - The implementation was then checked and reviewed by simply choosing a period of time to extract the data and present them onto the application.
    - The method in the code-behind of the CurrentView was re-written to only extract the current parameters and display on the application instead of the entirety of the table.
    - This was done by creating another list that extract the value of only the specific keys from the full data set, which becomes the list to bind to the DataGrid element.
    - The old implementation was kept for the MainView since it is responsible for all parameters.

Week 15 expected outcomes

- Modify to read data between period of times instead of only an instance of data point.
- Research on the Report Exporting feature.
- Stylize the scroll bar of the application to match its aesthetic.

Week 15(13/01/2025 - 18/01/2025)

Completed tasks/trainings

- 16/1/2025
    - Re-implement the data structure to include the Start_Timestamp and End_Timestamp instead of just Timestamp.
        - This allows the aggregated data presented make more sense, as it is the average data between two timestamps.
        - These parameters will be identical when saving the instantaneous value for the readings.
        - The End_Timestamp will be calculated from the Start_Timestamp when querying the average data points.
    - Remove the ID parameter in the database's table, as it is not working as intended.
- 17/1/2025
    - Enclose the DataGrid element in a ScrollViewer and set the height and width of the element to a fixed value to allows scrolling.
    - This enable the data to be presented nicely without compromising the current structure of the application.
    - Stylize the DataGrid to match the application's colour scheme as well as avoid data cluttering by adjusting the alignment and add padding.
- 18/1/2025
    - Separate the method in the code-behind of the CurrentView to an async Task of GetFullData() and the actual bind method.
    - The former method took star date, end date, and frequency as parameters, and populated the newly declared private list _fullData for the bind method to use.
    - This approach was adopted to increase modularity and reduce repetitive coding when extending the implementation to other views, which were also accomplished in this week task.
    - The lower and upper value for the gauge charts of each parameter was adjusted to match the values provided by the supervisor.
    - The simulated data (both from the Modbus simulation and PostgresSQL data insert query) were also updated to match the values.

Week 16 expected outcomes

- Modify the implementation for the Report feature of the application to match with the MVVM design pattern.

Week 16(20/01/2025 - 25/01/2025)

Completed tasks/trainings

- 23/1/2025
    - Modify the implementation for the Report feature of the application to match with the MVVM design pattern:
        - Add the corresponding Reading classes for each read-in parameters. This allows the readings for each parameter to be shown selectively for the corresponding reading.
        - Change the TextBox element that take sin user input for the start and end date for the report in XAML files to DatePicker, which return the class DateTime in its bidn field instead of string. This removes the needs for parsing the string in the ViewModel methods.
        - Move the async method GetFullData() from the code-behind of the CurrentView files to the ObservableObject class in the Core folder.
- 24/1/2025
    - Continue to modify the CurrentVM class to match with the MVVM design pattern:
        - Move the async method GetCurrentData() from the code-behind of the CurrentView file to the CurrentVM class, along with the StartDate, EndDate, and Freqeuncy properties that will be used to bind to the ItemSource in the XAML file later.
        - The OnCurrentReport() method in the code-behind is removed. Instead, a command was added to the CurrentVM class that is of type AsyncRelayCommand.
        - This class have similar implementation to the RelayCommand class developed earlier in the project, bur with an async Execute method instead.
        - The data flow will be:
            - CurrentView XAML file will send the user-input Date and Frequency information to the CurrentVM class.
            - These information will be passed to the GetFullData() async task within the GetCurrentData() async method to get the full data of the reading in that period.
            - The method is passed to the AsyncRelayCommand as a parameter, meaning that the method will only be called when the XAML element that is bound with that command is clicked.
            - The ObservableCollection (which is an innate class used to store a collection of observable classes) of the custom class CurrentReading will then be populated with the selective column from the FullData.
            - This collection will be bind with the DataGrid to display the data on screen.
- 25/1/2025

    - Continue to implement the Report feature, extended to other parameters instead of only current:
        - Add the necessary parameters for each reading (PhaseVoltageStartDate for Phase Voltage, etc.).
        - Change the class type of the FullData property to ObservableCollection<EnergyReading> instead of IList<EnergyReading> for more flexibility.
    - Re-organize the folder structure:
        - Rename the ObservableObject class to BaseVM class and move it to the ViewModel folder. This is because the added implementation make the class a lot more complicated than just an "Observable Object".
        - Move the RelayCommand and AsyncRelayCommand class to the Command folder located in the Core folder.
        - Move all the properties/fields that are necessary for the RunningWindow implementation to the BaseVM class as all its children classes use the same value.

Week 17 expected outcomes

- Implement the Export Report feature.
- Styling the new elements that have been added (scroll bar, date picker).

Week 17(27/01/2025 - 01/02/2025)

Completed tasks/trainings

- 30/1/2025
    - Develop a method to export the data grid shown when generating report to a CSV file for better visualization and presentation:
        - The method took the chosen start date and end date within the date picker elements when pressing the Report button and convert them to string under a specific format (dd_MM_yy).
        - These strings will be concatenated with the type of reading (whether it is current, line voltage, or phase voltage etc.) to make up the exporting file path.
        - The first row of the CSV file will always be the header, which is appended at the start of the StringBuilder variable.
        - For each reading in the readings generated, the variable will be appended with a row (which contain all the readings for that instance).
        - The variable will the be written into the newly generated file, and a message box will notify users about the exporting status.
    - This method is passed into the ExportReportCommand, which is then bind with the Export button in the View files.
    - The Export button will only be visible when the Report button is pressed, which will be controlled by a flag that turn true only when the LoadExport command is initiated.
    - The same method and logic are applied to all ViewModel files:
        - The Main page will not be implemented with these changes, as there will be a major change when implementing the Waring feature.

    - For each button in the View files, a ToolTip field will be added, instructing users on what to do. This will avoid confusion and increase functional clarity.
- 31/1/2025
    - Styling the DatePicker elements:
        - Create a custom style for the textbox element (which stored the chosen date) and for the calendar button (in which the date can be chosen when pressed):
            - The textbox element uses similar style to the SubmitId textbox, albeit with different text block and sizes to fit with the surrounding elements.
            - The calendar button element uses an icon in the Assets folder to distinguish itself from the others, along with a border element to customize the sizes and background colours.
            - Change the DatePicker's field to have today's date automatically picked for faster user interaction.
    - Modify the StartDate and EndDate properties within each ViewModel class to be nullable (by making the corresponding changes within the IEnergyReadingRepository and EnergyReadingRepository) and assign these properties to null values in the constructor. This way, the text block in the DatePickers' text block will be visible at the start.
- 1/2/2025
    - Styling the ScrollBar elements in the ScrollViewer:
        - Divide the scroll bar area into 3 distinctive areas: the up/down buttons and the middle area:
            - The up/down buttons were stylize using similar techniques to the minimize/maximize buttons previously developed, albeit with an icon as the foreground.
                - The corresponding commands are bind with these buttons.
            - The middle part is further divided into 3 parts: 2 tracking areas and 1 thumb (which allow users to control where the pages are):
                - The tracking areas are stylized by setting the necessary properties (such as IsTabStop, Focusable, or SnapToDevicePixel) as well as set the background to transparent.
                - The corresponding commands are bind with these parts.
                - The thumb is stylized by setting similar parameters as well as the element's colour when users' mouse is over.
        - The vertical and horizontal counterparts of this element are developed alongside one another, with the horizontal element have its orientation rotated (up/down to left/right, row to column, width to height).
    - This finalized the third stage of the project.

Week 18 expected outcomes

- Start implementing the Warning and Recommendation features (stage 4 of the project).

Week 18(03/02/2025 - 08/02/2025)

Completed tasks/trainings

- 5/2/2025
    - Research the Warning feature in current industry-issued EMS software:
        - The application will check the read-in parameters value and compare them with pre-defined threshold values.
        - If the read-in values are higher or lower than the upper or lower boundaries respectively, the application will show a warning, prompting the users to address and acknowledge the issue.
        - Users can set the threshold values in the setting/ config file.
        - Based on the content of the Warning, there might be an action recommendation features, which will be implement in the future.
    - Prepare a general plan for the initial implementation of the feature:
        - The values comparing logic will be added as a method in the Model classes, which also contains the corresponding fields and properties.
        - These properties will be passed into the ViewModel classes via event properties, which will be used to bind with the corresponding tags in the View files.
        - The ViewModel classes also responsible for controlling the warning acknowledgement feature, where a command will be created for users to acknowledge the warning popping up, bringing them to the background instead of the focus of attention.
- 6/2/2025
    - Implement the Warning feature in the Model classes based on the plan proposed:
        - The CurrentModel class will contains fields and properties related to the feature such as LowerThreshold and UpperThreshold, both of which will be hard coded to be 20 and 80 respectively in the constructor.
        - It also contains the WarningTriggered event, which will only be invoked if the current value in each line is lower or large than the threshold values. This will be controlled by a method that get called in the respective current properties.
            - This means that the checking process will happen at the moment of receiving the current value from the multimeter via Modbus RTU.
- 7/2/2025
    - Implement the Warning feature in the ViewModel classes based on the plan proposed:
        - Add the same WarningTriggered event in the CurrentVM class that get invoked when the events in the CurrentModel class get invoked, passing the same arguments to the new events.
        - It was decided that the warning section will only be visible in the MainVM section, where users can see and address or acknowledge all parameters' warning at the same time instead of having to go to each page to see the warning.
            - Future improvement may include showing the warning for the respective parameters in their page that are sync with the main page.
        - This requires the implementation to be further extended to the MainVM class, where an Observable Collection of string will be saved and populated with warnings passed from the triggered event of the respective ViewModel classes.
        - A method was also added, responsible for adding the warning string (as the parameter) into the collection. This will be called and have the warnings tring

passed into its argument whenever the Warning Triggered events from the CurrentVM class are invoked.

Week 19 expected outcomes

- Continue Warning feature implementation.
- Format the warning string to gives more details about the status of the monitoring parameter.
- Add some visual effects to the gauge charts when the value is higher or lower than the threshold values.

Week 19(10/02/2025 - 15/02/2025)

Completed tasks/trainings

- 13/2/2025
    - Continue to implement the Warning feature in the ViewModel classes:
        - The Observable Collection that is being populated constantly with warning string will be bind with the XAML ListBox element.
        - This implementation causes the exception 'System.NotSupportedException' in PresentationFramework.dll to be thrown, which typically occurs when the class is modified from a different thread than the UI thread.
        - This requires a Dispatcher class field to check the accessibility of the threading behaviour before adding the warning string to the collection.
- 14/2/2025
    - Developed the WarningModel class to replace simple string messages:
        - The class will include the properties: Message (string), Timestamp (DateTime), and Category (string).
        - Modified the WarningTriggered event to accept WarningModel as its data type instead of string.
        - Integrated WarningModel into parameter models for consistent data handling.
    - Added the Reading property of type EnergyReading to the ModbusMasterModel class:
        - Initialized the Reading field in the class constructor.
        - Assigned properties to Reading within the GetReadingRegister method.
        - Utilized Reading's Timestamp for generating warnings across different parameter models.
    - Refactored data transfer between MVVM layers:
        - Replaced creating new EnergyReading instances with assigning values to the existing Reading field.
        - Improved data consistency and efficiency during MVVM communication.
- 15/2/2025
    - Implemented dynamic warning ListBox styling based on the warning category:
        - Assigned corresponding Category (High, Low, Normal) in the CheckThreshold method.
        - Updated the XAML View file to bind the Category property for dynamic styling.
        - Replaced simple text display with rounded boxes in the ListBox.
        - Applied conditional formatting to change the background and border colours based on the warning category.

    - Standardized colour scheme using Flat Design Color Chart from https://htmlcolorcodes.com/color-chart/:
        - Selected reddish tones for high warnings, bluish for low warnings, and neutral yellowish for normal status.
    - Initialized the colour palette in the BaseVM class to ensure consistency and reuse across View Models.

Week 20 expected outcomes

- Implemented the Acknowledgement system and reflected that on the front-end UI.
- Start researching the custom thresholds system.

Week 20(17/02/2025 - 22/02/2025)

Completed tasks/trainings

- 18/2/2025
    - Enhanced gauge chart colour coordination with the warning ListBox:
        - Passed the Category string to a dedicated method in CurrentVM to update the StatusColor property.
        - Bound StatusColor to the GaugeFill property of each gauge chart for dynamic colour updates.
    - Addressed colour update delays observed in the initial implementation:
        - Refactored methods and optimized methods call to ensure real-time colour changes.
        - Conducted iterative testing with various code adjustments to achieve seamless synchronization.
        - Improved performance by reducing redundant method calls and enhancing property change notifications.
    - Final adjustments and testing:
        - Validated colour transitions across different warning categories.
    - Ensured consistent behaviour of the acknowledgment system and real-time UI updates.
- 19/2/2025
    - Developed the Acknowledgement sub-system for the Warning system:
        - Users can click on the warning to acknowledge that they are aware of the issue.
        - The acknowledged warnings will have a greyish background to indicate that the warnings are inactive.
        - This is tracked by another property in the WarningModel class, whose default value is false to indicate that warnings generated is not acknowledged unless users click on it.
        - Another command is developed in the MainVM class, taking the AcknowledgeWarning method as its parameter.
        - This method can be used to acknowledge a passed in warning and force the UI to update this change in warning acknowledgement status by removing and adding the acknowledged warning in the same place in the active warnings collection.
        - The command is bind to the action “mouse left click” on the designated border that the ListBox element occupied.

        - The change in colour is reflected by binding the Boolean properties IsAcknowledged to the condition in which the ListBox have its background colour changes from red/blue to grey.
- 20/2/2025
    - Expand the Warning system to other parameters:
        - Add the WarningModel properties, WarningTriggered events, and thresholds fields to the VoltageModel and PowerModel classes for each parameter and line/phase.
        - Set the thresholds for each parameter in their respective constructor.
        - Checked the read-in values for each parameter on each line/phase and assign the corresponding warnings to the WarningModel properties.
        - Add the status colours and WarningTriggered events to the VoltageVM and PowerVM classes for each parameter and line/phase.
        - Subscribed to the same events from the model in the view model classes to ensure both layers are aware of the changes in property.
    - Ensure that the warnings only triggered if there are a device connected to the application.
    - This is done by calling the IsDeviceAvailable method before assigning values to the warnings via calling CheckThreshold method.

Week 21 expected outcomes

- Implement a setting page where users can put their own threshold values for each parameter.
- Add other parameters thresholds in the setting page.
- Rearrange the elements in the SettingView to look more appealing.

Week 21(24/02/2025 - 01/03/2025)

Completed tasks/trainings

- 27/2/2025
    - Developed the Setting page, where users can change the lower and higher thresholds of each parameter based on their desire:
        - Add the SettingModel and SettingVM classes, with the former containing all relevant information on the page main features (lower and higher thresholds of all parameters) and the latter communicate these information to the view XAML file and vice versa.
        - Add the SettingView file, where users can input their desire thresholds setting for the warning.
        - Create another command in the NavigateVM class that change the CurrentView to Setting whenever triggered.
        - Add the SettingVM as the data context for the SettingView while also ensuring the view model is use as a resource for the app.
- 28/2/2025
    - Developed the SettingVM class:
        - Add all the properties that will be changed by users such as lower or higher thresholds for each parameter.
        - Ensure these properties are accessed and changed by changing the corresponding properties in an instance of SettingModel, which was developed to already constructed with default properties set.

        - Add a method to parse the input string to double and save to a JSON setting file. This ensure that the custom setting set by the users will be saved and can be used again.
        - The setting saved in this JSON file will be read and apply in the ModbusMaterModel class via another method that is called in the constructor. This ensures that the setting is used globally throughout the application.
        - This approach requires the SettingModel instance to be changed when new settings is input by users, which does not work as the instance remain static even after the change.
        - This makes the settings input only applied when the application is reset.
    - Fix the bug where the users-input settings only apply when the application is reset:
        - Refactor the Instance property to only create the instance within the lock statement.
        - This ensure that a specific thread is used to create that instance.
        - Add the UpdateInstance static method, where the instance creates dis updated within the same thread as when it was created.
        - Change the SettingVM and ModbusMasterModel classes to use the method to update the instance after the instance has been created.
- 1/3/2025
    - Add other parameters thresholds in the setting page.
        - Add the string property for each parameter's threshold in the SettingsVM class.
        - Add the double-to-string parsing logic in the SaveSettings method.
    - Rearrange the elements in the SettingsView to look more appealing.
        - Encapsulate the content of the page into a two-columns-grid for better visibility.
        - Add the division border similar to that from the normal Voltage/ Power pages to highlight and divide each parameter's section.

Week 22 expected outcomes

- Develop the Acknowledge All function.
- Set the minimum and maximum values for each parameter's gauge charts in the SettingModel class for a more centralized design.
- Add a way to check whether the thresholds value for each parameter is valid (are they within the minimum and maximum range).

Week 22(03/03/2025 - 08/03/2025)

Completed tasks/trainings

- 6/3/2025
    - Develop the Acknowledge All function.
        - Add a method to iterate through the ActiveWarnings collection and acknowledge them whenever called.
        - The method is pass into the command of the same name, which was bind to a button bellow the warning area to acknowledge all active warnings whenever pressed.

        - Add a CanAcknowledge field and property to the MainVM class, where it only changed to true when there is at least a warning being added to the collection.
        - The property is bind to the visibility of the Acknowledge All button, along with the Boolean to Visibility converter, copied from the parameter views.
- 7/3/2025
    - Allow the minimum and maximum values for the gauge charts to be set in the SettingsModel:
        - Add the properties for the minimum and maximum values for each parameter in the SettingsModel.
        - Set default values for these properties in the constructor.
        - Create an instance of the SettingsModel class in each parameter View Model classes:
            - The minimum and maximum values for the parameters do not change as per user input, hence the view model classes use an instance of the SettingsModel instead of SettingsVM.
- 8/3/2025
    - Add properties for each parameter's maximum and minimum values in thr View Model classes to allow binding with the "From" and "To" fields.
        - This allows for a more centralized design, where the properties that are easy to be changed when using on a regular basis can be created and changed in one place only.
        - It also allows for more features where users can freely change the minimum and maximum values their multimeter can read and send on the UI's setting page.
    - Checking the validity of the users-input lower and higher thresholds in their respective setter.
    - This marks the end of the final development stage of the application, with all the features required implemented.
        - The project is now ready for testing and reviewing.

Week 22 expected outcomes

- Test and review the project.

Week 23(10/03/2025 - 15/03/2025)

Completed tasks/trainings

- 13/3/2025
    - Finalized and cleaned up the code base:
        - Rearranged the components in each class to match OOP order of declaration: fields, constructor, events, properties, commands, functions.
        - Added comments for each block of code, classifying the codes for each feature in the application: variables, charts, reports, and warnings.
    - Fixed a database bug:

        - Reassigned the Data of the PostgreSQL folder to another disk. This ensure that the database can be connected to, which was a problem when moving folders to a newly bought disk.
        - Changed database password to 12345678. This is to prevent personal information contained in the old password to be seen by malicious parties when uploading onto online spaces.
- 14/3/2025
    - Tabulated all implemented and unimplemented features:
        - Implemented:
            - A GUI with styling.
            - Read data from a remote Modbus device (slave) with changeable slave id.
            - Parse the data read into a specific format defined in the PAC3120 user's manual.
            - Display the data read in either gauge charts or a choice between line, bar, or scatter charts.
            - Save the data read in a local PostgreSQL database.
            - Generate, display, and export a report of the aggregated data read based on start date, end date, and frequency (weekly, monthly, or yearly).
            - Generate a warning on the GUI whenever the data read exceeds a threshold.
            - Allow users to set the threshold for warning in a setting page.
        - Unimplemented:
            - The ability to change the IP address of the slave.
            - The ability to connect to and monitor multiple slaves at the same time.
            - The ability to connect to a remote database server, which affect the scalability of the application.
            - The ability to alert users via email or phone number whenever a warning is generated.
            - The ability to dynamically changed the maximum and minimum values of a parameter presented on the gauge chart.

Week 24 expected outcomes

- Identify the scopes and future considerations for the project.
- Prepare to present to supervisors the finalized application.

Week 24(17/03/2025 - 22/03/2025)

Completed tasks/trainings

- 20/3/2025
    - Identified the scope achieved, which is an application that can communicate to one other Modbus slave via a static IP address and slave ID, read and present the data read as charts, export reports, and generate warnings whenever a parameter's value is larger or smaller than the upper and lower threshold respectively.

    - Identified potential future development of the application:
        - Allow for simultaneous slave's connection, which also allow easier power consumption comparison between different plants.
        - Add the ability to change slave's IP address in the setting page.
        - Add the ability to change the application's window dimension in the setting page.
        - Add different languages and the way the application can change its language in the setting page.
        - Utilize more icons for the buttons, which increases UI simplification and better recognizability.
        - Implement a feature to send warnings to users via email or phone messages for better alert.
        - Implement a dynamic maximum and minimum parameter values update based on the read-in data.
        - Incorporate a better database structure for the application, which helps scalability.
        - Incorporate AI features that predict power consumption and provide more accurate suggestions.
- 21/3/2025
    - Present the application to supervisor,
    - Collaborate with the supervisor to test the application against the devices installed in Vietnam.
    - The application can establish a connection with the remote device, and the graphing and reporting functions work as intended.
    - The supervisors provide some feedback:
        - The current implementation is not accurate to what expected, which is an application that can read and compare multiple Modbus devices at the same time.
        - The database is localized and not suitable for large-scale implementation.
        - However, the features implemented are usable in some scenarios and can be used as a basis for further development.

Week 25 expected outcomes

- Upload the application onto GitHub.
- Write the README file for how to run the application.
- Start writing the summary of reflective journal document.

Week 25(24/03/2025 - 29/03/2025)

Completed tasks/trainings

- 27/3/2025
    - Create a new folder called Energy-Management-System.
    - Copy and paste the EMS folder, the sample simulation file, the SQL scripts, as well as all documents relevant to the project into this new folder.
    - Create a new GitHub repository at this folder using GitHub desktop.
    - Initiated a README file included:

        - Brief description of the project.
        - The implemented features.
        - Some apps screenshot.
        - Step by step instructions of how to run the application.
- 28/3/2025
    - Started writing the Host Organization section in the Summary of Reflective Journal document:
        - Introduced Saigon Electric Company Limited.
        - Introduced the project undertaken.
        - Introduced what was assigned.

Week 26 expected outcomes

- Finish the summary of reflective journal document.

Week 26(31/03/2025 - 05/04/2025)

Completed tasks/trainings

- 3/4/2025
    - Wrote the Swinburne Engineering Competencies section:
        - Analysed each competency against the work done.
        - Identified how each stage and features implemented in the application related to the engineering competencies required.
- 4/4/2025
    - Continue to write the Swinburne Engineering Competencies section.
    - Wrote the Influence on the future direction of your career section:
        - Reflected on what have been done, and what was learned during the internship.
        - Reflected on personal career preference and how the internship have affected that preference.
    - Wait for the completion letter from the supervisor.